Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 98 (2016) 107 – 116

Procedia Computer Science

The 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)

Just a Smart Home or Your Smart Home – A Framework for personalized User Interfaces Based on Eclipse Smart Home and Universal Remote Console

Lukas Smirek*[a], Gottfried Zimmermann[a], Michael Beigl[b]

[a]*Stuttgart Media University, Nobelstr. 10, 70569 Stuttgart, Germany*
[b]*Karlsruhe Institute of Technology, Vincenz-Priessniz-Str. 1, 76131 Karlsruhe, Germany*

Abstract

In the last few years substantial progress has been made in smart home technologies, and promises to support and assist us in our daily life are higher than ever. This holds not only for regular users but also for people with special needs such as the elderly and people with disabilities. The appropriate design of smart homes can enable a more independent life for these users and can give them the chance to stay in their familiar environment for a longer period of time. Hence, the smart home concept can play an important role when addressing the demographic change that is present in most industrial countries. However, although technologies seem to be advanced and the expected benefits are high, a wide-spread adoption has not yet taken place. There are various reasons for this situation. Among them, the lack of appropriate user interfaces for the heterogeneous user group of future smart homes and the problem of low interoperability between different smart home systems can be mentioned. Two platforms addressing these problems are the Eclipse Smart Home (ESH) project and the Universal Remote Console (URC). ESH focuses on the integration of different device and back-end technologies; URC provides a personalized, pluggable user interface. This paper analyzes the similarities and differences between the two systems. As a result of the analysis, a concept for integrating the ideas of URC into the ESH project is proposed. This concept is a first step towards a platform for personalized user interfaces in the Smart Home and Ambient Assisted Living domain.

Peer-review under responsibility of the Program Chairs

Keywords: Smart Home, AAL; Personalisation; User Interface; Eclipse Smart Home; Universal Remote Console;

* Corresponding author. Lukas Smirek. Tel. +49 711 8923-2672. *E-mail address:* smirek@hdm-stuttgart.de

Peer-review under responsibility of the Program Chairs
doi:10.1016/j.procs.2016.09.018

1. Introduction

Nowadays, words like Internet of Things (IOT), Smart Home or Ubiquitous Computing are no longer of academic interest only. The emergence of these concepts was enabled by the continuous growth of interconnected, electronic devices in our everyday life. Possible use cases range from fancy and entertaining ones (e.g., video/audio distribution) to such that are helpful for everyone (e.g., energy management) but also ones that enable a more participatory life for the elderly or people with disabilities. Especially smart homes and the related field of Ambient Assisted Living with its wide spectrum of assistive functions can enable a longer independent life at home and will play an important role when coping with the demographic change that is taking place in most industrial countries.

However, although the technological base for these concepts seems to be established[1] and the promised benefits are high, reality is still lagging behind expectations and a wide-spread adoption has not yet taken place.

From the authors' perspective, the following reasons are of special importance. First of all, the focus of the research community has been rather on what is technologically possible[1,2,3] than on what the real user needs are. In addition, research has neglected the topic of appropriate user interfaces[4]). Since the user group of future smart homes will reflect the full range of our society, there is a need for personalized user interfaces that take the individual user requirements and preferences into account.

The second problem is the lack of interoperability of different smart home systems. The smart home market is just evolving and therefore it is very difficult for a user to decide which one to choose. Furthermore, it is likely that a user gains the highest value by integrating different systems. Along with the problem of low compatibility between different systems comes the fact that devices and backend technologies overarching user interaction concepts are missing.

Due to these reasons a framework is needed, addressing on the one hand the integration of different backend technologies, and on the other hand the provisioning of device overarching and personalized user interfaces. In order to realize such a system, the Eclipse Smart Home project[5] (ESH) and the Universal Remote Console[6] (URC) were chosen for investigation.

The URC framework was chosen because two of the authors are involved in the related development and standardization processes. ESH was selected, because just like the URC runtime implementation it follows the approach of a central gateway instead of relying on a distributed operating system, as some other IOT platforms do. Frameworks such as the AllJoyn framework locate their code directly on the target devices. A similar architecture with a central gateway makes the integration of the two frameworks easier. Another argument for our choice was the open source nature of URC and ESH; in addition, both projects are driven by communities instead of industry..

The remainder of this paper is structured as follows. In the following section the needs and requirements for personalized user interfaces are introduced on the basis of some illustrative use cases. Section 3 introduces the core concepts of ESH and URC. Section 4 contains our analysis on the two systems. In section 5, we propose a concept on how to transfer the ideas and major benefits from the URC framework to the ESH project.

2. Requirements for personalized user interfaces

This paper aims to evaluate the personalization features of URC and ESH with a special focus on graphical user interfaces. In the first part of the evaluation, a comparison of the systems' architectures is done with regard to the available components and general features. In the next step, the systems' abstraction models for describing connected devices and services are compared with regard to their structure and expressiveness.

. The need for abstract representations of physical devices can be illustrated by the following use cases:

- Frequently, elderly people are familiar with a specific device and have problems to adjust to a new one. Since any device will sooner or later get broken (e.g., washing machine, HVAC system), it is beneficiary for people to keep their familiar user interface. In order to do so, a separation between the physical device and its abstract representation in the smart home system is required *(U1)*.
- Such a separation enables many other use cases, among them supporting users who became paraplegic by

an accident. In such a case, the user and their family would want to stay in their familiar home instead of moving into a new home with special equipment. Rather than exchanging a device, just for the sake of an inaccessible user interface, it is more cost-effective and user-friendly to provide an alternative user interface only (e.g. exchange or supplement a touch-screen control panel with one supporting eye tracking) *(U2)*.

- Usually, smart homes are inhabited by multiple people with different needs, e.g. people with disabilities or children who should be given limited access to certain functionalities. Having a common abstract layer, it is possible to connect different, personalized user interfaces to it at the same time *(U3)*.

However, sometimes not only exchangeable but also adaptive user interfaces are required. The adaptation of user interfaces can take place on different levels[10]. Some of them like adjusting contrast or font size, as well as taking the screen size into account or giving the user interface the look and feel of a native one of the rendering platform, can easily be done by the controller device at runtime. Hence this is out of the scope of our considerations. However, when user interfaces should be provided in different languages for people with disabilities or with icons for different cultural areas (*U4)*, it is necessary to exchange some parts of the user interface content. In order to make these supplementary user interface components available to a large user group and independent of location, a central repository for UI components is required[11].

Also, it is advantageous to give third parties the chance to provide their own solutions for a narrower user group *(U5)* (rare language, sign videos for deaf people or other Assistive Technology (AT) solutions). Such a repository should be open and extendable. Furthermore, third parties should be able to contribute on a very modular basis[12]. Finally, it is considered how the *context of use,* necessary for any user interface adaptation is handled by the systems.

3. Technology overview

This section explains the most important features of URC and ESH to get a common understanding of the two frameworks. More extensive descriptions can be found in [5] and [13].

3.1. Universal Remote Console

Universal Remote Console (URC[6, 14]) is a framework that was designed from its inception to enable personalized and exchangeable user interfaces[9]. The main idea of URC is to enable every user to control any device or service (*target*) with the user interface fitting their needs best. Targets can range from TV sets over kiosk to weather services. Therefore, every target exposes an abstract description of its operational user interface – the *user interface socket description* (short *socket description*) – that serves as a reliable contract between the target and any developed UI. Along with socket descriptions and their contents, further UI resources like labels and help texts in various languages or pictures can be defined, for the purpose of UI rendering. Resources can either be stored directly on a target or on a dedicated, globally accessible *resource server*. At runtime, any controller can connect to a target, read its socket description and use related resources (possibly from third parties) to render a personalized user interface. Targets that are not compliant to the ISO/IEC standard can be integrated via the Universal Control Hub[13] (UCH). This middleware solution downloads socket descriptions from the resource server and exposes them to any controller. The controller can connect to the UCH just like to a regular target via the URC HTTP Protocol[15] (see fig. 1).

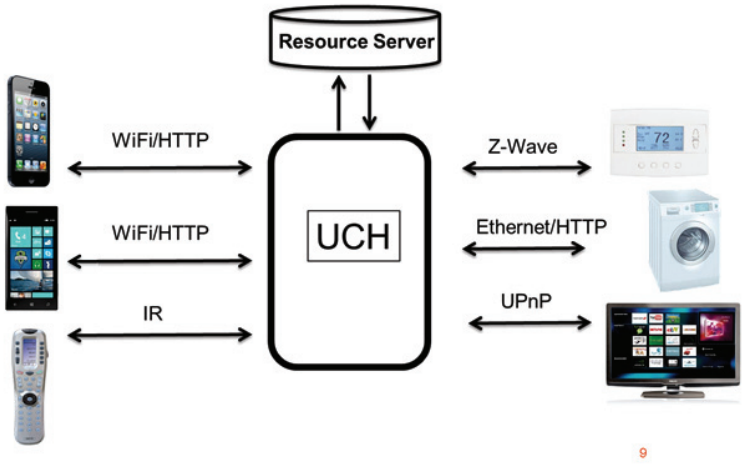

Fig. 1. Universal Control Hub (UCH) architecture. The UCH is the middleware between controllers (left) and targets (right). The UCH can download user interface resources from a resource server.

3.2. Eclipse Smart Home

ESH provides a flexible, modularized framework for smart home and ambient assisted living solutions with a focus on heterogeneous environments. Rather than defining a common communication protocol that must be supported by the different devices, ESH aims to cope with the currently very fragmented market for smart home systems and IoT devices. The provided modules form an abstraction and translation framework to enable use cases and interaction across system and protocol boundaries5. Fig. 2 provides an overview of the openHAB architecture which is the basis for ESH. Various devices or services e.g., TV set or weather service (things) can be connected to the framework, by loading specific bindings. Bindings implement a thing-specific protocol and are connected via an event bus to enable inter-component communication. A central item repository is also connected to the event bus. The stored variables (items) enable stateful interaction.

ESH defines a declarative model to describe things being connected to the ESH gateway. The model comprises the concepts of thing types and channel types. Every connected device or service is a thing that is of a certain thing type. Functionalities of things are described by channel types e.g., volume of a TV set or current temperature. Channel types should be of one of about currently 30 standard categories that add further semantic, useful for user interface rendering.

At runtime, as soon as a new thing is discovered, its channels are linked to an item of the item repository. This enables remote control by user interfaces or their connections via rules. Furthermore, the information given in a related channel type can be used to auto-generate a user interface. When rendering user interfaces, icons from different icon sets e.g., classic or modern can be taken into account. Icons are chosen depending on the channel category and the current value of the channel.

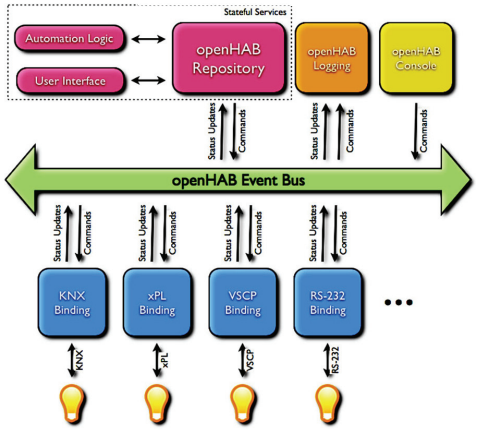

Fig. 2. The openHAB architecture (basis for ESH). The openHAB event bus connects consoles, logging facilities and the repository (top) with the devices and services of a Smart home via bindings (bottom).

4. Comparison of Universal Remote Console and Eclipse Smart Home

4.1. Architecture

Both the ESH framework and the URC reference implementation (UCH) are modular, Java-based systems. While ESH uses the OSGi framework, the UCH implements its own mechanism to load additional components. In order to be able to extend the system and to connect to new devices, the principles of modularization and hot deployment are used to load new components. Furthermore, both systems support discovery for new connected devices and services.

Elements like the item repository, the rule engine and the connecting event bus can only be found in the ESH framework. In the UCH reference implementation, stateful information must directly be stored in the target adapters. Currently, a rule engine or event bus is not yet implemented in the UCH. Still, the UCH allows for target-overarching user interfaces. However, communication between different targets and decision making processes must be directly implemented in the code of the user interface layer. In the same vein, a rule engine would need to reside on the controller, giving users the chance to configure their smart home according to their needs. While in ESH such a rule engine with its own syntax is available, in the URC framework rules must be hardcoded.

A concept like the URC resource server to store additional UI components (e.g., labels in different languages or different icon sets) and to support use cases like *(U4)* cannot be found in the ESH framework. Though, the ESH reference implementation openHAB 2 is available[16] as offline and online version. The online version automatically downloads new bindings from a remote GIT repository in case the added device or service is not supported by the locally-installed code. However, a concept for downloading user interfaces or parts of it related to the currently connected devices and services based on their abstract descriptions is not supported at runtime. Bindings containing icon sets can be also downloaded from the GIT repository. However, this is initialized by the user and not by the system – hence there is no notion of a user profile. Furthermore, the components contained in the GIT repository are not indexed for discovery. This is in contrast to the URC resource server where all user interface components are indexed and can be searched according to any given user profile or context of use.

4.2. User interface components and modelling of connected devices and services

Both the URC framework and the ESH framework define XML languages to describe connected devices and services in an abstract manner. This information can then be used for auto-generation of user interfaces, as well as a reliable contract for user interface developers. Although the concepts are similar in the idea of describing connected devices and services, they differ in their expressiveness and the way they can be used for user interface personalization. In URC, the concept of abstract descriptions is called *socket,* while in the ESH framework *thing*

types and *channel types* are mentioned. Since URC sockets contain information about a target's user interface rather than information about targets and their configuration, the equivalent of a socket is a channel-type or a channel group rather than a thing type.

However, as mentioned before, the concepts differ in their expressiveness. In both concepts, variables with their data types can be defined e.g., for indicating the current channel of a TV set. ESH uses a restricted set of datatypes only with a defined value set. Developers can add further restrictions to the datatypes like minimum and maximum values. On contrast, URC sockets employ the full expressiveness of XSD data types.

Furthermore, ESH defines only the concept of channel types that is almost equivalent to a variable in URC socket descriptions. In URC, socket descriptions contain variables for indicating a target's status, commands for synchronous interaction between controllers and targets and notifications for asynchronous interaction. Even more, URC socket descriptions can contain pre and post conditions in order to model tasks and their dependencies e.g., DVD can only be started when a disk is in the player's drawer.

In ESH, it is not possible to define commands. However, in some cases the data type of a channel also defines a command that is applicable. For example the data type *switch* can have the value "ON" and "OF" but it is also possible to send the command "toggle". However, developers can not define new ones.

Consequently, a more complex interaction between controller and controlee can be modeled with URC techniques than with the ones in the ESH framework.

Next, in the URC standard there is a strict separation between sockets and additional user interface resources. Socket descriptions and resources are always located in separate files. This is different in the ESH framework. Thing types and channel types typically contain standard labels and additional descriptions (in URC terms they would be separate resources). Nevertheless, it is also possible to provide labels and descriptions in the ESH framework in separate files; one file for each language.

The two systems also differ in their approach referring to the location of the abstract descriptions and additional UI components. In the ESH framework, thing type and channel type definitions are located directly in the bindings in a standardized folder. The same is true for additional user interface components like language files and icon sets (provided as separate OSGi bundles). This is quite different in the URC framework where supplemental user interface resources are stored on the resource server. This is of special interest when third parties, like user interface developers or AT experts, want to contribute supplemental resources for special user interfaces *(U5)*. It is not necessary to contribute a whole icon set. Instead, it is possible to add an atomic resource for one specific element of a socket description (e.g., a single label or pictogram for a trigger). Also, it is possible to add a complete user interface for one or several socket descriptions (e.g., a user interface for controlling the TV set as well as the DVD player). Furthermore, the format is completely open and not limited to SVG and JPG as it is in ESH. Therefore, URC has an advantage with regard to openness, modularity and expendability. Additional resources can be made available on a more fine-granular level via a simple upload to the resource server. In contrast, in the ESH framework alternative languages must be directly integrated in a binding, making the setup of a whole development environment necessary.

Another difference of the two frameworks is the way available descriptions of devices and services are organized. The idea of the URC framework is to build a hierarchical structure of standardized socket descriptions. The openURC Alliance[6] is working on a set of standard socket descriptions that can be extended by any third party via an inheritance mechanism. Thus, any hierarchy has an intrinsic semantic and user interfaces that are compatible to a class of targets are always compatible to its sub-classes. Hence, the exchangeability and portability of user interfaces is warranted while at the same time vendors of targets can still define a specific description of their product. While URC focuses on a hierarchical structure, the idea of ESH is to see devices and services as a set of fine-granular functions - the different channel types. Semantic can be added by linking Channel Types with categories.

4.3. Connecting User Interfaces

In order to support *(U1), (U2) and (U3),* it is necessary to connect alternative user interfaces or controller devices to a smart home system. ESH offers two possibilities for user interfaces to communicate with the main framework:

via a REST API and via user interfaces deployed in the OSGi container. Remote user interfaces executed on a client will usually use the REST API to connect to the framework. However, if a user interface is executed directly from the OSGi runtime, they can use all local Java services offered by the available OSGi bundles.

To connect controllers to any target or the UCH, the openURC Alliance defines a proprietary HTTP-based protocol[15]. While the ESH protocol is designed according to RESTful principles, controllers making use of the URC HTTP protocol must always establish a session with the controlled targets. This contradicts RESTful design principles. However, it has the advantage that controllers can set a context of use in the UCH to get a more personalized view on the connected targets and related user interfaces. For example, the UCH can make a pre-selection of user interfaces that are compatible to the controller device or set a language preferred by the user.

With the ESH REST API, it is also possible to access different icon sets. However, it is up to the controller to decide which icon set to use. In general, every decision influenced by the context of use must be taken on the controller.

Another advantage of enabling sessions is that it is possible to have personalized access to targets, e.g. when accessing a video on demand service. In the ESH framework all users get the same selection of movies for a certain provider. With the URC framework it is possible that depending on the connected controller (one of an adult and one of a child) a different selection of movies is provided.

4.4. User interfaces generation and personalization

In ESH as well as in the URC framework the information contained in the abstract descriptions of connected devices and services can be used to auto-generate graphical user interfaces. However, they differ in the degree to which they can be adjusted to the user and their needs & preferences.

The standard UCH ships with the JavaScript web client library[17] that renders a simple tabular user interface for each target, based on the information found in the pertaining socket description. The generator renders a separate user interface for every connected target. The ESH website announces the classic UI as the currently available user interface generator. It takes all items contained in the item repository and with that all connected things into consideration. This enables a more holistic view on the system than the URC web client library. In case that the items were structured by an administrator in groups (e.g. separate groups for light switches of different rooms), this can also be taken into consideration.

However, currently auto-generated user interfaces are nice for getting a first and quick overview of a system (e.g., for testing purposes). Nevertheless, in order to take different user needs into account or to build task-oriented user interfaces, they are limited in their capabilities. Hence, the ESH framework with its REST API, as well as the UCH with the URCHTTP protocol provide means to connect any user interface developed by any third party. Such user interfaces should not be rigid but should rather provide a skeleton that can be personalized e.g., exchanging pictograms according to the user's cultural background

As described by[10], user interface adaptation can take place on different levels. The adaptation mechanisms to control the upper levels (Presentation & Input events and Structure & Grammar) like adjusting font size, contrast or exchanging list boxes against radio buttons must be implemented in the controller of the user interface software and are out of scope of the smart home platform. However, the exchange of UI components on the content level must be supported by the smart home platform or at least by the related infrastructure *(U4)*.

The URC framework implements such an infrastructure through the concepts of user interface resources and the resource server. Hence, URC is predestined for providing alternative user interface contents. Every user interface resource is an alternative content that is stored and indexed on the resource server. As such, it can be used by any controller for rendering a personalized user interface thereby taking the context of use into account. It must be emphasized that resources can be more than simple labels or help texts. They can also be highly specialized resources like images for different culture areas, sign language videos for deaf users or simplified texts and pictograms for people with dyslexia *(U4)*.

ESH does not provide such a sophisticated infrastructure. However, it also provides some possibilities for exchanging user interface content. By making use of the Java internationalization mechanism, all texts can be

displayed in different languages. In order to do this for each language, a property file must be placed in the related binding. ESH defines a standard key scheme to reference all XML nodes in binding and thing type describing XML files. These schemes can be used inside the language property files to reference the XML nodes of interest and to add to them an alternative text value in different languages. Also, different icon sets can be provided as bindings. However, as mentioned above, all alternative user interface resources must be compiled into bindings and are stored locally. Furthermore, their format is limited to different languages and images in JPG and SVG format. On the opposite, the URC framework does not expose any restrictions on user interface resource formats.

4.5. Context of use

The URC standard supports the setting of a context of use for every user. This includes a user's needs and preferences, the characteristics of their personal controller and runtime platform, and the environment of the use situation. However, the current web client does not support a context of use. Resources on the resource server are indexed properly so that they can be searched and used by a more sophisticated user interface renderer taking the context of use into account. The ESH runtime can take environmental conditions to a certain degree into account by defining appropriate rules. However, it is not possible to set an individual context of use for every user – for example the language can only be set on the system but not on the user level.

4.6. Summary

The URC framework and the ESH framework have a lot in common. However, they do not totally overlap in their concepts and hence can complement each other. The most important similarity is that both platforms try to integrate different backend technologies and to provide abstract descriptions of connected devices and services.

Large differences between the systems are in the amount of supported backend technologies. Thanks to its strong community support, the reference implementation of ESH – openHAB – is more advanced in this aspect than the URC infrastructure.

On the other hand, the URC framework with the implementation of a resource server has a more advanced concept for providing personalized user interfaces to the user. Along with this concept comes a way for third parties to contribute further and specialized user interfaces. Additional user interface resources can be developed on a more fine-granular level and with more flexibility as in ESH. User interface resources are indexed in order to make them searchable and to take the context of use into account that can be set in the URC framework for every individual user.

Table 1: Summary of comparison between ESH and URC, looking at various aspects.

Comparison aspect	**ESH**	**URC/UCH**
Inter-component communication	Event bus, rule engine	Implementation in UI code
Interface for external components	REST API	Stateful URC-HTTP protocol
User interface (component) storage	Local, in bindings	Resource server
Alternative UI resource formats	JPG, SVG, different languages	No restrictions
Context of use	Only in UI	Supported in UCH
Abstraction concepts	Channel types	Variables, commands, notifications, pre & post conditions
Sessions	None	Between controller and UCH (user interface personalization) and on target level (target personalization)

5. Future work

In our future work we will work on an integration of the URC concepts into the ESH framework. Overall, the new additions to ESH must not hinder backward compatibility of existing ESH applications. A major step in this direction is the introduction of a resource server into ESH. To communicate with the resource server, a RESTful protocol is already under development. For a tight integration with current ESH development processes, only the links to the UI resources on the ESH Github repository will be stored on the resource server rather than the resources themselves. One possibility to implement the communication with the resource server is to develop an icon set binding that dynamically downloads appropriate user interface components from the related repository. However, this would restrict formats to JPG and SVG, and make the consideration of context of use cumbersome.

Hence an extension of the current ESH REST API is planned. to not hinder backward compatibility none of the current REST end points will be removed. Instead, new ones will be added. These new end points will connect to the old ones via hyperlinks.

In order to set a context of use, every user must be modelled as a separate resource with all channels they are allowed to access as sub-resources. Furthermore, every channel must provide a URL under which the currently relevant user interface resources can be found. Upon calling on this URL, resources are either returned from a cache or are downloaded from the resource server.

In order to model different user preferences, work towards an integration with the Global Public Inclusive Infrastructure (GPII)[18] has been planned. GPII provides a mechanism to transfer platform-independent user preferences via a cloud-based infrastructure to various target systems, such as ticket machines, kiosks or library PCs. Furthermore, different matchmakers (statistical and rule-based) were developed in order to match the user preferences with the user interface technologies available on the various target systems. Currently, there are plans to integrate the matchmakers with the resource server in order to find the best fitting user interface resources for a given context of use.

6. Discussion

So far, the main focus of the investigation has been on graphical user interfaces. However, control of smart homes is frequently associated with implicit ways of human computer interaction and disappearing user interfaces. Of course, these concepts can also be personalized. The concept of the resource server is definitely not restricted to graphical user interfaces only. However, further investigations are necessary into what kinds of resources can be stored and indexed on it.

Beside these considerations, a platform for personalized user interfaces that is based on a resource server and abstract descriptions for connected devices requires some future research in the fields of security and the acceptance of abstract user interfaces. A resource server that is open for contributing well designed and helpful additional UIs also opens the door for misuse and the injection of malware. A solution to cope with that problem could be to establish a review process, as it is done in some existing app stores. New contributions must pass this review process and some tests before they are available to the public.

Another question is how well abstract user interfaces will be designed and supported by software developers. Appropriate tools are needed in order to simplify the creation process and to achieve an acceptable quality level. A high quality level is also of special importance when further possibilities for automated generation of user interfaces are examined or implemented. The above analysis of the ESH and URC framework shows that there are enough similarities between the systems for a future integration, and there are benefits on either side that are not available on the other side.

The common ground for a future integration of ESH and URC are first, the idea of integrating different backend technologies, and second, the provisioning of abstract descriptions for the connected devices and services. The two systems are complementary to each since URC can profit from the well-established community of ESH and its status as an official Eclipse project, as well as from the large amount of supported backend technologies. At the

same time, ESH can benefit from URC by adopting the concept of a resource server for user interface resources. Such a contribution to the ESH framework would facilitate a platform for flexible and open user interfaces. This would enable users to download and use user interfaces according to their personal preferences and needs, and appropriate to the context of use. Furthermore, third parties would be able to design supplemental user interfaces.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Program (FP7/2007-2011) under grant agreement no. 610510, Prosperity4All ("Ecosystem infrastructure for smart and personalized inclusion and PROSPERITY for ALL stakeholders"). This publication reflects only the authors' views and the European Union is not liable for any use that may be made of the information contained herein.

References

1. Whitmore A., Agarwal A., Li Da Xu, *The Internet of Things - A survey of topics and trends*
2. Solaimani S., Keijzer-Broers W., and Bouwman H. *What we do and don´t know about the smart home: an analysis of the Smart Home literature,* Indoor and Built Environment, 2013
3. Chan, M. Estve, D. Escriba C., and Campo E. *A review of smart homes - Present state and future challenges," Computer Methods and Programs in Biomedicine,* vol. 91, no. 1, 2008, pp. 55–81,
4. Mavrommati I. and Darzentas J. *An overview of AmI from a user centred design perspective in Intelligent Environments*, 2nd IET International Conference, vol., 2006, pp. 81–88.
5. Eclipse Smart Home - A Flexible Framework for the Smart Home, http://www.eclipse.org/smarthome/ (last accessed: 31.01.2016)
6. OpenURC Alliance. OpenURC, (last accessed 02. 2015), http://www.openurc.org.
7. Kreuzer K. *openHAB 2.0 and Eclipse Smart Home*; 16.06 2014 by; http://kaikreuzer.blogspot.de/2014/06/openhab-20-and-eclipse-smarthome.html (last access: 31.01.2016)
8. openHAB: http://www.openhab.org
9. Zimmermann G. *Towards coherent, task-oriented and scalable user interfaces in the home environments*. In proceedings of 3rd IET International Conference on Intelligent Environments (IE07), 24-25.09. 2007, Ulm University, Germany (p. 36ff). Retrieved from http://www.accesstechnologiesgroup.com/pubs/Zimmermann2007-IE07/
10. Zimmermann, G., Vanderheiden, G., & Strobbe, C. *Towards deep adaptively a framework for the development of fully context-sensitive user interfaces*. In Universal Access in Human-Computer Interaction. Design and Development Methods for Universal, Springer, 2014, pp. 299-310.
11. Zimmermann, G., Wassermann, B. *Why we need a user interface resource server for intelligent environments*. In Workshops Proceedings of the 5th International Conference on Intelligent Environments, 2009, vol. 4, pp. 209–216. IOS Press. http://doi.org/10.3233/978-1-60750-056-8-209
12. Peissner M. et al. *Requirements for the successful market adoption of adaptive user interfaces for accessibility.* Universal Access in Human-Computer Interaction. Design for All and Accessibility Practice. s.l. : Springer, 2014, pp 413-442.
13. Zimmermann G., Vanderheiden G. *The Universal Control Hub: An Open Platform for Remote user interfaces in the DigitalHome,* Springer LNCS. Human-Computer Interaction. Interaction and Techniques, vol. 4551/2007, 2007, pp. 1040–1049, ZimmermannVanderheiden2007-HCII/ (last accessed: 2016-05-15).
14. SO/IEC 24752 (Information Technology user interfaces Universal Remote Console 5 parts, 204
15. URC-HTTP Protocol 2.0 (ATR), http://www.[6.org/TR/urc-http-protocol2.0-20131217/
16. https://github.com/openhab/openhab-distro/blob/master/docs/sources/getting-started.md
17. Webclient JavaScript Library. http://www.[6.org/tools/webclient-generic-html-3.1/
18. Home | gpii.net. http://gpii.net/