Journal of Ambient Intelligence and Smart Environments 8 (2016) 125–148
DOI 10.3233/AIS-160370
IOS Press

A comparative study of systems for the design of flexible user interfaces

Christopher Mayer [a,*], Gottfried Zimmermann [b], Andrej Grguric [c], Jan Alexandersson [d], Miroslav Sili [a] and Christophe Strobbe [b]

[a] *AIT Austrian Institute of Technology GmbH, Health & Environment Department, Biomedical Systems, Donau-City-Str. 1, 1220 Vienna, Austria*
E-mails: christopher.mayer@ait.ac.at, miroslav.sili@ait.ac.at
[b] *Stuttgart Media University, Responsive Media Experience Research Group, Nobelstr. 10, 70569 Stuttgart, Germany*
E-mails: gzimmermann@acm.org, strobbe@hdm-stuttgart.de
[c] *Research and Innovations Unit, Ericsson Nikola Tesla d.d., Krapinska 45, 10002 Zagreb, Croatia*
E-mail: andrej.grguric@ericsson.com
[d] *German Research Institute for Artificial Intelligence, Campus D3 2 Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany*
E-mail: jan.alexandersson@dfki.de

Abstract. There are a number of Information and Communication Technology (ICT) services that can be used to increase quality of life of older adults and persons with special needs. Unfortunately, there is often a mismatch between the offered services and their user-friendliness, hindering their use and reducing their utility. One of the most important factors is the ability to provide accessible, attractive and user-friendly interaction. In this study, three systems for the design of flexible user interfaces are compared based on a generic framework. The framework introduces a multi-step adaptation process, the concept of context of use and the distinction between adaptable and adaptive user interfaces. The three systems – AALuis, GPII/URC and universAAL – are independently presented and compared by means of ten framework-related criteria. The analysis of the systems reveals similarities (e.g., context of use, resource provision, and commitment to standards) and differences (e.g., abstract description of the user interfaces, generation of user interfaces or pluggable user interfaces and maintenance of the user model). This paper shows for each system where it is best suited in terms of application areas and desired system features. There is a need for further research to work towards harmonization between the systems and their application areas so that they may mutually benefit from each others' strengths.

Keywords: User interaction, user interface generation, AALuis, GPII/URC, universAAL

1. Motivation

The research field of user interface design has emerged with a large community, several journals and regular conferences. Well-designed interfaces have been identified as a driving force behind business success. There have been numerous examples of bad design resulting in technology being discarded or even labeled as erroneous, when in fact the problem is that the consumers have just failed to understand its functionalities. In one study, it was shown that half of all "malfunctioning products" returned to stores by consumers are in full working order, but that customers cannot figure out how to operate the device [7].

Older adults constitute a challenging user group with special needs and abilities. This is a heterogeneous group with rapidly changing characteristics and requirements, as ageing may bring impairments in vision, hearing, mobility, dexterity and cognitive skills

*Corresponding author. E-mail: christopher.mayer@ait.ac.at.

in a higher degree. Older adults are vulnerable. Today, there is often a mismatch between available services and their user-friendliness, which can reduce the usage of many services from which older people could benefit. The success of these services depends on their ability to provide user interaction that is accessible to their users.

To increase the benefits for older adults using Information and Communication Technology (ICT)-based products and services, developers and service providers must not only focus on the functionality itself but also, even especially, on the user interface. An appropriately and correctly designed user interface will guide the user to understand the underlying service – a vital precondition for the service's intended usage. A fundamental prerequisite of this is the personalization or adaptation of user interaction to user needs. Individuals need accessible user interfaces for their preferred ICT devices and services.

The purpose of this paper is to provide a generic framework for designing and deploying user interfaces that are tailored to particular user groups. In the light of this framework, three concrete systems – AALuis, GPII/URC and universAAL – are analyzed and compared. These systems have been chosen because of the authors' involvement in them. However, it is claimed that they represent a wide variety of current systems for the design of flexible user interfaces, and can therefore be considered as typical. The study was initiated by an intensive face-to-face workshop which focused mainly on the technical aspects of user interfaces and their generation. A typical application area for these systems is Ambient Assisted Living (AAL), but they are not restricted to this domain and are rather generally developed for the design and generation of flexible user interfaces.

The remainder of this paper is structured as follows: In Section 2, one finds a short description of related work regarding conceptual frameworks for technical comparisons. Section 3 introduces some concepts and terms, and defines a generic framework for the design of adaptive user interfaces, providing a basis for the comparison of systems. Section 4 describes the three selected systems, focusing on the design of adaptive user interfaces. In Section 5, the criteria for the comparison of the systems are given followed by a detailed analysis of the three systems in light of these criteria. In Section 6, the results of the qualitative comparison are summarized and discussed. Finally, Section 7 provides conclusions and elaborates on possible extensions and combinations of the systems.

2. Related work

Comparing different systems in a holistic way is a challenge, since various aspects need to be considered. These include both technical and user-oriented aspects. This paper focuses in particular on the technical potential and system features. It does not cover user aspects such as usability and acceptance from the end-user point of view, nor the ease of use for developers, although both of these are important issues on the one hand for the acceptance of user interfaces and on the other hand for development costs.

There do exist conceptual frameworks available for the technical comparison, but these focus just on particular aspects to be considered for user interface adaptation. Examples of these from the literature are as follows: (a) models as basis for the user interface generation [16,21], (b) methods used for the user interface generation [38], (c) tools supporting the generation process [23], (d) the final user interface [34], or (e) multi-modality [5,15]. These aspects are important, but do not cover the whole user interface adaptation process as described in Section 3 and hence do not fulfill our intended purpose. Thus, more specific comparison criteria have been formulated, taking the state-of-the-art into account. Additionally, the Cameleon Reference Framework (CRF) [3], which defines four levels of abstraction for user interface models, is used for a final summary of the three systems in Section 4.4. It is again a very useful framework for the comparison, but does not cover the whole process as described in Section 3.

3. Introduction to user interface adaptation

User interface adaptation covers different stages in the design and lifecycle of user interfaces, and there are various terms, concepts and aspects to consider.

3.1. User interface adaptation aspects

User interface adaptation can change any aspect of a user interface. Figure 1 illustrates some adaptation aspects and organizes them within our 3-layer model, which breaks a user interface down into layers for *presentation & input events*, *structure & grammar*, and *content & semantics* (ordered by increasing depth). Adaptation aspects range from *shallow* modifications (e.g., text size, line spacing, contrast settings, speech volume, keyboard settings, shortcut keys) related to

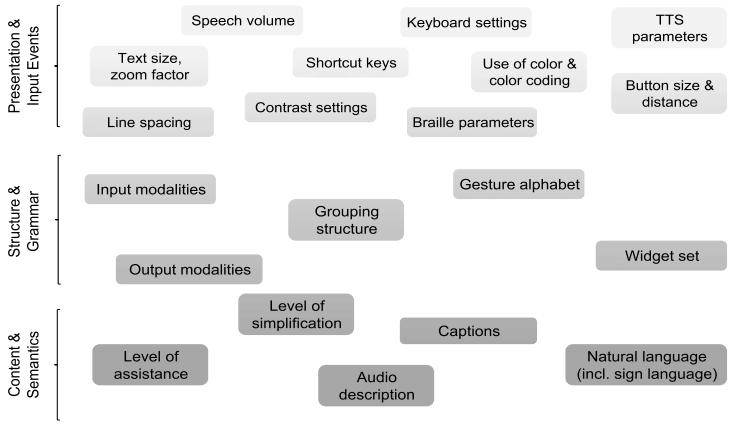

Fig. 1. Some aspects of user interface adaptation, depicted in a 3-layer user interface model.

the presentation & input events layer, over *medium deep* modifications (e.g., input and output modalities, grouping structure, gesture alphabet) related to the structure & grammar layer, to *deep* modifications (e.g., natural language, captions, audio descriptions, level of simplification) related to the content & semantics layers of a user interface.

3.2. Context of use

In the process of adapting a user interface, at least one of the three user interface layers (see Section 3.1) is adapted to a specific *context of use*, consisting of the following main components [3]:

- **User:** The user interface needs to be tailored to the user's needs and preferences, which may be dependent upon his/her capabilities, current physical activity, permanent and temporary impairments, moods, etc. Some of these needs and preferences are dynamic, i.e., can change based on time and situation, even during a user interaction session.
- **Runtime Platform:** Each runtime platform comes with its own hardware and software constraints. Hardware constraints include, for example, screen size, screen resolution, and support for pointing (e.g., mouse, D-pad, touch screen). Depending on the underlying operating system and Web browser, software features such as specific fonts, runtime libraries, and scripting support may or may not be available at runtime.
- **Environment:** The environment and situation in which the use occurs can significantly impact usability and accessibility aspects of a user interface. Examples include ambient noise, lighting conditions or the time of day.

3.3. Adaptable versus adaptive user interface

A system is said to have an *adaptable user interface* or to be *adaptable*, when it allows the user to customize the user interface [29]. A system is said to have an *adaptive user interface* or to be *adaptive* when it adapts the user interface by itself or suggests such adaptations to the user at runtime [29], based on a user model and other context-of-use data [24]. If the system only remembers previous customizations made by the user, this would not be called an adaptive system, but rather only an adaptable system. A system can be hybrid, i.e., provide both adaptability and adaptiveness. For example, if the user increases the font size, the system may ask the user if he/she also wants to switch to a high-contrast theme.

3.4. A framework for user interface adaptation

Figure 2 illustrates the process of user interface adaptation in general, spanning design time and runtime. At design time, the application developer creates an abstract user interface that leaves some leeway with regard to its final rendering (see Section 3.5 for a more detailed description of the adaptation process and its actors). The format of the abstract user interface varies significantly between technologies. Some technologies employ task models, some use abstract forms of widgets and controls, some state diagrams, and some employ semi-concrete user interfaces that can be parameterized at runtime. Of course, these approaches can be combined as well.

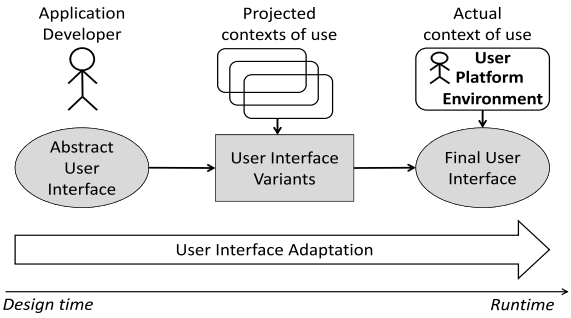

Fig. 2. The general process of user interface adaptation, spanning the overall development and rendition process from design time to runtime.

Between the design of the Abstract User Interface (AUI) and usage at runtime, user interface resources may be prepared in advance by user interface designers or automatic systems (or both). At this point, assumptions (projections) are made for the context of use at runtime. The goal of this activity is to have some building blocks of a Concrete User Interface (CUI) "in store" that would otherwise be impossible for generation at runtime.

At runtime, when the user interacts with the system, the actual context of use is known, and the Final User Interface (FUI) is built as an instantiation of the abstract user interface. If user interface integration is involved, the final user interface employs suitable user interface resources that have been prepared before runtime. As a possible complication, the context of use may change during runtime, and the concrete user interface may adapt immediately to the change.

User interface adaptation is a complex process typically occurring at both design time and runtime. Each of these time points has its own opportunities and constraints. At design time, a human expert can create sophisticated user interfaces with variations for adaptation. However, designers do not know the exact context of use that will occur at runtime. Therefore, they have to make guesses and provide internal (i.e., built-in) or external (i.e., supplemental) variations that will be selected at runtime. At runtime, the context of use is available, and can be used to pick and/or tailor an appropriate user interface. However, the user interface must be available immediately for rendition, therefore human design is out of reach at runtime.

3.5. Steps of user interface adaptation

Based on the general process of user interface adaptation (see Fig. 2), the following typical steps were identified as relevant for the adaptation of user interfaces, each occurring at some point between design time and runtime, inclusive (Note that some design approaches only support a subset of these steps):

1. **User interaction modeling:** At development time, a developer implements an application and describes its functionality related to the user interaction in an abstract fashion. Forms of *abstract descriptions* are, for example, task models, user interface models, Web service interface descriptions, descriptions of an Application Programming Interface (API), or ontologies.
2. **Default user interface design:** A user interface designer implements a default user interface for the application, based on the abstract description of its functionality (see step 1). Along with the user interface, components such as icons, labels and videos are designed. The user interface and these components are collectively called *user interface resources*.
3. **Supplemental user interface design:** User interface designers (either from the manufacturer of the application, or from third parties) contribute alternative user interface resources, based on the application's abstract description (see step 1). These resources are supplemental to the default resources (see step 2), and may include alternative user interfaces (or parts thereof), e.g., a simplified version, that can be used to substitute or extend the default user interface (or parts thereof). Each user interface resource can be tailored to a particular context of use, namely a specific user group, runtime platform, environmental condition, or any combination thereof. It must have appropriate meta-information (e.g., key-value pairs) attached, to facilitate its integration in the final user interface at runtime when a specific context of use is known. A local or global *resource repository* stores the supplemental user interface resources and their meta-information.
4. **Context of use instantiation:** At runtime, the runtime platform instantiates the particular context of use, consisting of a *user model* (which may be retrieved as a whole or in parts from a user model repository), a *platform model* and an *environment model*. These models may be described in various formats, including key-value pairs or formal ontologies. Note that the context of use can dynamically change during runtime.

5. **User interface accommodation (system-driven):** An adaptive system adapts the concrete user interface to accommodate the particular context of use at hand, i.e., finding the best matching user interface for the actual context of use.[1] Two types of adaptations are distinguished (and may co-occur): user interface integration and user interface parameterization. *User interface integration* means that a set of matching user interface resources (e.g., dialogs, icons, labels, videos, captions) is chosen to form the user interface. *User interface parameterization* means that the user interface is "tweaked" along pre-determined parameters (e.g., font size). This step can be a continuous process with subsequent iterations of refinements, perhaps reacting to a dynamically changing context of use during a user interaction session. Both forms may ask the user to confirm suggested adaptations.
6. **User interface customization (user-driven):** When interacting with an adaptable system, the user can take the initiative for adaptations by *customizing* the user interface via explicit operations (e.g., through a settings dialog, a menu or shortcuts). These operations may be part of the runtime platform, or may be directly built into the user interface. As in step 5, a customization can be realized as user interface integration or user interface parameterization, or both. User-initiated customizations can be recorded (e.g., in a user model) to make them permanent for the user and the particular context of use.

This process can vary depending on the overall development approach and toolset used, the user interface quality requirements, the availability of user interfaces, the availability of use context parameters and other influences. For example, the application could be monolithic, with its user interface and business logic lumped together. In this case, the developer would develop the application's user interface model (step 1), the default user interface (step 2) and (optionally) supplemental user interfaces (step 3). Also, rather than having a human designer create a user interface (steps 2 & 3), it could be generated automatically based on the abstract description and other resources. In this case, the design of user interfaces could be deferred until runtime, when the exact context of use is known, and a resource repository (step 3) might not be needed, since the "best-matching" user interface would be generated on demand in real-time. Furthermore, user interface accommodation and customization may be combined as a mixed dialog, i.e., steps 5 and 6 can occur any number of times in an interwoven fashion.

[1] Matching approaches are out of scope for this framework, but may involve automatic inference mechanisms, a user's explicit choice, or a reactivation of a previous user choice.

4. Introduction to the selected systems for user interface adaptation

The three systems – AALuis, GPII/URC and universAAL – are respectively described in the following sections and finally summarized with respect to the user interface levels of the Cameleon Reference Framework (CRF) [3] (see Section 4.4). The framework is used as a joint overview of the three systems and as a wrap-up of the bottom-up approach.

4.1. AALuis

AALuis is a framework mainly focusing on the user interface generation [18] and covers all steps of user interface adaptation as described in Section 3.5. Figure 3 illustrates components of this framework together with the information flow from the calling service/application to the end user and back. It covers the service integration, the user interface generation and the user interface rendering. The information flow starts with the connection between the digital service and the task model, which models the interaction between the user and the system (step 1 in Fig. 3). Each task describes the logical activities that should support users in reaching their goals [31] and each task is either a user, system, interaction or abstract task [33]. Next, in light of the task model, model interpretation yields a set of currently active interactions (step 2), and the user interface is transformed and adapted (step 3). Finally, the interaction between the user and the functionality of the service completes the information flow (steps 4–6).

The framework can either be used with local or remote services. The former are integrated as separate OSGi bundles and the latter are loosely coupled as Web services, which allows an easy connection of any platform to the system. Services are described in a twofold manner by two separate files: the first defines the task model of the service in Concur Task Trees (CTT) notation [32,33], and the second one provides a binding of the service methods and parameters to the CTT tasks. Optionally, a description of additional re-

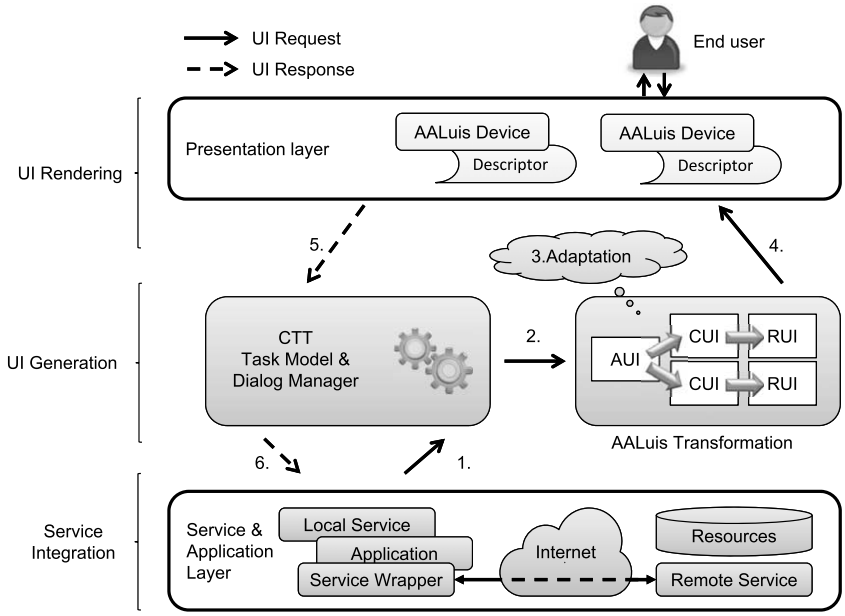

Fig. 3. AALuis user interface generation framework.

sources used for different output modalities can be provided.

The user interface is generated at runtime based on the interaction – specifically, using the task model provided by the CTT. Final results of the automatic transformation process are renderable user interfaces (RUIs) in HTML5, which are dynamically generated for every specific I/O channel. The flexible transformation approach also supports other output formats, such as VoiceXML [30], and thus allows the usage of a wide range of I/O devices. There is a set of default transformation rules for different devices available and additional ones can be provided.

The generation process takes into account not only the task model but also additionally available information. Besides the I/O devices and their capabilities, this information also comprises user and environmental contexts [19]. The task model and the Abstract User Interface (AUI) – which is derived from the task model and which is device and modality independent – are enriched step by step to create a device and modality dependent user interface that is optimized for a specific I/O channel, i.e., the final renderable user interface. The abstract user interface and the intermediate steps of the transformation process (i.e., the concrete user interfaces or CUI), are described in MariaXML [35], a model-based language for interactive applications.

In Fig. 4, one can see an example of a user interface for the television generated by the AALuis system. This user interface is generated solely based on the provided task model in CTT notation and the actual context of use. The presented graphical user interface uses an avatar as a virtual presenter of information. This creates additional value by adding a visual display to the verbal information.

4.2. GPII/URC

In the following, a combination of the Universal Remote Console (URC) [14,43] and the Global Public Inclusive Infrastructure (GPII) [44] technologies, and their main components is introduced.

4.2.1. Universal Control Hub (UCH)

The Universal Control Hub (UCH) architecture is a middleware-based profiling of the URC technology. It allows for any kind of controller to connect to a middleware unit that in turn communicates with one or more applications or services, called *targets* [14]. The UCH implements the URC technology as a gateway between a set of controllers and a set of targets. The UCH follows the CEA-2014 standard [4], offering a list of remote user interfaces from which a connected controller can choose an appropriate one. Figure 5 illustrates the UCH architecture.

When a target (e.g., a television) is discovered by the UCH, it requests and downloads an appropriate target adapter (TA) from the resource server. Then, a user interface socket for that target is instantiated in

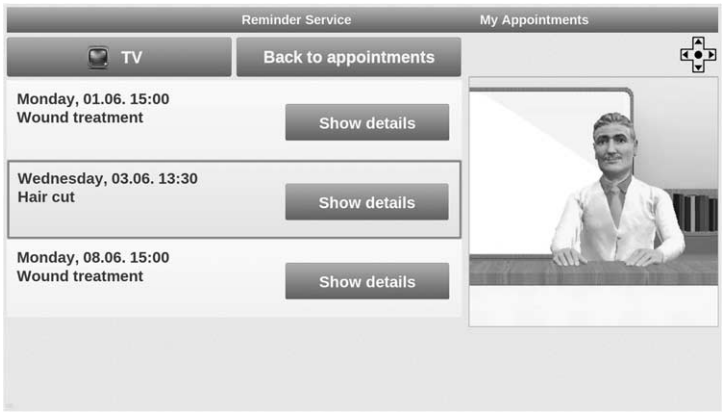

Fig. 4. Example of a user interface generated by the AALuis system for a TV.

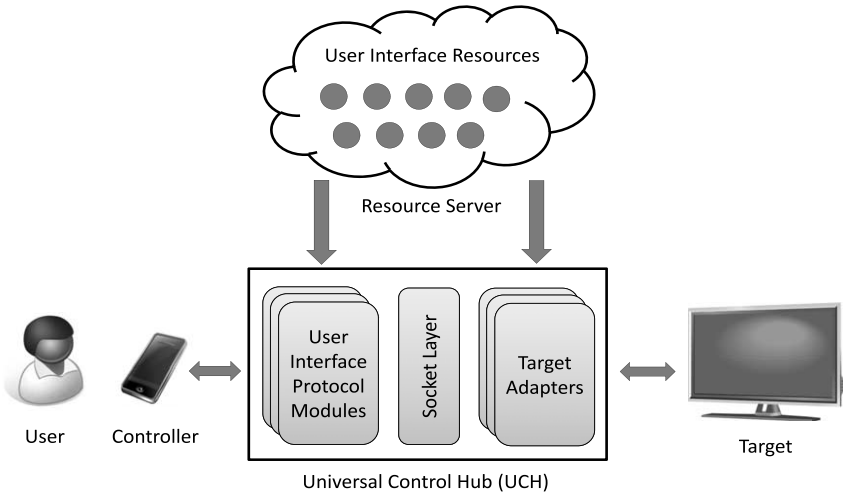

Fig. 5. Overview of the Universal Remote Console (URC) architecture.

the UCH's socket layer. The variables, commands and notifications in the user interface socket represent the state of the connected target. If multiple targets are connected to the UCH, each one has an associated user interface socket as a means to access and modify the target's state.

When the user connects with his/her controller (e.g., a smartphone) to the UCH, the controller receives a list of connected targets and user interfaces that are available for remote control. The UCH may download a number of user interface resources from the resource server in order to build the final user interface. More specifically, only those user interface resources which are suitable for the actual context of use (i.e., the user and his/her controller device) are downloaded and presented. It is up to the User Interface Protocol Module (UIPM) to decide how to interact with the controller (i.e., in which protocol and format), but most commonly HTML over HTTP is used. Multiple UIPMs can be present in the UCH at the same time, offering choices of protocols and formats, and the controller can pick the most suitable one.

Pluggable user interfaces can be written in any programming language and run on any controller platform. Currently, most user interfaces are based on HTML, CSS and JavaScript, communicating with the user interface socket via the URC-HTTP protocol [41], using AJAX and WebSockets as communication mechanisms. Thus, they can simply run in a controller's Web browser, and no prior installation is needed on the controller side.

The UCH components (target adapters, user interface protocol modules) have open APIs and can be downloaded on demand from the resource server at

runtime, thus providing a fully open ecosystem for user interface management.

Currently, the German research project SUCH (Secure UCH) develops a UCH implementation with special security and safety mechanisms built-in, focusing on confidentiality, privacy and integrity of data as well as non-repudiation and access policies based on a role concept. The implementation is guided by the ISO/IEC 15408 Common Criteria [6] methodology for IT security evaluation and aims to be a certified open-source implementation of the UCH at the Evaluation Assurance Level 4 (EAL4).

4.2.2. Resource server

The openURC resource server (see Fig. 5) is a repository for arbitrary user interface resources. It is used as a deployment platform for application developers and user interface designers. The resource server acts as a marketplace for the distribution of target adapters, user interface socket descriptions, user interface protocol modules, and user interface resources that may or may not be specific to a particular context of use. The resources are tagged with key-value pairs for searchability, following the Resource Description Protocol 1.0 standard [27] of the openURC Alliance [26].

Controllers and UCHs can download appropriate components from the resource server upon discovery of specific targets and requests for user interface resources specific to user preferences, and device and environment characteristics. Controllers and UCHs access the resource server through a REST-based protocol [28], which is standardized by the openURC Alliance.

4.2.3. GPII personal preference set

While the URC enables pluggable user interfaces (in any modality) designed for specific user groups in a specific context of use, it does not define a user model (needed for step 4 in Section 3.5), nor does it actively support user-driven customization at runtime (cf. step 6 in Section 3.5). However, the Global Public Inclusive Infrastructure (GPII) defines a so-called *personal preference set*, which encompasses parameters for user interface parameterization and their preferred values for a specific user. This new format of a personal preference set is currently being standardized as a future version of ISO/IEC 24751 [12]. A user may allow his/her personal preference set to be stored in the cloud, thereby making it (or an appropriate subset of it) available to any device with which the user interacts. For example, a public computer in a library could automatically turn on closed captions in videos for users with this preference enabled in their personal preference set. A user may have to set up an initial user preference set with closed captions enabled, or the system may remember that the user customized this setting in a previous interaction (cf. step 6 in Section 3.5).

4.2.4. Integration of URC and GPII technologies

Taken together, the combination of the URC technology and the GPII technology covers all steps of user interface adaptation, as described in Section 3.5. URC with its user interface socket provides the abstract user interface description and the concept of pluggable user interfaces. These user interfaces may be parameterizable to respond to a user's preference set, a device's constraints and/or a situation of use. At runtime, a user's preference set is downloaded from the GPII preference server, or read from a local repository (e.g., a USB stick). In addition, device and environment characteristics are captured by the GPII device and runtime handlers. In essence, URC provides a roughly matching user interface (user interface integration), and GPII allows for further fine-tuning (user interface parameterization) to achieve an optimal adaptation for the user, the device and the environment. This mechanism even allows for fine-tuning during runtime, in order to react to changing conditions of use (e.g., ambient light or noise).

Figure 6 shows a demonstrational online banking application whose Web interface has been manually designed, based on the GPII/URC platform. At the top, the Personal Control panel allows the user to customize presentational and other aspects of the Web interface. In this demo application, video clips in sign language have been prepared by a third party, giving a detailed explanation for each input field of the wire transfer form. If the user sets their preferences to include sign language output, they will see small sign language icons next to the input fields, and pressing them will show an overlay window with a human sign language interpreter or sign language avatar (whatever is preferred and is available).

4.3. universAAL

The universAAL UI framework is a product of a consolidation process including the existing AAL platforms Amigo, GENESYS, MPOWER, OASIS, PERSONA, and SOPRANO [11] in order to converge to a common reference architecture based on a reference model and a set of reference use cases and requirements [40]. As a result of the work done, the uni-

Fig. 6. A demonstrational online banking user interface, based on the GPII/URC platform. Through the Personal Control Panel (shown on the top of the screen) the user can set presentational and other adaptation aspects. Sign language videos are available for the input fields via a special icon. The screenshot shows a video in international sign language (with a human sign language interpreter) for the input field "account number".

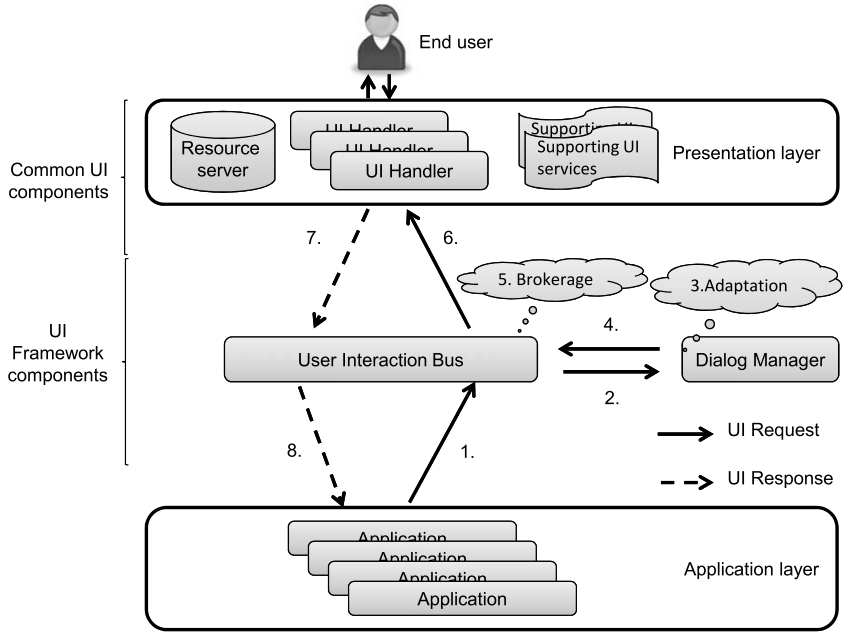

Fig. 7. universAAL User Interaction management framework.

versAAL Framework for User Interaction in Multimedia, Ambient Assisted Living (AAL) Spaces was accepted as Publicly Available Specification by the International Electrotechnical Commission under the reference IEC/PAS 62883 Ed. 1.0.

Figure 7 illustrates components of the universAAL UI framework together with the information flow from the calling application to the end user and back. The UI framework consists of the following parts:

– The User Interaction Bus (UI Bus) includes a basic, ontological model for representing the abstract user interfaces and means for exchanging those messages between UI Handlers and applications. *Forms* – the ontological representation of the typical user interaction components, like textual inputs, multiple selections, buttons, and so on – are created by UI Callers (as part of the applications that are connected to the UI Bus). These are sent to UI Handlers (within UI Requests) to be rendered, filled by user, and sent back to UI Callers to be processed. UI Requests along with the *form* (abstract user interface) carry the information about addressed user, the priority of the

Fig. 8. Example of a generated user interface by the universAAL framework.

dialog, and the language and privacy level of the content. The language information is used by the UI Handlers to retrieve the proper internationalization file from the resource server.

– The UI Handler is the component that presents dialogs (payloads of UI Requests) to the user and packages user input into UI Responses that carry the input to the appropriate application. UI Handlers ensure similar or same look and feel across various applications and devices, which helps the user to quickly become accustomed to the system. Different UI Handlers have different capabilities in terms of supported modalities, devices, appropriateness for access impairments, modality-specific customizations, context-awareness capabilities, etc.
– The Dialog Manager manages system-wide dialog priority and adaptations. It adds adaptation parameters (e.g., user preferences) to the UI Requests coming from the application and returns it to the UI Bus. There, the UI Bus Strategy decides which UI Handler is most appropriate for the interaction with the user in this specific moment (considering current context and information, such as user location).
– The universAAL resource server stores and delivers resources, such as images, videos, etc., which cannot be represented ontologically, to the UI Handlers [39].

In Fig. 8, one can see an example of a user interface generated by the universAAL framework. The main screen of the universAAL system for the assisted person rendered by the UI Handler GUI Swing is shown. It displays an example setup where Medication Manager, Agenda, Long Term Behaviour Analyzer (LTBA) and Food and Shopping services are installed together with system-provided services for filtering installed services (Search) and changing current setup of user interaction preferences (UI Preferences Editor).

4.4. Summary

Figure 9 gives a summary of the three systems for user interface adaptation in terms of the user interface levels of the Cameleon Reference Framework (CRF) [3]. The Cameleon Reference Framework structures the user interface into four levels of abstraction, from task specification to the running interface. The level of tasks and concepts corresponds to the Computation Independent Model (CIM) in Model Driven Engineering; the Abstract User Interface (AUI) corresponds to the Platform Independent Model (PIM); the Concrete User Interface (CUI) corresponds to the Platform Specific Model (PSM); and the Final User Interface (FUI) corresponds to the program code [37]. Figure 9 presents the equivalents to these levels for each of the three systems. GPII/URC and universAAL have equivalents for two and three of the four abstraction levels respectively, and AALuis has all four.

5. Comparison of the three systems

In the following, the three systems are described in detail based on ten defined comparison criteria. The criteria have been developed by the authors based on the generic framework presented in Section 3.4. The motivation behind all ten criteria is to cover all steps for user interface adaptation and to allow for a qualitative comparison of the three systems. Each crite-

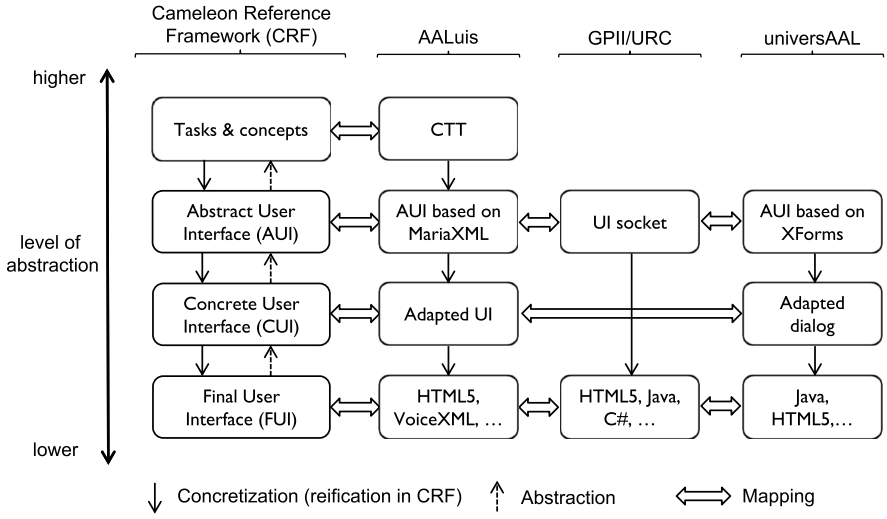

Fig. 9. Relation of the three systems with respect to the Cameleon Reference Framework (CRF).

rion is briefly presented with respect to the framework and literature. The first eight are directly related to the generic framework (see Section 3.4) and the last two indirectly. All of them are of general importance for adaptive systems. After the short description of each criterion, the systems are described based on some guiding questions. The main attention has been on overlapping and complementary aspects to highlight the possibility to create added value by combining these different systems or by enriching one of the systems by selected features.

5.1. *Form of abstract user interaction description (C01)*

A separation of the presentation (i.e., user interface) from the application and its functionality is helpful to ensure that service developers can focus on the application development (cf. step 1 in Section 3.5) and do not need to be concerned with the user interface or any special representation required by the target group [19] (cf. steps 2 & 3). As described in the approach of ubiquitous computing [49], it allows the input to be detached from a particular device and to easily exchange I/O devices for interacting with the requested application [17].

> *In which form are the abstract user interaction descriptions defined and how is the separation realized? Are there any tools to support the developer?*

In **AALuis**, the detachment of the functionality of the service from the automatically generated user interface is realized by the task model and a simple binding file. The Concur Task Trees (CTT) notation is used to model the structure of the interaction in a well-defined way and allows an unambiguous interpretation of the interaction flow. The CTT is a hierarchical logical decomposition of the tasks represented in a tree-like structure with an identification of the temporal relationships among tasks at the same level [32]. Developers can use the Concur Task Trees Environment (CTTE), which is a graphical tool for the creation and analysis of task models [22]. The binding file, in XML format, acts as connector between the CTT tasks and the corresponding service functions to be called. It models a many-to-one relationship between them. Furthermore, the task parameters for each function call are defined within the binding file [20]. The abstract user interface in MariaXML notation is generated from a set of currently active tasks of the CTT, the so-called Presentation Task Set.

The **Universal Remote Console (URC)** framework facilitates pluggable user interfaces based on an abstract user interface, called a (user interface) socket. The user interface socket description is a contract between the application developer and the user interface designer. A socket represents the interaction elements of a target in an abstract (modality-independent) manner, by means of variables, commands and notifications, as follows:

- A variable denotes a data item that is persistent in the target, but usually modifiable by the user (e.g., the power status or the volume level on a TV).

– A command represents a trigger for the user to activate a special function on the target (e.g., the start sleep function on a TV).
– A notification indicates a message event from the target to the user (e.g., a could not connect message on a TV).

An open-source socket builder tool is available for application developers to conveniently create and edit a socket description, along with other URC components and resources (e.g., labels).

The **universAAL** UI Framework enables independence between the application and the presentation layer. Application developers need only to consider what information they want to present to and receive from the user. The *how* aspect is handled within the UI Framework, meaning that information received from the part of the application connected to the UI Bus (UI Caller) is just sent to the UI Bus. There, the most suitable interaction modality for the addressed user is selected, based on the user's capabilities and user interaction preferences. The components dealing with how to present information to the user and how to obtain user inputs are called UI Handlers [10].

Dialog descriptions in universAAL enable the composition of abstract representations of the user interfaces. They are based on XForms [2], which is an XML format and W3C specification of a data processing model for XML data and user interface(s), such as web forms [46].

5.2. Support for user interface design (C02)

The user interface can be manually designed or automatically generated based on an abstract description of the functionality (cf. step 2 in Section 3.5). Either way, these approaches yield a default user interface which can be supplemented by third parties, and further adapted at runtime. There is a need to support the designer or the system to generate the default user interface, and this support can take various forms.

How is the design of the default user interface supported?

In **AALuis**, the default user interface is automatically generated based on the abstract description of the service and a default set of transformation rules. The rules are available for different I/O device classes, such as, tablets, smartphones, TVs. The user interface designer can additionally create new transformation rules for additional device classes and modalities. For example, the transformation from the CUI to a final user interface represented by VoiceXML can be defined in an additional transformation rule and, thus, be used as an alternative for the user interface representation.

In the **URC** framework, final user interfaces can be manually designed or automatically generated based on the user interface socket description. At runtime, all user interfaces, including those manually designed and automatically generated, are available to be used for whatever context of use is present. If delicate design is an important feature of a product, user interface designers and human factors experts can design user interfaces in any common language and for any common platform (e.g., HTML5, Java, C#, iOS, Android). The user interface code just needs to embed small code fragments for communication with the socket (e.g., via the URC-HTTP protocol, see Section 4.2). For the design of Web applications in HTML5, an open-source JavaScript library is available. The user interface designers can use whatever design tool they are familiar with, as long as it allows them to add custom script code.

If manually designed user interfaces are too expensive, or if no appropriate user interface is available for a specific application and context of use, it is also possible to generate one based on the description of a service's socket (in this case, steps 2 & 3 are done by some automatic system at development time or runtime). GenURC [52], part of the openURC resource server implementation, is such a system, transforming a user interface socket and its labels into a Web browser-based user interface. As soon as the socket description, a grouping structure and pertaining labels are uploaded for an application, GenURC generates a user interface in HTML5 that is especially tailored for mobile devices by following the one window drill-down pattern [42]. The result is then made available to clients, along with all other (manually designed) user interfaces for that specific target.

In **universAAL**, final user interfaces are automatically generated by the selected UI Handler upon receiving an abstract representation of the user interface. From an application developer's point of view there are four options when designing a user interface, each differing by the level of universAAL UI Framework usage. The easiest option in terms of developer effort is to simply select one of the universAAL UI Handlers and determine the desired look and feel package. The second option is to do the same, but to use additional recommendations (recommendation ontology) to give the UI Handler instructions how to render the FUI. The third option is to design another alternative look

and feel package for an existing UI Handler or even a complete UI Handler from scratch. The last option is to avoid the universAAL UI Framework and to design a final user interface specifically for one application. When using the universAAL UI Framework, resources such as images, video and other files that cannot be represented with the universAAL data representation model are only referenced in AUIs and loaded via an appropriate UI Handler using its specific language (e.g., Java Swing, HTML5, Android, etc.).

5.3. Third party contributions (C03)

The provision of additional resources and alternative user interface resources for use in the adaptation process is mentioned in step 3 in Section 3.5. These resources can either be provided locally or globally for the system deployment by the developer him/herself or by third parties, which can be external companies or other user interface designers or experts.

> *Is it possible for third parties to contribute to the user interface adaptation process and if so, how and what tools are available?*

There are several possibilities for contributions by third parties in **AALuis** and the system is designed in a modular and flexible way to support these contributions. These range from new transformations, to the integration of new AALuis-enabled devices and the description of their capabilities, to additional resources as supplemental content for the user interfaces – e.g., pictures, logos or videos. New transformation rules aim at the provision of additional I/O modalities. An AALuis-enabled device, including the description of its capabilities, refers to any device that runs a small and simple application handling the automatic connection to the AALuis system either via UPnP or socket-based communication and provides basic information regarding its capabilities. Supplemental resources, e.g., sign language videos, can be included to allow an enrichment of the user interfaces. The AALuis system is prepared for use with a resource repository but currently has none. Instead, additional resources are provided via a URI.

In the **URC** framework, the developer of an application is responsible for creating a single user interface socket which handles the communication between the service backend and any of its user interfaces. Once a socket has been created and made publicly known by its developer, anybody can build supplemental user interfaces or alternative parts of user interfaces. If a user has access rights to the socket, it allows for multiple (supplemental) user interfaces that can plug into the socket at the same time or at different times. Thus, internal and external user interface designers can inspect the service's socket description and create pluggable user interfaces that map to the socket, accommodating the needs of a special user group, a special controller device and/or a special context of use in general. Such pluggable user interfaces can be developed and contributed by third parties (e.g., external Human-Computer-Interaction (HCI) experts, user groups, and users themselves), and deployed to the openURC resource server for a selected audience to be used. The resource server creates a market for user interfaces that is separate from the market of applications.

In **universAAL**, the existing infrastructure is modular and allows addition of new (pluggable) UI Handlers and/or look and feel packages. Coexistence of alternative UI Handlers at the same time during the system usage is also possible and, since each UI Handler has its own profile in terms of appropriateness for some specific parameter (e.g., user impairment, device support, context awareness, etc.), this can add additional benefits in different contexts of use. Additional resources are uniquely described via global URIs and loaded at runtime by the UI Handlers. Pluggable UI Handlers developed by third-party HCI experts and designers can also be offered to the end users the same way as universAAL services via the uStore (similar to Apple's AppStore). Some examples and guidelines are available for UI Handler developers.

5.4. Context of use influencing the adaptation (C04)

Figure 2 and Section 3.2 describe the context of use, covering information about the user, the runtime platform and the environment, and its importance as a source of information for any adaptation process of the user interface (cf. step 4 in Section 3.5). The context information can be modelled and implemented in various ways.

> *Which context information is modelled and how is it implemented? What aspects of the context of use are taken into account in the adaptation process of the user interface?*

AALuis uses various information sources in the transformation process. They are represented in different context models and cover different aspects relevant to finding the best match of user interface representation. The information ranges from data about the in-

dividual's current environmental conditions (e.g., surrounding light or noise), to the capabilities of registered I/O devices (e.g., screen size or supported I/O modalities) to user capabilities and preferences (e.g., hearing capability, preferred modality and I/O device) [16]. This threefold information is used for the automatic selection of the best-suited I/O device, modalities and presentation. The environmental and device model are currently implemented as key-value pairs, but are open for extension to upcoming standardized models. The user model is so far based on MyUI [36], but an extension to GPII preferences is under consideration.

In the **URC** framework, the UCH architecture defines an extensible context model for the user and platform. Commonly used terms for the description of user preferences and platform and environmental characteristics are being developed by GPII in the form of key-value pair, with URIs as keys and a clear definition of their value spaces. These terms are specified in a repository-based approach so that they can be quickly changed to reflect technological advancements. Currently, common preference terms cover a wide variety of user preferences, and can be extended if needed. In the future, these terms are planned to be adopted by the next version of ISO/IEC 24751 [13].

The **universAAL** UI Framework uses different ontology-based user, system and environmental models and a very modular design approach in the adaptation process. During this process, user interface preferences and other user data (such as impairments, location, language, etc.) are used together with descriptions of interaction devices and other UI Handler-specific descriptions to determine the best possible UI Handler for the given situation, and also to enable a "follow me" scenario for the end user where dialogs coming from the applications "follow" the user as he/she is changing the location. The Dialog Manager, as a component specifically designed to adapt new dialogs coming from the applications, is responsible for providing explicit and implicit update mechanisms for the user interface preferences subprofile that can not only be updated but also extended at runtime. The adaptation mechanisms use a weighted scale of the adaptation parameters to ensure the best possible adaptation.

5.5. Maintenance of user model (C05)

The user model is an important source for any adaptation of the user interface (cf. step 4 in Section 3.5). Thus, the question arises whether the user model can be changed manually by means of tools or whether changes are done automatically as a result of common user interactions.

Can the end user look into and change his/her own user model and if so, how?

AALuis provides predefined profiles to be chosen by the individual. While this has the advantage that predefined profiles can be initialized easily, it also has one big disadvantage: there is always the question of how to choose the most suitable and appropriate settings. The chosen approach is to link the profiles to the CURE-Elderly-Personas, which are fictitious persons synthetically generated from average traits mixed across countries [50,51]. The user can browse through the personas and choose the best fitting. Thus, the underlying user settings are selected to be used in the user context model. The user can also customize these settings to his/her needs and these modified settings can be provided via a profile repository for future use. A support tool for the selection and customization of user profiles is under development and will be provided in the near future. Currently, no automatic update of the user model via a feedback loop is realized.

GPII is currently working on a couple of tools that will allow the user to create and maintain their preference set (i.e., user profile). The Preference Management Tool (PMT) is a Web browser application that supports a user in setting up a personal preference set and populating it with initial values that reflect their user interface preferences (e.g., font size, contrast settings, keyboard settings, etc.). The Personal Control Panel (PCP) provides a central point of preference control for the user at runtime. The PCP is always available, together with any application that the user is currently using. Therefore, the user can customize the application's user interface through the PCP, and changes will immediately take effect on the application. Thus, there is just one single preference settings dialog for all applications, so that the user will quickly get familiar with it. Changes on the PCP will automatically be recorded in the user's preference set, which is either stored locally or on the preference server. Hence, the system can reactivate them when the appropriate context reoccurs.

All user profile related data in **universAAL** is stored on an always up-to-date central profiling server that exposes this information for services that may need it. The user profile ontology is extended by the user preferences subprofile related to user interaction and inspired by the standards ETSI ES 202 746 [9] and

ISO/IEC 24751-2 [13]. The Dialog Manager provides means ("screen" in terms of GUI) for an explicit change of user interface related preferences by the end user, as well as a pluggable architecture for adaptors that can automatically influence adaptation parameters. During its first use, the system recognizes the type of user based on a user role (connected to the user profile) and initializes the user interface preferences based on stereotype data associated with the user role (e.g., assisted person, caregiver).

5.6. User interface integration and/or user interface parameterization (C06)

As stated in steps 5 & 6 in Section 3.5, there is a difference between user interface integration and user interface parameterization. In other words, the full user interface or just parts of it are exchanged or the user interface is fine-tuned by setting predefined parameters.

What method(s) is/are used for the system-driven accommodation and the user-driven customization? Just integration, just parameterization or a combination of both?

AALuis combines both approaches. On the one hand, different widgets are integrated into the user interface based on the automatic device and modality selection. An example of such widgets are the on-the-fly generated avatars. These are used as an additional I/O modality and virtual presenter for arbitrary messages. These messages can, for example, provide additional information explaining the currently displayed user interface. On the other hand, user interface parameterization is utilized for tweaking of the user interface with respect to contrast, font size, etc. related to the presentation and inputs event layer.

The **URC** framework supports user interface integration through its concept of supplemental resources and the openURC resource server as a marketplace for user interfaces and user interface components. GPII supports user interface parameterization, with the personal preference set acting as a central reference point for setting the user interface parameters at runtime. Taken together, the two approaches result in a system that supports both user interface integration and user interface parameterization.

The **universAAL** UI Framework combines both approaches as well. User interface integration is supported by alternative UI Handlers. Parameterization is achieved via user-specific preferences and impairments and user interface-related recommendations coming from the applications which send outputs to the user. Both users' preferences and application designers' recommendations influence the final generated user interface. Appropriateness for certain impairments, as part of the UI Handlers' description, influence the process of selecting the most appropriate UI Handler for a given user/situation.

5.7. Support for adaptability and/or adaptivity (C07)

As described in Section 3.3, user interfaces may be adaptable, i.e., the user can customize the interface (cf. step 6 in Section 3.5), and/or adaptive, i.e., the system adapts the interface on its own or suggests adaptations (cf. step 5 in Section 3.5). As stated, hybrid solutions are possible as well. If adaptivity is supported, the matchmaking between the current context of use and the "best" suited adaptation is a delicate and important task for the acceptance of the user interface. There are various matching approaches, including rule-based selections or statistical algorithms.

Are adaptability and/or adaptivity supported and if yes, how is it realized? If adaptivity is supported, which strategies for matchmaking are deployed?

With respect to adaptability and adaptivity, **AALuis** again uses a hybrid approach with a broad range of features. The user can customize his/her user preferences at runtime, and the updated user model immediately influences the transformation process. This refers to the adaptability of the system. Adaptivity, on the other hand, is reflected in the automatic device and modality selection strategy and the user interface parameterization based on the user model. Currently, a rule-based selection process is implemented, but different strategies (e.g., based on Bayesian networks) are under development. These selection strategies take into account the environmental model, the device capabilities and their availability and the user preferences.

In **GPII**, a user can change his/her preferences at any time via the Personal Control Panel (PCP, see Section 5.5), i.e., the system is adaptable. The changes will immediately take effect, and the system will remember the new settings and the context under which the settings were changed by the user. In addition, the system can learn from the user's preferred settings and their specific contexts, and can propose these settings to the user in similar contexts (i.e., the system is also adaptive). The GPII/URC system is also adaptive with regard to the automatic selection of a (supplemental) user interface that fits the platform characteristics. In

GPII, rule-based and statistical matchmakers are currently being developed. The rule-based matchmaker applies rules created by experts to propose new settings, and can even suggest new assistive technologies that the user is not yet familiar with. The statistical matchmaker looks at all user profiles across all users (data mining over anonymized data), thus filling gaps and proposing settings for unknown contexts in a user's preference set. For example, for a user who has configured his/her Windows computer at home and who is now initializing his/her new iPhone device, the statistical matchmaker could activate the iPhone settings of another person who has used the same or similar windows settings.

In **universAAL**, when a meaningful match between the most current user context, preference data and different managers responsible for covering different I/O channels (UI Handlers) is desired, it is necessary to have good profiles describing the capabilities of those software components responsible for user interaction. This information, including appropriateness for access impairments, modality-specific customizations and context-awareness capabilities, is initialized at the moment of component registration, but is also refreshed if something in the setup changes. In addition, the UI Framework is immediately updated if an adaptation parameter changes (e.g., if the user location changes, then the UI Framework sends the dialog to another UI Handler which is closer to the user).

5.8. *User interface aspects affected by the adaptation (C08)*

In Fig. 1, various aspects and levels of adaptation are mentioned and clustered into a 3-layer model approximating the depth of their leverage (cf. step 5 and 6 in Section 3.5). The resulting adaptations can be shallow modifications impacting the presentation & input events layer, medium-deep modifications impacting the structure & grammar layer, or deep modifications impacting the content & semantic layer of the user interface model.

Which aspects and user interface layers are affected by the adaptations?

The adaptations in **AALuis** affect all three user interface layers described in Fig. 1. User interface parameterization handles aspects related to the presentation and input events layer, such as text size, line spacing, contrast setting, color setting, button size & distance, and so forth. On the structure and grammar layer, AALuis provides adaptations regarding input and output modalities, grouping structures and the provision of various widget sets. Adaptations with respect to the content and semantics layer are all supported by the data handling and cover audio descriptions via the avatar, captions, et cetera.

Regarding **GPII/URC**, most presentation aspects are covered by the GPII Personal Control Panel (PCP). Deeper aspects of adaptation can be dealt with by providing supplemental user interfaces or user interface components. In particular, the URC framework allows for the definition of a user interface structure via grouping sheets, which can be used to generate a specific navigation structure at runtime. Also, captions, audio descriptions and natural language-specific resources can be downloaded from the openURC resource server at runtime.

In **universAAL**, support for adaptability and adaptivity affects the top two layers described in Figure 1. Before runtime, end users or deployers decide on the most appropriate UI Handlers with corresponding look and feel packages. UI Handlers realizing the pluggability aspect and providing for diversity in terms of different presentation, structure and content delivery are also capable of fine-grained adaptations during runtime. Additionally, since the universAAL UI Framework is designed for distributed environment adaptation aspects, it may utilize the position of a user to realize the so-called "follow me" scenario with content delivery as the user moves around. Means of such delivery depend on which UI Handlers are installed as well as on the privacy level of the content to be delivered, appropriate modalities, user preferences and some other user specific needs and wishes.

5.9. *Support for multimodal user interaction (C09)*

It is generally assumed that multimodal user interaction increases the flexibility, accessibility and reliability of user interfaces and is preferred over unimodal interfaces [8]. It includes input fusion and output fission, which refers to modality fusion when capturing user input from different input channels to enhance accuracy, and modality fission when using different output channels for presenting output to human users [47,48]. It is thus directly related to multimodality regarding input and output channels.

In which way is multimodal user interaction supported and at what stage are input fusion and output fission realized?

AALuis allows the usage of multimodal user interfaces. The I/O modalities can be combined in a flexible way depending on the context models used. The required data are either provided dynamically by the service as additional resources (e.g., pictures, logos or videos) or automatically generated (e.g., text-to-speech and avatars). The data are stored and represented in an internally defined format, which allows the transport of different data types. In the transformation process, the chosen modality set is incorporated into the transformation context variable and these are rendered in the final user interface [20]. An example of multimodality is the simultaneous output of text, speech and an on-the-fly generated avatar. A next, planned step towards natural user interaction is to extend the set of transformation rules by a VoiceXML transformation. Furthermore, an extension towards a possible ambient output modality, such as changing ambient light, is under investigation.

In **GPII/URC**, a user interface may employ any modality for input and output, in any combination. However, specific interaction mechanisms are out of the scope of the system, and hence input fusion and output fission are not directly supported. User interface designers and system developers are free to use whatever code, library or framework they want to use to combine input and output events in a manner appropriate to their application.

The **universAAL** UI Framework has a strong support for multimodality. When the framework accepts the UI Request coming from applications, it adds additional context and user related adaptation parameters before deciding on the output modality, time of delivery and output location. The connection of the UI Framework to the Context Bus (for context information about the user environment) and the Service Bus (for sharing functionalities between different parts of different applications and higher level reasoners), allows to use information about the user's surrounding in further dialog processing.

5.10. Support of standards (C10)

Standards play an important role in ensuring exchangeability of user interfaces and compliance with the changes in needs of older adults. They offer, for example, the possibility to prepare user interface generation to run not only on currently available devices, but also on an upcoming generation of devices.

> *Are there any pre-existing standards applied and are there any standards influenced by or based on the system?*

AALuis supports and implements a wide range of standards, such as UPnP, XML, HTML5, CSS, JS, XSLT, WSDL, OSGi. The Concur Task Tree (CTT) notation is used for the task model, as is the model-based language MariaXML for the intermediate stages (AUI and CUI) of the user interface in the transformation process. Both are W3C working drafts submitted to the W3C Model-Based UI Working Group in 2012 [45]. The insights and experiences from applying these working drafts in AALuis can be a good basis for contributions to the W3C working group. The standardized usage of the CTT notation and the integration of services as separate OSGi bundles or loosely coupled web services, allow connection with any service platform and, thus, usage of the system by any service provider.

The **URC** framework is specified as the international standard ISO/IEC 24752 [14] in multiple parts. The openURC Alliance is a consortium for the development and standardization of the URC technology, and has published a series of Technical Reports as standardized implementation guidelines for the URC ecosystem [25]. More Technical Reports are currently under development. URC and UCH build upon a broad set of commonly established standards, including XML, UPnP Remote User Interface, CEA-2014 and Bonjour. The preference set of GPII will be specified in a future version of ISO/IEC 24751, using a repository-based approach to allow for flexible changes in the set of common preference terms, as is needed when new technologies emerge.

The **universAAL** Framework for User Interaction in Multimedia, Ambient Assisted Living (AAL) Spaces is specified as a Publicly Available Specification by the International Electrotechnical Commission (IEC) under the reference IEC/PAS 62883 Ed. 1.0., released at the end of 2013. The XForms specification is used for the description of the abstract user interfaces. Data representation is based on RDF and OWL specifications which are used for describing all universAAL ontologies, including the user interface preferences ontology inspired by the ETSI ES 202 746 standard [9] and ISO/IEC 24751-2 [13].

6. Discussion

In the following, the results of the above comparison of the three systems are discussed. Rather than highlighting strengths and weaknesses of each system, the aim is to identify use cases best suited for each of the three systems. The section elaborates on similarities and differences with a focus on open issues. Furthermore, possible paths towards consolidation are outlined, with the purpose of increasing harmonization between the systems.

6.1. Similarities and differences

The analysis of the three systems with respect to the comparison criteria reveals many differences and even similarities, although the systems have been developed independently. Table 1 summarizes the comparison of the systems with respect to the ten criteria. It provides a systematic overview of the different approaches followed in the three systems.

All three systems are designed in an open way to support third party contributions (C03). The provision of additional resources is possible in all systems, but realized in different ways, i.e., via an URI (AALuis) vs. the openURC resource server (GPII/URC) vs. the universAAL resource server (universAAL). It is one of the main goals of all systems and a possible aspect for harmonization by creating a single point for resource provision (see Section 6.3.3). A further aspect of third party contributions is the provision of supplemental user interfaces. It is realized by the provision of additional transformations and descriptions of AALuis enabled devices (AALuis) vs. supplemental pluggable user interfaces (GPII/URC) vs. alternative UI Handlers (universAAL).

The systems use similar contexts of use in the adaptation process (C04), but their representations of the context models are different, i.e., key-value pairs based on MyUI (AALuis) vs. the GPII preference set (GPII/URC) vs. standards for the user model (universAAL). The context of use is directly related to the support for adaptability and adaptivity (C07). All three systems under comparison provide this support, whereas the tools are diverse. As a future goal, these tools should be harmonized and possibly merged, to achieve a cross-platform compatibility of contexts of use (see Section 6.3.2).

All systems support user interface integration as well as parameterization (C06) and affect all three layers of user interface aspects (C08). Again, the realization differs between the three systems. For details see Table 1 and Sections 5.6 and 5.8.

The usage of standards is of utmost importance for all three systems (C10), whereas the form differs slightly, i.e., using standards and working drafts (AALuis) vs. already standardized and contributing (GPII/URC) vs. using standards and being available as a specification (universAAL). Supporting standardization activities jointly would create additional possibilities. First feasible joint standardization activities are sketched in Section 6.3.2.

There are differences in the form of the abstract description of the user interaction and interface (C01), i.e., CTT notation for modelling the interaction and MariaXML for the abstract user interface representation (AALuis) vs. user interface sockets constituting a contract between the service back-end and a user interface (GPII/URC) vs. dialog descriptions using XForms (universAAL).

Another distinguishing feature concerns support for the user interface design (C02), i.e., automatic generation based on existing but extensible transformation rules (AALuis) vs. at design time manually crafted,[2] pluggable user interfaces (GPII/URC) vs. automatic generation based on existing and extendable UI Handlers (universAAL).

Especially the differences in criteria C01 and C02 allow for a differentiation between the best-suited application areas and constraints of the three systems (see Section 6.2), but also provide a potential for harmonization so that they may mutually benefit from each system's advantages (see Section 6.3.1).

The maintenance of the user model (C05) is handled differently in all three systems by means of different tools, i.e., customizable persona-based profiles (AALuis) vs. the web-based Preference Management Tool and Personal Control Panel (GPII/URC) vs. initialization of customizable user preferences based on the user role (universAAL). This is again closely related to possible standardization activities and especially to create a common set of maintenance tools taking into account different strengths and avoiding disadvantages (see Section 6.3.2).

Finally, the systems can be distinguished by their support for multimodality (C09), i.e., whether it is supported in the transformation process (AALuis) vs. possible but specific interaction mechanisms are out of

[2] It is also possible to automatically generate user interfaces in GPII/URC, but design by human experts is the most common way.

Table 1
Summary of the comparison of the selected systems

	AALuis	GPII/URC	universAAL
C01: Form of abstract user interaction description	Task model of application in CTT notation; binding file for the connection between tasks and the service calls; tool for the CTT creation available; abstract user interface represented in MariaXML	User interface socket description as "contract" between application developer and user interface designer(s); open-source socket editor available	Abstract user interfaces based on the XForms W3C specification
C02: Support for user interface design	User interface automatically generated; default transformation rules for certain I/O device classes	User interface designers can use any design tool for any user interface platform; user interfaces may also be generated automatically	Final user interfaces are automatically generated; selected UI Handler contains device specific transformations
C03: Third party contributions	Possible third party contributions are new transformations, new AALuis enabled devices and the description of their capabilities, and additional resources as additional content	Any third party can create a supplemental/alternative user interface and user interface resources based on the application's user interface socket; the openURC resource server is the central marketplace for all user interfaces	Any third party can create a supplemental/alternative UI Handler
C04: Context of use influencing the adaptation	User, environmental and device model	Common terms on user preferences, device and environment characteristics, specified by GPII in a repository-based approach	Different user, system and environmental models based on ontologies
C05: Maintenance of user model	Predefined but adaptable user profiles linked to the CURE-Elderly-Personas	Preference Management Tool (PMT) for setting up a personal preference set; Personal Control Panel (PCP) for changing user interface aspects at runtime	User interface preferences based on stereotype data associated with the user role; always up-to-date user profile stored in profiling server
C06: User interface integration and/or user interface parameterization	User interface integration: different widgets are integrated into the user interface based on the automatic device and modality selection; User interface parameterization: tweaking of the user interface with respect to contrast, font size, etc.	User interface integration: pluggable user interfaces and openURC resource server; User interface parameterization: GPII personal preference set	User interface integration: pluggable UI Handlers providing different interfaces and resource server for providing specific resources; User interface parameterization: user interface preferences in addition to recommendations on user interface rendering
C07: Support for adaptability and/or adaptivity	Adaptability: user can customize his/her user preferences at runtime; Adaptivity: currently, a rule-based selection process is implemented, but different strategies (e.g., based on Bayesian networks) are under development	Adaptability: GPII Preference Management Tool (PMT) and GPII Personal Control Panel (PCP); Adaptivity: GPII matchmaker (learns from the user's configuration)	Adaptability: decision of UI Handler and specific setup; Adaptivity: set of adapters in Dialog Manger are responsible for intelligent management of specific adaptation parameter
C08: User interface aspects affected by the adaptation	Presentation layer: by user interface parameterization; Structure layer: adaptations regarding input and output modalities, grouping structures and the provision of various widget sets; Content layer: adaptations are supported by the data handling	Presentation layer: GPII mechanisms by user interface parameterization; Structure and content layers: URC mechanisms by user interface integration	UI Handlers handle fine-grained adaptation in the presentation, the structure and the content layer.
C09: Support for multimodal user interaction	Defined data format which allows to transport different data types in the transformation process	Out of scope for URC/GPII; multimodality can be added by any code, library or framework	Automatic selection of different UI Handlers managing different I/O channels
C10: Support of standards	UPnP, XML, MariaXML, HTML5, CSS, JS, CTT, XSL, WSDL, OSGi; Can contribute to the W3C working group on MBUI	ISO/IEC 24752; openURC Alliance maintaining Technical Reports for the URC ecosystem; XML, UPnP Remote User Interface, CEA-2014, Bonjour, ISO/IEC 24751	Data model: RDF, OWL, XForms; UI Preferences: ETSI ES 202 746, ISO/IEC 24751-2

scope (GPII/URC) vs. strong support (universAAL). Regarding multimodality, it can be clearly seen that the systems have a different focus – especially for GPII/URC for which it is out of scope. Details can be found in Table 1 and Section 5.9.

6.2. *Best suited application areas and system constraints*

Based on the similarities and differences, one can distinguish between application areas best suited for the different systems while highlighting constraints.

The abstract description of the user interaction or user interfaces by task models and XForms (AALuis and universAAL), and the automatic generation thereupon, allow for the provision of a consistent look and feel for any application provided by the system. This is especially helpful for older adults to give them the same feeling and interaction options with the possibility to adapt to their preferences. A problem of automatic generation is that it needs a trade-off between flexibility to cover as many use cases as possible, and usability and accessibility. In contrast, manually designed user interfaces (GPII/URC) facilitate tailoring the user interfaces for special needs, services and special user groups. This allows flexibility in the design of user interfaces, especially with focus on usability. Thus, one should balance pros and cons if the focus is on the same look and feel for different applications but with the slight problem of reducing usability and accessibility (universAAL and AALuis), or if the focus is on tailored user interfaces for each application and target group, but with the drawback of additional development efforts due to the large number of user interfaces to be developed (GPII/URC).

Another difference is the fact that GPII/URC is mainly designed to control appliances and devices as targets while AALuis and universAAL mainly focus on digital services. On the one side, the user interface sockets (GPII/URC) represent the functionality of the targets. On the other side, the interaction with the digital service is modelled by the CTT (AALuis), or via the UI Bus which includes a basic, ontological model for representing the abstract user interfaces and a means for exchanging messages between UI Handlers and services (universAAL).

A major difference between AALuis and universAAL, as examples of automatic user interface generation, and GPII/URC is the fact that for AALuis and universAAL, the user interface is generic for the application and specific to the I/O controller. This means that, when a new service is added to the system, it can be accessed without requiring new code for user interfaces. Furthermore, it allows for a simple modality and device change at runtime. In contrast, for GPII/URC the user interface is specific for the application *and* the I/O controller. This means that, upon extending a system by new targets, new user interfaces must be made available to control the new targets. Note that there are also means within GPII/URC for generating user interfaces automatically, leading to generic user interfaces for the application, but this is not the main purpose of the system. Additional I/O controllers can be added in AALuis and universAAL by providing new transformation rules and UI Handlers, respectively. In GPII/URC, these are added by the provision of additional pluggable interfaces for each application.

Another distinction between the three systems is the additional effort needed to use it, and where in the lifecycle this effort occurs. This is directly related to the first two steps of the generic interface adaptation framework. In AALuis, the overall efforts for using various user interfaces on different I/O controllers are shifted toward the service developer, as the creation of a task model for each application. Thus, a service developer must learn the CTT notation once before getting started. Further transformation rules can be developed by user interface designers and developers if needed, but are not necessary for initial usage of the system, since existing transformation rules can be reused. For GPII/URC, the efforts are more equally distributed between a service (target) developer who defines the socket description and develops the target application, and one or more user interface experts who design interfaces for each target and controller platform. In universAAL, the efforts are focused on the abstract description of the user interaction via the UI Bus by the service developer and the usage of the existing UI Handlers, which are realized in Java and can be extended. The initial effort is mainly related to getting to know the universAAL platform and the underlying concept of ontologies and XForms. In summary, the efforts are clearly distributed differently.

There are differences with regard to how the systems bind to their back-ends, i.e., the applications providing the functionality. AALuis supports the integration of local services as separate OSGi bundles, or remote services as loosely coupled Web services. For the latter option, there is no explicit connector development needed. Thus, there is no service development in the same system necessary, which allows service developers to use existing platforms (so they do not need

to migrate to a different development environment). In GPII/URC, a service developer needs to implement a target adapter on the UCH platform (currently in Java) to provide the binding. In universAAL, the application's connection to the universAAL system is realized by using the service bus via an OSGi bundle. Thus, the application developer needs to integrate his/her developments into the universAAL platform. Summarized, one can say that in AALuis there is no need for additional binding, i.e., developers do not need to develop anything within the AALuis system when connecting a Web service application, rather only to provide the task model. In contrast, in GPII/URC and universAAL, programming is needed to integrate applications into the systems.

Another difference in the systems is the division of roles in the three systems. In GPII/URC, there is a clear separation between the development and design efforts. This implies that user interface designers can provide user interfaces for the system. AALuis and universAAL do not follow a strict separation of roles. To provide new user interfaces for additional I/O devices, new transformation rules and UI Handlers, respectively, have to be implemented.

6.3. Work towards harmonization

Harmonization of the three systems can help them to mutually benefit from each others' strengths. Harmonization can happen at various levels with different efforts. Based on the defined criteria and the comparison, the following possibilities for convergence are identified and envisioned.

6.3.1. Conversion between abstract descriptions of the user interface

The three systems cover different application areas (see Section 6.2). To mutually benefit from their advantages and to allow an interchangeable usage of the three systems, an interface for interlinking the systems on any level should be sought. Providing a single, common, abstract way to describe of the user interaction and interface is not considered to be reasonable, since all concepts have advantages and drawbacks and hence a harmonization would lead to a loss of flexibility. The abstract description is realized in different forms and thus a possible compromise is the creation of (semi-) automated conversion tools.

The idea is to use the CTT notation for the description of the task model, because there is good tool support for the generation and also a runtime simulator available (CTTE [22]). CTT could be the input for the generation of a user interface socket description, which could be used to connect manually designed user interfaces. Conversely, task models or an AUI in MariaXML could be generated from the socket description which could, in turn, be the starting point for the automatic user interface generation. The same conversion tools could be developed to create a user interface description using XForms. All these conversion tools would allow the front-end of one system to be used by the others. In summary, this would finally allow for the use of manually designed user interfaces *and* automatic generated ones within all three approaches.

6.3.2. Standardization of context models and common set of tools for maintenance

All three approaches use similar context models covering information about the user, the runtime platform and the environment, but unfortunately based on different formats and standards. GPII develops a register of terms for the description of user preferences and platform and environmental characteristics, which will result in a future version of ISO/IEC 24751. Using common terms and values for the models including user preferences, device characteristics and environmental factors would allow for the exchange (import/export) of user preferences and device characteristics between the platforms. Furthermore, the connection of systems for sharing environmental context would be possible, e.g., using the universAAL context bus as input for any adaptations in the other systems. A community process for registration of common terms via the GPII registry server and using a common user preference server would support this idea. A vision of common context models is to share and use the same tools for the maintenance of a user's preference set and to access a common server with descriptions of device characteristics.

6.3.3. Sharing resources

Sharing resources and supporting third party contributions are of great importance for all three systems, but the provision and description of user interface resources is handled in distinct ways. A common vocabulary for the description of user interface resources via metadata, e.g., realized by key-value pairs, is envisioned to allow for the import and export of resources such as labels, images and videos from one platform to the other. A starting point for the creation process can be the resource property vocabulary [48] by openURC. Besides sharing the resources between systems, a common description would offer the possibility to use a joint platform as a common resource server for upload-

ing and downloading resources for user interface adaptations via user interface integration.

6.4. Limitations of the study

Finally, a number of important limitations need to be noted. First, this comparative study does not provide quantitative data for the comparison of the systems. The definition of benchmarks and metrics would allow a more objective comparison, but were out of scope since the focus was on a conceptual comparison. An evaluation through competitive benchmarking, like in the EvAAL competitions for AAL systems [1], would help to obtain quantitative data with respect to e.g., effort needed to use the systems, usability of the (generated) user interfaces, et cetera. Furthermore, the comparison focuses on technical aspects and does not include means to compare the systems from the perspective of the user. Finally, the developers' perspective is not analyzed from a cost perspective in our comparison. To do so, an empirical study would be needed, which is also out of scope of the presented work.

7. Conclusion

The purpose of the current study was to provide a generic framework for the design of flexible user interfaces, and to analyze the three systems AALuis, GPII/URC and universAAL within this framework in a detailed comparative study. In this investigation, the aim was to assess differences and similarities of the systems based on ten criteria. As a result, the best-suited application areas, system constraints and possibilities of harmonization of the systems were elaborated.

The following conclusions can be drawn from the present study. First, although independently developed, all three approaches show similarities especially with respect to the context of use, the resource provision, the adaptation processes and the commitment to standards. Second, there are differences in the realization of these issues, and especially the tools provided to support the developer and the user. Finally, the systems differ greatly in their abstract description of the user interaction and interface, and the automatic generation of user interfaces or the usage of pluggable user interfaces.

These findings provide the following insights for future research. A common, abstract way to describe the user interaction and user interface would be desirable, but perhaps unattainable, due to the various strengths of the different approaches. The creation of (semi-) automatic conversion tools is envisioned, to retain the strengths of all concepts. Concerning context of use and the context models, a joint standardization effort is needed to allow for the exchange between different systems. This would yield a new opportunity to use a common set of tools for the maintenance of a user's preference set, to access a common server with descriptions of device characteristics and to share environmental context. Finally, a common format for the description of resources is required to allow for import and export of resources such as labels, images and videos from one platform to the other.

To overcome the limitations of a conceptual comparison and allow for a quantitative comparison, benchmarks and metrics for an objective comparison should be part of future work. Furthermore, the user's point of view regarding usability and accessibility of the user interfaces, and the ease of use for developers to get an impression of cost efficiency of the systems should be evaluated in additional (empirical) studies.

All future efforts targeting a harmonization of the systems can provide valuable input to, can be influenced by and should be coordinated with the W3C working group on model-based user interfaces [45].

Acknowledgements

The comparison of the different design systems for flexible user interfaces was initiated by the authors after a number of presentations and discussions about the three systems at various workshops and conferences (dissemination activities by the associated projects). The idea was to deepen the exchange of concepts and to intensify the discussions about collaboration possibilities. Based on this, the study at hand was started by an intensive face-to-face workshop which focused mainly on the technical aspects of user interfaces and their design. The results were summarized and used as a starting point for this joint paper, which aims at formalizing the results thus revealed. The paper was further developed by several bilateral meetings, discussions and several virtual workshops.

The project AALuis is co-funded by the AAL Joint Programme (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).

Work on URC and GPII has been funded by the US Dept. of Education, NIDRR, under Grants

H133E030012 and H133E080022 (RERC on IT Access), and by the European Commission, under the FP6 Grant Agreement 033502 (i2home) and the FP7 Grant Agreement 289016 (Cloud4All), and by the Saarland government under the contract T/2-EFI-001-04/05/2013 (SUCH).

Regarding universAAL, the research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 247950 (universAAL).

The opinions in this paper are those of the authors and not necessarily those of the funding agencies.

References

[1] J.A. Álvarez-García, P. Barsocchi, S. Chessa and D. Salvi, Evaluation of localization and activity recognition systems for ambient assisted living: The experience of the 2012 EvAAL competition, *Journal of Ambient Intelligence and Smart Environments* **5**(1) (2013), 119–132.

[2] J.M. Boyer, XForms 1.1. W3C Recommendation 20 October 2009, http://www.w3.org/TR/xforms/, Oct. 2009 [accessed: 02/2014].

[3] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon and J. Vanderdonckt, A unifying reference framework for multi-target user interfaces, *Interacting with Computers* **15** (2003), 289–308.

[4] CEA, CEA-2014-A: Web-based Protocol and Framework for Remote User Interface on UPnP Networks and the Internet (Web4CE), Technical report, 2007.

[5] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May and R.M. Young, Four easy pieces for assessing the usability of multimodal interaction: The care properties, in: *InterAct*, Vol. 95, 1995, pp. 115–120.

[6] C. Criteria, Common Criteria, http://www.commoncriteriaportal.org/, Feb. 2014 [accessed: 02/2014].

[7] E. den Ouden, Development of a design analysis model for consumer complaints: Revealing a new class of quality problems, PhD thesis, Technische Universiteit Eindhoven, Eindhoven, March 2006.

[8] B. Dumas, D. Lalanne and S. Oviatt, Multimodal interfaces: A survey of principles, models and frameworks, in: *Human Machine Interaction*, D. Lalanne and J. Kohlas, eds, Lecture Notes in Computer Science, Vol. 5440, Springer, Berlin, Heidelberg, 2009, pp. 3–26.

[9] ETSI, ETSI ES 202 746, Human Factors (HF), Personalization and User Profile Management, User Profile Preferences and Information, Technical report, Feb. 2010.

[10] A.M.M. Gil, D. Salvi, M.T.A. Waldmeyer, P.A. Jimenez and A. Grguric, Separating the content from the presentation in AAL: The universAAL UI framework and the swing UI handler, in: *Ambient Intelligence-Software and Applications*, Springer, 2013, pp. 113–120.

[11] S. Hanke, C. Mayer, O. Höftberger, H. Boos, R. Wichert, M.-R. Tazari, P. Wolf and F. Furfari, universAAL – an open and consolidated AAL platform, in: *Ambient Assisted Living*, R. Wichert and B. Eberhardt, eds, Springer, Berlin, Heidelberg, 2011, pp. 127–140.

[12] ISO/IEC, ISO/IEC 24751-1:2008, Information technology – individualized adaptability and accessibility in e-learning, education and training – Part 1: Framework and reference model, Technical report, Sept. 2008.

[13] ISO/IEC, ISO/IEC 24751-2:2008, Information technology – individualized adaptability and accessibility in e-learning, education and training – Part 2: 'Access for all' personal needs and preferences for digital delivery, Technical report, Sept. 2008.

[14] ISO/IEC, ISO/IEC 24752-1:2014, Information technology – user interfaces – universal remote console, Part 1: General framework, 2014.

[15] J.A. Larson, T. Raman, D. Raggett, M. Bodell, M. Johnston, S. Kumar, S. Potter and K. Waters, W3C multimodal interaction framework, W3C NOTE, 6, 2003 [accessed: 09/2014].

[16] Q. Limbourg, J. Vanderdonckt et al., *Comparing Task Models for User Interface Design*, Vol. 6, Lawrence Erlbaum Assoc., 2004, pp. 135–154, Chapter 6.

[17] N. Marquardt and S. Greenberg, Informing the design of proxemic interactions, *Pervasive Computing, IEEE* **11**(2) (Feb. 2012), 14–23.

[18] C. Mayer, M. Morandell, M. Gira, K. Hackbarth, M. Petzold and S. Fagel, AALuis, a user interface layer that brings device independence to users of AAL systems, in: *Computers Helping People with Special Needs*, K. Miesenberger, A. Karshmer, P. Penaz and W. Zagler, eds, Lecture Notes in Computer Science, Vol. 7382, Springer, Berlin, Heidelberg, 2012, pp. 650–657.

[19] C. Mayer, M. Morandell, M. Gira, M. Sili, M. Petzold, S. Fagel, C. Schüler, J. Bobeth and S. Schmehl, User interfaces for older adults, in: *Universal Access in Human-Computer Interaction. User and Context Diversity*, C. Stephanidis and M. Antona, eds, Lecture Notes in Computer Science, Vol. 8010, Springer, Berlin, Heidelberg, 2013, pp. 142–150.

[20] C. Mayer, M. Sili, M. Gira, M. Morandell, S. Fagel, A. Hilbert, C. Schüler and I. Cernei, Avatar enriched user interfaces for older adults, in: *GLOBAL HEALTH 2013, the Second International Conference on Global Health Challenges*, 2013, pp. 1–4.

[21] G. Meixner, M. Seissler and M. Orfgen, Specification and application of a taxonomy for task models in model-based user interface development environments, *International Journal on Advances in Intelligent Systems* **4**(3 and 4) (2012), 388–398.

[22] G. Mori, F. Paternò and C. Santoro, CTTE: Support for developing and analyzing task models for interactive system design, *IEEE Trans. Softw. Eng.* **28**(8) (Aug. 2002), 797–813.

[23] B. Myers, S.E. Hudson and R. Pausch, Past, present, and future of user interface software tools, *ACM Transactions on Computer-Human Interaction (TOCHI)* **7**(1) (2000), 3–28.

[24] A. Norcio and J. Stanley, Adaptive human-computer interfaces: A literature survey and perspective, *IEEE Transactions on Systems, Man and Cybernetics* **19**(2) (Mar. 1989), 399–408.

[25] openURC Alliance, Index of Technical Reports, http://www.openurc.org/TR/, Dec. 2013 [accessed: 02/2014].

[26] OpenURC, OpenURC Alliance e.V., http://www.openurc.org/, March 2013 [accessed: 03/2014].

[27] openURC, Resource Property Vocabulary 1.0. Latest specification, http://openurc.org/TR/res-prop-vocab1.0/, Dec. 2013 [accessed: 02/2014].

[28] openURC, Resource Server HTTP Interface 1.0 (DRAFT), http://www.openurc.org/TR/res-serv-http1.0-20131126/, Nov. 2013 [accessed: 02/2014].
[29] R. Oppermann, Adaptively supported adaptability, *International Journal of Human-Computer Studies* **40**(3) (1994), 455–472.
[30] M. Oshry, R. Auburn, P. Baggia, M. Bodell, B. David, D.C. Burnett, E. Candell, J. Carter, S. McGlashan, A. Lee, B. Porter, and K. Rehor, Voice Extensible Markup Language (VoiceXML) 2.1, http://www.w3.org/TR/voicexml21/, June 2007 [accessed: 02/2014].
[31] F. Paterno, *Model-Based Design and Evaluation of Interactive Applications*, 1st edn, Springer-Verlag, London, UK, 1999.
[32] F. Paternò, ConcurTaskTrees: An engineered approach to model-based design of interactive systems, in: *The Handbook of Analysis for Human-Computer Interaction*, Lawrence Erlbaum Associates, 2002, pp. 483–500.
[33] F. Paternò, C. Mancini and S. Meniconi, ConcurTaskTrees: A diagrammatic notation for specifying task models, in: *INTERACT 1997 Proc. of the IFIP TC13 International Conference on Human-Computer Interaction*, Chapman & Hall, 1997, pp. 362–369.
[34] F. Paternò and C. Santoro, A logical framework for multidevice user interfaces, in: *Proc. of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM, 2012, pp. 45–50.
[35] F. Paternò, C. Santoro and L.D. Spano, MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments, *ACM Trans. Comput.-Hum. Interact.* **16**(4) (Nov. 2009), 19:1–19:30.
[36] M. Peissner, D. Häbe, and A. Schuller, MyUI deliverable D2.2. Adaptation concept and Multimodal User Interface Patterns Repository, Technical report, 2012, Available at http://www.myui.eu/deliverables/MyUI_D2-2_final.pdf.
[37] D. Raneburger, G. Meixner and M. Brambilla, Platform-independence in model-driven development of graphical user interfaces for multiple devices, in: *Software Technologies*, J. Cordeiro and M. van Sinderen, eds, Communications in Computer and Information Science, Vol. 457, Springer, Berlin, Heidelberg, 2014, pp. 180–195.
[38] K. Sousa, H. Mendonça and J. Vanderdonckt, Towards method engineering of model-driven user interface development, in: *Task Models and Diagrams for User Interface Design*, Springer, 2007, pp. 112–125.
[39] C. Stocklöw, A. Grguric, T. Dutz, T. Vandommele and A. Kuijper, Resource management for multimodal and multilingual adaptation of user interfaces in ambient assisted living environments, in: *Universal Access in Human-Computer Interaction. Applications and Services for Quality of Life*, C. Stephanidis and M. Antona, eds, Lecture Notes in Computer Science, Vol. 8011, Springer, Berlin, Heidelberg, 2013, pp. 97–106.
[40] M.-R. Tazari, F. Furfari, Á. Fides-Valero, S. Hanke, O. Höftberger, D. Kehagias, M. Mosmondor, R. Wichert and P. Wolf, The universAAL reference model for AAL, in: *Handbook of Ambient Assisted Living*, Vol. 11, 2012, pp. 610–625.
[41] P. Thakur and B. Rosa, URC-HTTP Protocol 2.0. Draft Technical Report 2012-10-22, OpenURC Alliance, http://openurc.org/TR/urc-http-protocol2.0-20121022/index.html, Oct. 2012 [accessed: 02/2014].
[42] J. Tidwell, *Designing Interfaces*, O'Reilly, 2010.
[43] G. Vanderheiden and G. Zimmermann, Use of user interface sockets to create naturally evolving intelligent environments, in: *Proc. of the 11th Int. Conf. on Human-Computer Interaction (HCII 2005)*, 2005.
[44] G.C. Vanderheiden, J. Treviranus, M. Ortega-Moral, M. Peissner and E. de Lera, Creating a Global Public Inclusive Infrastructure (GPII), in: *Universal Access in Human-Computer Interaction. Design for All and Accessibility Practice*, C. Stephanidis and M. Antona, eds, Lecture Notes in Computer Science, Vol. 8516, Springer International Publishing, Jan. 2014, pp. 506–515.
[45] W3C, Model-based user interfaces (MBUI) working group, http://www.w3.org/2011/mbui/, Feb. 2014 [accessed: 02/2014].
[46] W3C, The Forms Working Group, http://www.w3.org/MarkUp/Forms/, Jan. 2014 [accessed: 02/2014].
[47] W. Wahlster, Towards symmetric multimodality: Fusion and fission of speech, gesture, and facial expression, in: *KI 2003: Advances in Artificial Intelligence*, A. Günter, R. Kruse and B. Neumann, eds, Lecture Notes in Computer Science, Vol. 2821, Springer, Berlin, Heidelberg, 2003, pp. 1–18.
[48] W. Wahlster, *SmartKom: Foundations of Multimodal Dialogue Systems (Cognitive Technologies)*, Springer-Verlag Inc., New York, 2006.
[49] M. Weiser, The computer for the 21st century, *SIGMOBILE Mob. Comput. Commun. Rev.* **3**(3) (July 1999), 3–11.
[50] B. Wöckl, U. Yildizoglu, I. Buber, B. Aparicio Diaz, E. Kruijff and M. Tscheligi, Basic senior personas: A representative design tool covering the spectrum of European older adults, in: *Proc. of the 14th International ACM SIGACCESS Conference on Computers and Accessibility, ASSETS '12*, ACM, New York, NY, USA, 2012, pp. 25–32.
[51] B. Wöckl, U. Yildizoglu, I. Buber-Ennser, B. Aparicio Diaz, and M. Tscheligi, Elderly personas: A design tool for AAL projects focusing on gender, age and regional differences, in *Proc. of the 3rd AAL Forum: Partner-Ships for Social Innovations in Europe, Smart Homes 2012*, Lecce, Italy, September 26–28, 2011, pp. 125–130.
[52] G. Zimmermann, J.B. Jordan, P. Thakur and Y. Gohil, GenURC: Generation platform for personal and context-driven user interfaces, in: *Proc. of the 10th International Cross-Disciplinary Conference on Web Accessibility, W4A '13*, ACM, New York, NY, USA, 2013, pp. 6:1–6:4.