## D203.1 Runtime Environment Final Prototype

<table>
<thead>
<tr>
<th>Project Acronym</th>
<th>Prosperity4All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grant Agreement number</td>
<td>FP7-610510</td>
</tr>
<tr>
<td>Deliverable number</td>
<td>D203.1</td>
</tr>
<tr>
<td>Work package number</td>
<td>WP20</td>
</tr>
<tr>
<td>Work package title</td>
<td>Collaborative development tools/Environments T203.3 Runtime Environment</td>
</tr>
<tr>
<td>Authors</td>
<td>Christos Mettouris, Marios Komodromos (UCY), Lukas Smirek (HDM), Daniel Ziegler (FHG)</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>Public/Consortium</td>
</tr>
</tbody>
</table>
Ecosystem infrastructure for smart and personalised inclusion and PROSPERITY for ALL stakeholders
www.prosperity4all.eu

Keyword List

Integrated Runtime Environment, Integration, AsTeRICS Runtime Environment, ARE, URC, UCH, MyUI, REST, URC-HTTP protocol, SSE events, Use Case Scenario

Version History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
<th>Organisation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/06/2016</td>
<td>Christos Mettouris</td>
<td>UCY</td>
<td>Initial authoring (v1)</td>
</tr>
<tr>
<td>2</td>
<td>06/07/2016</td>
<td>Christos Mettouris</td>
<td>UCY</td>
<td>Authoring of v2</td>
</tr>
<tr>
<td>3</td>
<td>07/07/2016</td>
<td>Daniel Ziegler</td>
<td>FHG</td>
<td>Authoring/updating of MyUI and Integration Sections</td>
</tr>
<tr>
<td>4</td>
<td>08/07/2016</td>
<td>Lukas Smirek</td>
<td>HDM</td>
<td>Authoring/updating of UCH and Integration Sections</td>
</tr>
<tr>
<td>5</td>
<td>08/07/2016</td>
<td>Christos Mettouris, Marios Komodromos</td>
<td>UCY</td>
<td>Authoring of v3</td>
</tr>
<tr>
<td>6</td>
<td>11/07/2016</td>
<td>Christos Mettouris, Marios Komodromos</td>
<td>UCY</td>
<td>Final updates</td>
</tr>
<tr>
<td>7</td>
<td>19/07/2016</td>
<td>Till Riedel, Eva de Lera</td>
<td>KIT, RtF</td>
<td>Internal Review</td>
</tr>
<tr>
<td>8</td>
<td>23/07/2016</td>
<td>Christos Mettouris, Marios Komodromos</td>
<td>UCY</td>
<td>Authoring of Final Draft (v4) - Updates and additions after internal review</td>
</tr>
<tr>
<td>9</td>
<td>29/07/2016</td>
<td>Christos Mettouris, Marios Komodromos, Lukas Smirek, Daniel Ziegler</td>
<td>UCY, HDM, FHG</td>
<td>Authoring of Final Version (v5) – Final Updates</td>
</tr>
</tbody>
</table>
# Table of Contents

Executive Summary ........................................................................................................... 1

1  Aim of the document..................................................................................................... 2

2  Contribution to the global architecture ....................................................................... 6

3  The AsTeRICS Framework Architecture .................................................................... 8

   3.1 Aim ............................................................................................................................. 8
   3.2 Background ................................................................................................................ 8
   3.3 The Original AsTeRICS Architecture .................................................................... 10
   3.4 Architectural Updates and Developments to the AsTeRICS Architecture within Prosperity4All: Rest Services ............................................................................... 12

4  Universal Remote Console (URC) .............................................................................16

   4.1 Aim ............................................................................................................................ 16
   4.2 Background ............................................................................................................... 16
   4.3 Original architecture ................................................................................................ 17
      4.3.1 ISO/IEC 24752 compliant communication .................................................... 17
      4.3.2 Integration of non-URC enabled targets via the Universal Control Hub ........ 18
      4.3.3 URC-HTTP Protocol ....................................................................................... 20
   4.4 Updates of the URC Architecture .......................................................................... 21

5  MyUI ............................................................................................................................ 22

   5.1 Aim ........................................................................................................................... 22
   5.2 Background ............................................................................................................... 22
      5.2.1 Adaption mechanism ....................................................................................... 22
      5.2.2 MyUI components ........................................................................................... 24
   5.3 Original architecture ................................................................................................. 25
   5.4 Updates to the MyUI Architecture ........................................................................... 26

6  Integration of AsTeRICS, URC and MyUI platforms .................................................28

   6.1 The Adopted Integration Schema ........................................................................... 30
   6.2 RESTful based communication between the AsTeRICS ARE and the other platforms 31
   6.3 Integration of ARE and UCH (Cross Component Communication) ....................... 33

Ecosystem infrastructure for smart and personalised inclusion and PROSPERITY for ALL stakeholders
www.prosperity4all.eu
6.4 Integration of MyUI and UCH .............................................................. 34
6.5 Integration of MyUI and ARE ............................................................. 35
6.6 Communication Routes between AsTeRICS ARE, MyUI and URC ............ 37
   6.6.1 ARE → MyUI → URC ................................................................. 37
   6.6.2 MyUI → ARE → URC ................................................................. 38
6.7 Setting up the Integrated Runtime Environment ...................................... 38
7   Use Case Scenario: Assisted Living ......................................................... 39
7.1 Personas ............................................................................................. 39
7.2 Environment/Available Technology ...................................................... 40
7.3 Personalized Technology Control ........................................................ 40
   Room parameter control .................................................................... 41
7.4 Enabling the Scenario .......................................................................... 41
7.5 Scenario Description ........................................................................... 42
7.6 Components, Tasks and Roles ............................................................. 43
   7.6.1 ARE → MyUI → URC ................................................................. 43
   7.6.2 MyUI → ARE → URC ................................................................. 46
7.7 Results ............................................................................................... 48
8   References ........................................................................................... 49

**List of Figures**

- Figure 1: Prosperity4All Architecture Schema: links from WP203.3 to other WPs ..... 6
- Figure 2: The original AsTeRICS architecture .............................................. 10
- Figure 3: An OSGi-oriented overview of the ARE architecture ....................... 12
- Figure 4: The REST enabled ARE architecture ............................................. 14
- Figure 5: “ARE Status Service” and REST Services Architecture .................... 15
- Figure 6: The Universal Control Hub (UCH) Architecture ............................ 20
- Figure 7: Overview on MyUI approach to adaptive User Interfaces ............... 23
- Figure 8: Schematic diagram of the main components of the MyUI infrastructure 24
List of Abbreviations

AAIM Abstract Application Interaction Model
AT Assistive Technologies
AsTeRICS Assistive Technology Rapid Integration & Construction Set
ARE AsTeRICS Runtime Environment
ACS AsTeRICS Configuration Suite
OSGi Open Service Gateway initiative
REST REpresentational State Transfer
UCH Universal Control HUB
URC Universal Remote Console
IRE Integrated Runtime Environment

Ecosystem infrastructure for smart and personalised inclusion and PROSPERITY for ALL stakeholders
www.prosperity4all.eu
Executive Summary

This document aims to present the architectural updates, developments and adaptations of AsTeRICS ARE, MyUI and URC towards their integration into the Runtime Environment, which is the main objective of Task 203.3 of the Prosperity4All project.

According to Prosperity4All DoW, the Prosperity4All Runtime Environment will result from the integration of the three major modules:

- AsTeRICS Runtime Environment – ARE: support of diverse Assistive Technologies - AT
- Universal Remote Console (URC): access a variety of products and services via a personal interaction device
- Mainstreaming Accessibility through Synergistic User Modelling and Adaptability - MyUI: Supports run-time adaptations to cover temporal changes of user needs (e.g. during one day, or in the course of ageing or rehabilitation) and to overcome barriers of use directly when they occur

The AsTeRICS ARE, URC and MyUI, as well as their integration into the Prosperity4All Integrated Runtime Environment (IRE) will constitute Development Frameworks within the Prosperity4all Developer Space (DSpace), while parts of them (like libraries and code snippets) will constitute building blocks within the Developer Space to assist developers in building AT, accessible and personalized applications.

The aim of Task 203.3 is to design and develop the integration mechanisms for the three frameworks into one Integrated Runtime Environment, as well as to present useful real life use cases where this Integrated Runtime Environment can be used. However, it should be noted that the development and/or adaptation process of each of the three frameworks into their fully functional prototypes may as well be part(s) of other task(s) within Prosperity4All project. Therefore, the presentation of fully functional frameworks at this stage of development may in some cases not be feasible, but the interaction and integration mechanisms are fully operational.

On the one hand, the IRE enables to lower development costs and simplify the development of assistive solutions, mainly through the capabilities the IRE Development Framework offers to developers, in terms of reusing a variety of components (e.g., ARE bundles, URC targets/controllers, MyUI adaptive interfaces) and functionalities. The Integrated Runtime Environment developed in this work also attempts to address two scientific challenges: firstly, how to offer appropriate interactions to heterogeneous groups of users, and secondly, how to overcome interoperability problems of heterogeneous devices and services.
1 Aim of the document

This document aims to present the architectural updates, developments and adaptations of the following frameworks: the AsTeRICS Runtime Environment (ARE), the Universal Remote Console (URC) and MyUI towards the key target of Task 203.3 which is their integration into an Integrated Runtime Environment in the context of the Prosperity4All project.

The Prosperity4All Integrated Runtime Environment aims at providing accessible user interfaces aligned to the individual needs and preferences of diverse users. For this purpose it includes three major functional modules:

- Support of diverse and individually tailored AT provided by the AsTeRICS runtime environment
- Access to a variety of products and services via user interface sockets as defined by the Universal Remote Console (URC)
- Serving flexible, runtime adaptive user interfaces that match dynamic personal, technical and environmental requirements using MyUI

The Prosperity4All Integrated Runtime Environment will be available as a Development Framework through the Prosperity4All Developers Space (WP201), which provides a means for developers to browse, search, learn about, and use accessible components, tools, parts, and building blocks. Moreover, the three functional modules AsTeRICS ARE, URC and MyUI will also be available individually in DSpace as Development Frameworks. In this manner, developers will be able not only to use the integrated Prosperity4All Runtime Environment as a Development Framework for developing rich applications that utilize both AT and accessible user interfaces to communicate with remote sensors, but also to use separately each one of the three functional modules for their applications according to the actual application requirements.

In addition, S/W and H/W parts of AsTeRICS ARE, URC and MyUI such as libraries (e.g. the UCH Java library - see section 6.3) will constitute building blocks within the Developer Space to assist developers in building AT and personalized applications.

The Prosperity4All Runtime Environment will be also accessible via the Nexus framework (WP201) which supports the integration of heterogeneous components and building blocks to enable easier development of accessible applications and assistive technologies from a variety of existing, reusable components/parts (offered in the Developer Space). An initial development and testing of the interconnection of Nexus with the AsTeRICS platform (Facetracker) has already been conducted in [1]. For more information about the Prosperity4All Developers Space and the Nexus framework the interested reader may refer to deliverable D201.1 “Specification of Architecture, Security, Payment Infrastructure, and Developer Space”.

Ecosystem infrastructure for smart and personalised inclusion and PROSPERITY for ALL stakeholders

www.prosperity4all.eu
This work has also extensively used and was also driven by the work conducted under WP103, and more specifically by D103.1. Deliverable D103.1 serves as the “user experience architecture” document for Prosperity4All, providing specifications and guidelines such as the Design Kit which we have used and applied to ensure that our developments indeed meet the needs of potential users. Moreover, we have used the Design Kit to design and develop the Assisted Living use case scenario described in section 7.

Just like the Global Public Inclusive Infrastructure (GPII) as a whole, the Integrated Runtime Environment created in Prosperity4All should not be a monolithic, self-contained system that includes all capabilities of the previously independent systems. Instead of aiming at such a tight technical integration of the functional modules, the focus of the work on the Prosperity4All Integrated Runtime Environment described in this deliverable has been set to build interfaces and components that make integration easy. Making these components available to implementers via the Prosperity4All Developer Space enables them to flexibly build software systems using different ways of integration based on the specific requirements and technical capabilities.

Based on this concept, the three systems considered in this work (AsTeRICS ARE, URC and MyUI) are loosely coupled, so as to build an Integrated Runtime Environment that is both able to support as a development framework the implementation of AT solutions, while also being open to future extensions by other, complementary systems and functional modules.

In Section 2 titled “Contribution to the global architecture” we talk about how T203.3 links to other WPs, their relations and its contributions to the overall Prosperity4All architecture.

Section 3 describes the architectural updates and developments to the AsTeRICS framework and more specifically to its Runtime Environment in the context of the Prosperity4All project. According to Prosperity4All DoW, the ARE updates concern:

i. adapting the AsTeRICS Runtime Environment to major platforms and
ii. integration of Prosperity4All modules (ARE, MyUI and UCH) for data and event exchange with other components and user interfaces

Point i. has been addressed in [2] and supported by the developments in this deliverable, enabling thus cross-platform support for the ARE for the Linux and RaspberryPI platforms. This report is mostly focused on point ii. integration of Prosperity4All modules, ARE, MyUI and UCH for data and event exchange, including not only the architectural and code updates towards this integration, but also the final use case scenarios and demos developed.

In many cases, references will be made to the ACS (AsTeRICS Configuration Suite), which represents the frontend designer toolkit of the AsTeRICS framework and communicates with the ARE to enable assistive models (i.e., applications) deployment and execution. The AsTeRICS updates and developments in this document concern the introduction of REST functionality in the ARE architecture as a means of communication of remote (and local)
modules with the ARE for the exchange of information, data and resources. Although software updates to the ACS will be needed for the accommodation of the above functionality (introducing the Web ACS), examining any ACS related software updates goes beyond the scope of this deliverable.

Section 4 describes the updates of the Universal Remote Console (URC) and the closely related Universal Control Hub (UCH). The latter is the runtime environment for different targets that shall be controlled by any personalized User interface (represented by the two other technologies). The ARE and MyUI are connected to the UCH via the URC-HTTP protocol. A major step was the development of a new client side Java library that is needed to connect the ARE to the UCH.

In the context of the Prosperity4all project a new version of the UCH with an embedded HTTP server was developed in order to simplify the installation and startup process, as well as the interaction with the other components. Finally, a number of modules were developed, in order to connect the Universal Control Hub to the devices and services that were needed to realize the use case scenario described in section 7.

Section 5 describes the architectural updates and developments of the MyUI Runtime environment in the context of the Prosperity4All project and in association to the integration work performed as part of this deliverable.

Section 6 is about the key objective of this task and deliverable, which is the integration of AsTeRICS ARE, MyUI and URC platforms. The integration relies mainly on HTTP based communication between the three platforms, enabling in this way remote access to their services and functionality, as well as platform management. In specific, we define a RESTful based communication between the AsTeRICS runtime environment and MyUI, while the communication between AsTeRICS runtime and URC, as well as the communication between MyUI and URC are based on the URC-HTTP protocol, which is not yet restful at this stage of developments. Key architectural figures in this section are Figure 11 and Figure 14 (see sections 6 and 6.2).

Section 7 describes the use case scenario that is implemented and realized by using the Integrated Runtime Environment prototype. The use case scenario aims to examine how the integration of the three modules can indeed be used effectively in real life situations regarding accessibility, AT and smart home.

This deliverable includes three ANNEXES.

- ANNEX I describes the REST API of the AsTeRICS ARE (see section 3.4) – the document can serve as a developers’ manual.
- ANNEX II describes the UCH java library (see section 6.3) – the document can serve as a developers’ manual.
• ANNEX III includes installation instructions for the Integrated Runtime Environment (see section 6.7).

All source code and binary files developed under T203.3 can be found in Github: https://github.com/mariokom/RuntimeEnvironment.
2 Contribution to the global architecture

The Prosperity4All Integrated Runtime Environment serves as a Development Framework within SP2 under WP203. In Prosperity4All Architecture Schema (Figure 1), which depicts the different WPs in the Prosperity4All project and their associations, one can observe that WP203 is directly linked with WP201 and WP202, while it has connections with WP301 and WP401 as well.

As already mentioned, the work under T203.3 has extensively utilized the work conducted under WP103 in D103.1 to ensure that our developments meet the needs of potential users (the Design Kit was used to design and develop the Assisted Living use case scenario described in section 7).

The work conducted in T203.3 has been greatly inspired and guided by WP201.1 “Infrastructure Architecture”. More to the point, in WP201 the main Prosperity4All technical architecture was defined, which the work in T203.3 has utilized in terms of the Developer Space Architecture and the Nexus framework specification. The Integrated Runtime Environment meets the Developer Space requirements as a Development Framework, and as such will be indexed within DSpace.
Regarding WP202, part of the work conducted in T203.3, and in particular the UCH components that were developed and/or updated are based on the target adapter framework developed in Task 202.4 of WP202.

The work conducted in this deliverable will be utilized by T202.5 “Real-Time User Monitoring Modules”. The aim is to provide interfaces that facilitate code reuse for user monitoring (e.g. sensor plug-ins) in different assistive applications on different platforms. This Task will especially utilize the OSGi-based (AsTeRICS ARE) Runtime Middleware Environment to enable reduction of development complexity and code reusability in the form of pluggable modules that can be directly integrated into new AT applications (you may read more about code reusability of the Integrated Runtime Environment in section 6.6.2).

Furthermore, the work conducted in T202.6 in terms of the Self-Adaptive UI Modules also affects greatly the work conducted in T203.3 regarding MyUI developments. More to the point, T202.6 on Self-Adaptive UI Modules requires that self-adaptive components from approaches like MyUI and URC are adapted so that to provide runtime adaptations with modern standard frameworks without requiring specific infrastructures, relying on the newest web technologies such as HTML5 and CSS3. T202.6 mainly concerns the architecture and implementation technologies of MyUI which needs to be re-implemented to meet the aforementioned requirements. T202.6 is planned to be completed after the final submission of the present deliverable.

Part of the work conducted in this deliverable, the ARE REST API and the URC Java Library, have been tested under WP401 “Evaluation Framework and Technical Validation”, T401.3.

In addition, the REST-API of the AsTeRICS Runtime Environment that was developed in the context of this work was fully integrated and used within Task T301.1 “Learning and training software applications”, while T301.4 “Pluggable user interfaces for home appliances, home entertainment and home services” plans to utilize the URC sockets.
3 The AsTeRICS Framework Architecture

3.1 Aim
This section describes the architectural updates and software developments to the AsTeRICS framework in the context of the Prosperity4All project. As aforementioned due to the close relation and interaction of the ARE and ACS submodules the architectural changes to the ARE are presented in the general context of the AsTeRICS framework. In specific though, the updates and developments described in this section concern the introduction of REST functionality in the ARE architecture as a means of communication with the ARE and exchange of information, data and resources.

3.2 Background
AsTeRICS – the Assistive Technology Rapid Integration and Construction Set [3] – is an open framework for the development of Assistive Technologies, with the main focus on novel, affordable and flexible AT-solutions. A plethora of sensor, processing and actuator plugins provides a powerful, AT-centred infrastructure, which can be used to control and use ambient assistive services by means of desired sensor combinations – without programming [4]. Interested 3rd parties like research institutions or companies in the field of AT can use the framework to integrate their products into the existing AT-landscape [4].

The AsTeRICS Runtime environment (ARE) is an OSGi-based middleware [3, 5, 6], which allows software plugins to run in parallel. The plugins/modules usually represent a sensor or an actuator and are implemented as independent OSGi bundles. The runtime environment identifies AsTeRICS plugins from other OSGi bundles based on metadata defined inside the plugins. The ARE expects from plugin-developers to define the structure of their plugins (properties, inputs, outputs and event ports) in XML files. Based on these XML files, the middleware constructs a runtime representation of each installed AsTeRICS plugin. Furthermore, the ARE expects a runtime model (system model) that usually comes from the AsTeRICS Configuration Suite (ACS). The ACS is currently running on a Windows Personal Computer (.Net 4.0 required) and mainly used to graphically design the layout of the system as a network of interconnected components. The system model is another XML file that defines the components participating in a specific application, connections between them, events and other properties. Based on this file, ARE knows which plugins to activate and how to define the data flow between them. Since the system model represents the main communication means between the ACS and ARE, it is expected to be a serializable object, easy to transfer and translate. ARE and ACS communicate through an appropriate TCP/IP-based communication protocol named ASAPI.
The AsTeRICS Application Programming Interface (ASAPI) is a proprietary interface/protocol designed, implemented and used for communication between the ARE and external clients such as the ACS. In principle, ASAPI is a “service” that is provided by the ARE and can be consumed by different clients deployed on the same (as the ARE) or remote devices.

In the context of the Prosperity4All project, the aim is to update the communication medium between the ARE and the ACS, in order to eliminate limitations of the ASAPI protocol and allow a pure and unobstructed network based communication. More to the point, Prosperity4All aims to realize the accessibility for all concept, and towards this goal, we plan to utilize the power of web services via the implementation of a Representational State Transfer - REST enabled ARE. REST is a software architecture style that allows for creating scalable web services. We have selected this solution so that the ARE will be easily accessible and manageable via the network, not only by the ACS, but by any program (e.g., MyUI, URC, GPII) that needs to communicate with it. Of course, the original ACS will be updated as well (WebACS – as part of T203.2), in order to be able to communicate with the new version of the ARE.

Other reasons for replacing the ASAPI protocol with REST services is that the ASAPI is a proprietary/custom protocol in contrast to the widely used REST architecture style that uses the well-known HTTP protocol. Moreover, it is easier and straightforward to control the ARE by simply invoking a certain resource via a URL, even from another program than the WebACS, which is not the case by using the ASAPI protocol. Finally, although the ASAPI protocol offers remote connectivity since its TCP/IP based, an issue may arise when accessing the ARE remotely in case the ports and TCP/IP traffic are blocked by the firewall rules. This does not apply for REST HTTP that provides unobstructed remote access, while mechanisms are available for ensuring security.

Another important development activity was the implementation of the ARE Status Service, which is based on the notion of a publish-subscribe mechanism that any client software can exploit in order to subscribe and receive notifications for ARE events that are of interest (e.g. a different model has been deployed directly using the ARE). In specific, this service aims to provide a way for other programs such as the Web ACS, URC and MyUI to be informed and updated about the ARE status by registering to it as clients.

In this section we will focus on the updated ARE architecture and the new developments, so as to provide an abstract overview of the component’s workflow that utilizes in a great extend the OSGi technology, as well as concentrate on the software developments that were performed as part of this work.
3.3 The Original AsTeRICS Architecture

The original AsTeRICS architecture presented in Figure 2 consists of two components that communicate with each other to achieve the desired result:

(1) AsTeRICS Runtime Environment - ARE
The ARE is an OSGi-based middleware that allows software plugins to run in parallel. A plugin usually represents a sensor, a processor or an actuator and is implemented as an independent OSGi bundle. The ARE in order to function expects a runtime model (system model) which usually comes from the AsTeRICS Configuration Suite (ACS).

(2) AsTeRICS Configuration Suite - ACS
The ACS is running on a Windows Personal Computer and mainly used to graphically design the layout of the system as a network of interconnected components in a model. The system model is an XML file that defines the components participating in a specific application, connections between them, events and other properties.
As mentioned above, the ARE component is based on OSGi. This offers the advantage of parallel execution of the internal ARE components and the ability of importing possible code extensions with ease. Everything is implemented as an OSGi bundle and the bundles are communicating together with virtual ports and channels (also OSGi bundles). From an OSGi perspective, we can categorize the bundles to:

1. **AsTeRICS Model specific bundles**
   These are the model components used to implement the model’s sensors, actuators and processors. Every component defines additional information about its nature in an XML file called bundle descriptor.

2. **Ordinary OSGi bundles**
   In this category there exist OSGi bundles that either manage the AsTeRICS Model specific bundles such as the core functionality of the ARE middleware, or are responsible for the communication either between the AsTeRICS Model specific bundles (virtual ports), or the communication of the ARE with the outside world.

Figure 3 shows an OSGi-oriented overview of the ARE architecture.
3.4 Architectural Updates and Developments to the AsTeRICS Architecture within Prosperity4All: Rest Services

In the context of the Prosperity4All project, the communication medium of applications to the ARE (and between the ARE and the ACS) is being refined. The aim is to eliminate any communication limitations of the ASAPI protocol and allow a pure and unobstructed network-based communication facilitating the “accessibility for all” concept of the Prosperity4All project. The aim is to implement a REST enabled ARE that will be easily accessible and manageable via the “open” network (see Figure 4), not only by the ACS, but by any other program as well that needs to communicate with it, e.g., the URC and MyUI. The communication with the ARE will be thus exclusively over “open” network through the help of the HTTP protocol.

One of the main requirements for refining the communication medium between the ARE and the outside world was to provide the capability to integrate the very large set of assistive functionalities offered by the implemented AsTeRICS model components (i.e., OSGi
bundles) into existing applications. The target was to still facilitate the design and development of assistive applications through the use of the ACS, but at the same time enable integration of assistive functionalities into existing software applications implemented in different technologies. In specific, the ASAPI substitution offers platform and language independence. This means that a developer is able to reuse and integrate assistive functions into existing software applications, without any concerns about the language and/or the platform used to implement and deploy the applications. In this way, the development effort is reduced and the development process is simplified since these assistive functionalities have already been implemented in their majority, reducing also the costs of integration.

For more information on the architectural updates of the ARE, as well as a number of use cases implemented based on these updates, the reader is referred to the following scientific paper published based on this work [7].

**ARE REST Service**

The HTTP communication protocol offered via the REST architecture style will be exploited, but it needs to be mentioned that this is implemented within an OSGi container. To “talk” HTTP we have used an HTTP server (Grizzly server) implemented within an OSGi bundle [8]. This bundle uses an embedded OSGi service, called HTTP Service. The HTTP server was intentionally implemented as an internal OSGi bundle/component so as to be able to interact with any other component in the ARE, while at the same time communicate over the network.

Regarding the ACS, in order to be able to communicate with the REST enabled ARE, it is currently being updated as part of T203.2 (Assistive Technologies (AT) Configuration Environment) in Prosperity4All, so as to include the corresponding functionality. In fact, the newly implemented Web-ACS will define a REST Client that issues specific requests to the ARE’s REST server using the newly developed REST API.
ARE Status Service

Apart from the Restful communication, the new ARE will provide an additional service called “ARE Status Service” by which any interested platform/system/program will be able to automatically be informed about the current situation and status of the ARE. More particularly, Server-Sent Events (SSE) Support [9] that adheres to the HTTP protocol will be used, in order to allow communication only via a single protocol and enable the ARE to asynchronously push “State” related information to any client registered to a specific service. This follows a publish-subscribe mechanism that any client software can exploit in order to subscribe and receive notifications for ARE events that are of interest (e.g. a different model has been deployed directly using the ARE). Once the connection to the ARE is established by the client, the ARE (playing the role of the server) will push the required information at the appropriate time. This service aims to provide a way for other programs such as the Web ACS, URC and MyUI to be informed and updated about the ARE status by registering to it as clients. Figure 5 presents the refined architecture of the ARE and in specific provides details on the new REST architecture style and the HTTP communication.
model of the ARE. Note that the client(s) needs to subscribe using the SSE mechanism, so as to enable the server to keep a connection alive and push messages on the required events.

The ARE REST API was developed by UCY under T203.3 (D203.1) and can be found in ANNEX I of this document which can serve as a developers manual.
4 Universal Remote Console (URC)

4.1 Aim

This section describes the architectural updates and developments to the Universal Remote Console [10] in the context of the Prosperity4All project. Due to the close relationship between the URC-Standard and the Universal Control Hub (UCH) [11] which is its implementation, they are both considered in this section. The described modifications relate to an appropriate integration of AsTeRICS, MyUI and URC. Thereby, one major task was to implement a Java based, client side library of the URC-http protocol [12, 13] in order to enable communication between ARE and UCH.

Furthermore, a new, standalone version of the UCH was developed. This simplifies the deployment of the UCH. This is due to the fact that no additional installation and configuration of a Tomcat server is needed anymore.

Finally, in order to support the use case scenario described in section 7, a number of UCH modules were developed to connect to the Philips Hue and the electricity outlet Wöhlke Websteckdose.

4.2 Background

The overall purpose of the URC framework (standardized in ISO/IEC 24752) [14] is to provide a mechanism, enabling users to control any target with any controller devices fitting best the user’s needs. Targets can be devices or services, usually such that can be found in the Smart Home or Ambient Assisted Living domain. Controllers can range from User interfaces running on PCs, Smart Phones over traditional Remote Controls to regular or specialized hardware.

In order to realize such a system every target provides an abstract description of its operating interface. In URC terms this is called a User Interface Socket Description - or just Socket Description. Socket Descriptions contain all information about a device’s properties, which can be accessed by a user. Basically, these are variables that can be controlled by the user, commands that can be send to the target, and finally notifications that the device can send to a user interface.

Based on the concept of Socket Descriptions, third party user interface developers can create personalized user interfaces, and publish them via a dedicated Resource Server. At runtime, users can select a user interface fitting best their needs, download it from the Resource Server and virtually plug it into a target. This is why they are referred as pluggable User Interfaces.
As a solution for applying the concept of Sockets to not URC compliant targets the Universal Control Hub was developed. In the backend it has mechanisms to discover different targets from different technologies. As soon as a target is discovered its Socket Description is downloaded from the dedicated Resource Server and exposed to the user. Consequently, the principle of pluggable User Interfaces is again applicable.

To enable the communication between any user interface and the UCH, the URC-HTTP protocol was developed. The UCH reference implementation ships with a Component implementing the server side of the URC-HTTP protocol. All received messages are transformed by the UCH in target specific ones and forwarded to the appropriate target.

4.3 Original architecture

When considering the URC architecture two main cases must be distinguished. The first one is when targets and Controllers are totally compliant to ISO/IEC 24752. Most important is here, that targets expose their Socket Descriptions on their own. If communication between a not standard compliant controller and/or target shall take place case 2) the Universal Control Hub provides a middleware solution to integrate the different communication partners. In this case the UCH is responsible for exposing the targets’ Socket Descriptions.

4.3.1 ISO/IEC 24752 compliant communication

The international standard ISO/IEC 24752 specifies communications between a target that a user wishes to access and operate, and a universal remote console (URC) that presents the user with a remote user interface through which they can discover, select, access and operate the target. The URC is a device or software through which the user accesses the target. If the URC is software, it is typically hosted on the user’s physical device, but a distributed approach is also possible.

Communications between the target and URC take place over a network, the target-URC network. ISO/IEC 24752 does not specify the network protocol to be used, and a target or URC may support any number of appropriate connection protocols. These protocols are used to provide discovery of targets, and to establish and maintain control sessions between URCS and targets. Targets and URCS access the target-URC network through target-URC network links. Targets support discovery by providing essential information in a target description. Among this information is the link to the target’s abstract description of its operating interface – the User Interface Socket Description (see below).

Each target provides a User Interface Socket (or short Socket), or set of Sockets, through which a URC can access part or all of the target’s internal states and provide control inputs to the target. For each Socket, a target provides a dedicated User Interface Socket
Description (or short Socket Description) in a XML-file which describes the Socket in a machine interpretable manner.

Overall, there are three types of socket elements whose state is synchronized between a target and a connected URC and that must be described in a dedicated Socket Description.

- Variables are state variables of the target and can typically be changed by the URC,
- Commands represent function calls on the target that can be invoked by the URC,
- Notifications propagate special target states in which normal operation is suspended on the target and requires a notification or action of the user.

Additionally, the target provides resources that pertain to the user interface of the target, as viewed through that Socket. The Socket Description and resources are used by the URC to find or generate an appropriate user interface, given the functionality of the target, the nature of the URC device, and the user's interaction preferences.

Resources can either be provided by a target itself, by the URC or by a dedicated resource server. Resources can range from different user interfaces to exchangeable parts of a User Interface. Latter could be labels in different languages, as well as additional help items (e.g., in text format or as sign language videos.

Dependent on the implementation of the URC users can either choose their preferred User interface on their own or it is automatically selected by the URC according to some context or personal information. If it is not available on the URC or on the target it is downloaded from the Resource Server. After the selection of a user interface it is virtually plugged into a targets Socket.

Interaction between a target and a URC consists of a discovery phase and an optional control phase. The discovery phase initializes the URC to locate and identify all available targets and their sockets. The control phase is the time period during which a target and a URC initiate, maintain and terminate a control session. A control session is a connection between the URC and a target’s socket for the purpose of the URC controlling a functional unit of the target via the socket. When multiple URCs are connected to the same Socket, each has an independent control session, although target state values may be shared [10].

### 4.3.2 Integration of non-URC enabled targets via the Universal Control Hub

Today, we are encountering a growing number of networked devices and services at private and public places. However, these devices and services are using different networking platforms and technologies. Some of them are already standardized but there are also many proprietary technologies available. It seems to be virtually impossible to get all manufacturers of these devices and services together to agree first to adopt the URC standards, and second to use a specific Target-URC Networking protocol for communication.
between URCs and targets. Instead, a different solution must be sought that can harvest the benefits of the URC standards, and works with existing devices and services available on the market.

The Universal Control Hub (UCH) handles at this point the profiling of the URC standards which implement standard-conforming targets and URCs "in a box", providing connection points to existing targets and controllers that are not URC-compatible. The UCH is a middleware that establishes a control connection between non-compliant controller and non-compliant target devices/services that would otherwise not understand each other. The UCH is designed to be extremely extensible and scalable with regard to diverse targets and controllers. By using the mechanisms as defined in ISO/IEC 24752, it provides an open platform for pluggable user interfaces that are based upon User Interface Sockets.

In the UCH architecture (see Figure 6), any network-enabled device/software can be used as "controller". Also, the "target" is a network-enabled device or service that can be remotely controlled via any networking platform. Note that this is different from a "conforming target", as defined by ISO/IEC 24752-1:2008. The "conforming URC" and "conforming" targets are virtually folded into the UCH and its components.

The UCH discovers a target through a Target Discovery Manager (TDM), and controls it and receives events from it through a Target Adapter (TA). To a controller, the UCH makes itself discoverable through the UIList, an extended UPnP RemoteUI mechanism, as defined in [CEA-2014-A].

The UCH communicates with a controller for the purpose of target control through a User Interface Protocol Module (UIPM). By connecting to controllers and targets through these device-specific and network-specific adapters, the UCH “talks to them” in their native language. Acting as a middleware, the UCH then translates the communication with targets and controllers into the "URC language", exposing the targets and their functions as sockets that the controllers can access. Thus the UCH bridges between non-conforming controllers and non-conforming targets, fulfilling the URC requirements on behalf of them.

There exist many middleware products that bridge between controllers and targets in a similar way. What is unique for the UCH is that it uses a standard approach for the translation between targets and controllers, thus opening a way for third parties to make their pluggable user interfaces and accompanying resources available to the UCH after installation. The UCH constitutes an open platform for control interfaces that uses the mechanisms defined in the URC framework to allow for pluggable user interfaces and supplemental resources dynamically at runtime.

Moreover, the UCH architecture is designed to be extensible for new targets and controllers.
Except for the core UCH (socket layer), its components (blue boxes: UIPMs, TDMs and TAs) are embedded into the UCH through well-defined APIs (red vertical boxes attached to the components). Any party can develop such components, and a UCH can load them at runtime. This mechanism is extremely powerful in conjunction with a resource server (see below). Then new targets and new controllers can be integrated into a local UCH system on the fly, by downloading the appropriate components from the resource server [15].

4.3.3 URC-HTTP Protocol

In the UCH architecture, a UIPM is responsible for presenting one or multiple sockets to the user through a user interface that is rendered on a controller. In some cases, controllers and their software themselves might have knowledge about the URC technology, in particular about sockets and their resources (These controllers are "conformant URCs", as defined by ISO/IEC 24752-1:2008.) They might want to get direct access to a target's Socket and pertaining atomic resources, in order to build a suitable user interface based on the Socket elements and their values.

However, in most cases either the Controller or the target is not compliant to ISO/IEC 24742. Therefore, the URC-HTTP protocol, as defined by [12], facilitates direct access for a controller to the sockets running in a UCH. This protocol defines HTTP-based messaging for a controller accessing the sockets on a UCH and its functions. An optional TCP/IP-based update channel is also defined to inform controllers about status updates of connected targets.

The UCH open-source reference implementation by the Trace Center includes a UIPM for the URC-HTTP Protocol.

The URC-HTTP protocol is designed to be usable by clients running in Web browsers, such as scripts and plugins. For example, with the Webclient JavaScript Library [16] that is available via the openurc.org website.
4.4 Updates of the URC Architecture

In order to integrate the UCH with the AsTeRICS runtime (ARE) and the MyUI framework, several updates are mandatory.

In order to enable a direct communication between the AsTeRICS runtime, which is OSGi-based, and the Universal Control Hub it was necessary to transform the Java Script based Webclient library [16] to a Java implementation. This work was done by UCY in cooperation with HDM.

Furthermore, a new, standalone version of the UCH was developed – the UCH 4.0 – by HDM. The new standalone version ships with its own embedded tomcat server and does not rely anymore on a separately installed servlet container. This simplifies heavily the deployment of the UCH. This is due to the fact that all required configuration is already set in the predefined UCH package and thus the installation and configuration of the separate servlet container has become obsolete.

The development of the new UCH standalone version was necessary to enable a future integration with the GPII infrastructure and also the MYUI and ARE (AsTeRICS framework). When logging in to a system with a GPII installation not all GPII supported components are available. Hence it must be possible to easily deploy further components at runtime. The new UCH version 4.0 meets this requirement. It can be downloaded at runtime by any third party and the executable jar file can be started without any further installation or configuration (for installation instructions please refer to ANNEX III of this document). Finally, in order to realize some illustrative use cases, the following target discovery modules and target adapters were developed and/or updated by HDM:

- Wöhlke Websteckdose: The Wöhlke Websteckdose is a multi-electricity outlet that can be located in the network. It is very generic in the sense that it can be used to remotely switch on or of any electric device (e.g. fan heater). Modules to discover, configure and control a Wöhlke Websteckdose via the UCH were already available, however, some minor bug fixes were required.
- Philips Hue: The Philips Hue is an illumination system that can be used to create spatial ambiances. During the project, a new target discovery module, a new configuration manager and a new target adapter were developed.
- VLC Player: The VLC Media Player is a well-known Media system. During the project, a new target discovery module, configuration manager and target adapter were developed in order to control a VLC media centre that is located in the network.

All UCH components that were developed/updated are based on the target adapter framework developed in Task 202.4.
5  MyUI

5.1  Aim

This section describes the architectural updates and developments to MyUI [17] in the context of the Prosperity4All project.

5.2  Background

MyUI provides an environment to render and adapt a user interface to the user context during runtime. All knowledge on various user interface design solutions and adaptations is contained in design patterns. MyUI provides also an abstract format to define the interaction possibilities of a user with the application called AAIM (Abstract Application Interaction Model), which is based on UML2 State Machine Diagrams. Instead of developing many variants of the user interface in order to meet the individual needs of the diverse end users, the application developers can focus on implementing the application logic. The knowledge about the user interface adaption lies in the pattern repository.

5.2.1  Adaption mechanism

User interface adaptations in MyUI follow a three-stage process (see Figure 7) including the User Interface Parameterization, the User Interface Preparation as well as the User Interface Generation and Adaptation. A summary of each stage is described in the following sections, for the full description please refer to the MyUI Developer Documentation [18]. After significant user profile updates the process of user interface adaptation starts again.
User Interface Parameterization

In a first step, general user interface characteristics are set in order to adapt the overall user interface appearance and interaction mechanisms to a specific user, specific I/O devices and a desired corporate or product identity. These general user interface settings are done via global variables and stored in the MyUI User Interface Profile. A MyUI User Interface Profile is initiated at the beginning of a new interaction session with a MyUI application. A repeated User Interface Parameterization cycle (i.e. user interface profile update) is triggered when newly available information about the user and his/her context leads to a significant change in the user profile.

User Interface Preparation

On the basis of the current interaction situation and relevant variables of the user interface profile the most suitable display and control elements are selected from a repository of user interface components and elements. This second step makes sure that the selected user interface building blocks correspond to individual user needs and device-related requirements. It is triggered whenever a new state of the application is entered (i.e. a new interaction situation is active) and after each user interface profile update. The new selection of display and control elements is compared with the currently displayed selection. If a difference between the current and the new components is identified, adaptation patterns to switch from the current to the new user interface are selected in the next step.
User Interface Generation and Adaptation

The selected display and control elements are composed and rendered to an individualized user interface. This rendering process takes place at the beginning of a new interaction session with a MyUI application in order to generate the interface. The same process is also triggered when in the user interface preparation process (step 2) a new set of display and control elements is selected. In this case, also the selected adaptation patterns are executed to switch from the current to the new user interface.

5.2.2 MyUI components

![Figure 8: Schematic diagram of the main components of the MyUI infrastructure]

MyUI Runtime

The MyUI Adaptation Engine generates and adapts the user interface for a specific end user in a specific context. The Adaptation Engine composes individualized user interfaces from reusable software components during runtime. Even if the current focus is accessibility the MyUI Runtime can be used for all kinds of adaptations so all users can profit from a tailored user interface. The implementation is based on CakePHP, MySQL (Server, other DBs possible) and jQuery (Client) and runs on every PHP-enabled webserver. It does not have an interface on its own but generates all of the applications adaptive interfaces.
MyUI Context Manager

The MyUI User and Context Management Infrastructure detects and interprets relevant characteristics of the end user and her/his environment which bring about special accessibility requirements. This information is stored and updated in a user profile which serves as basis for generating an individualized user interface. If context conditions change, the MyUI user interface is adapted immediately e.g. because the attention threshold is lower in crowded places the system will show less elements per screen. It is implemented using C# and SQLite and should run on every platform with a .NET runtime environment available. Besides the console-based server there is a client application with a graphical user interface for management of the context manager server.

MyUI Development Toolkit

The MyUI Development Toolkit provides intuitive and efficient tools and mechanisms to user interface developers and designers for the creation of MyUI applications with the MyUI adaptive user interfaces infrastructure. By the use of an additional abstraction layer (an abstract model of the UI) developers do not need to implement concrete/final user interface solutions, rather they define and model the interaction possibilities the users have when using the system. The tool is implemented as a plugin for the Eclipse IDE in Java. It depends on the Eclipse PDT (PHP Development Tools). For the preview feature the development toolkit requires a local installation of both the MyUI Runtime and Context Manager.

5.3 Original architecture

The implementation of the MyUI Runtime Environment is built up as a CakePHP plugin. The CakePHP plugin approach helps developers to easily extend their applications with reusable software components (plugins). A CakePHP plugin contains its own models, views and controllers as well as its own webroot folder where images, css files, javascript libraries etc. are located. Each CakePHP plugin is saved in the plugins folder of the main CakePHP application. The general folder structure of a CakePHP plugin looks like follows:

```
plugin_name/
    controllers/
    components/
    models/
    views/
    elements/
    helpers/
    webroot/
```

MyUI is also following this plugin structure. The controller package contains the controller classes such as the profiles_controller and applications_controller together with the
individualization_patterns component. Classes representing real objects are stored in the models folder. Unlike normal CakePHP plugins the MyUI Core plugin does not contain fully implemented views in its views directory. Instead it makes heavy use of the CakePHP elements mechanism to generate the user interface structures out of interaction patterns and user interface elements at runtime. Besides that, plugins can integrate own external content or libraries such as images, javascript files, css files etc in the webroot folder. The MyUI Core plugin, the adaptation engine itself and the adaptation handlers are located here. The complete structure of the plugin is shown in Figure 9.

Figure 9: Package diagram of the MyUI cakePHP solution

5.4 Updates to the MyUI Architecture

Rather than changing or adding parts of the existing implementation, the MyUI runtime environment has been entirely reimplemented in Prosperity4All. Due to the purely browser-based implementation using JavaScript and HTML5 it does not depend on any server-side technology. Along with that the runtime implementation has been structured to allow for modular extensions: Different user interface solutions can be represented by different situation factories, additional data sources and functionality can be integrated using additional services. From an implementers view, the AAIM that is represented by a JSON data structure has been upgraded to be the central point of configuration for a MyUI-based application using existing sets of user interface components and backend services.
Figure 10: Implementation structure of the MyUI Runtime

The capabilities of adaptive user interface components will be built upon the self-adaptive UI modules currently being created in T202.6 and are therefore not yet covered by the MyUI runtime implementation described here. Due to its modular structure till the integration of these, other web frameworks may be used for the implementation of user interface components. For example, the current demonstrator is implemented using the Polymer Framework\(^1\).

\(^1\)https://www.polymer-project.org/
6 Integration of AsTeRICS, URC and MyUI platforms

Through the Integrated Runtime Environment this work also attempts to address two scientific challenges: firstly, how to offer appropriate interactions to heterogeneous groups of users, and secondly, how to overcome interoperability problems of heterogeneous devices and services. We address the above mentioned challenges by enabling the integration of Adaptive User Interfaces and Middlewares, aiming to support real life Assistive Technologies and Internet of Things scenarios. When developing this prototype, a key requirement was to address adaptability, not only at the graphical level, but also to support people with special motoric needs in Assistive Technologies scenarios. For more information on the above scientific contributions the reader is referred to the following scientific paper published based on this work [19].

The integration of the three main platforms, AsTeRICS, URC and MyUI rely on a RESTful based communication. The aim is to accomplish an efficient, effective and straightforward way of communication and exchange of data, information and resources between these components, while at the same time maintaining the ability for other programs to be able to potentially communicate with the new system of platforms. In fact, the REST specification is nowadays the norm for developers since it is widely and heavily used, and thus makes it easy and straightforward for developers to utilise the Integrated Runtime Environment. Figure 11 depicts how the three main platforms can be utilized in the context of the GPII. GPII (C4All Matchmaker) initiates the AsTeRICS runtime (ARE) in order to execute/manage a model. The GPII is able to identify the user via NFC communication (see Figure 12) and based on the needed solution that best accommodates the needs of the user, starts the appropriate AsTeRICS model on the ARE. Currently the model is started by initiating a local command from the GPII to the ARE, since both software platforms are running on the same machine. Our aim is to achieve a RESTful based communication between the GPII and the ARE so that ARE management can be done remotely by the GPII. To achieve this, a RESTful client implementation on the GPII is needed, which is considered as part of future work.

For the integration of MyUI with the GPII, currently both need to run on the same machine providing thus the possibility of using the existing JSONSettingsHandler to write a file containing the user’s preferences into a subdirectory of the MyUI server directory. In this way, MyUI application is able to load this file when being started in a web browser and hand it over to the adaptation mechanism provided by the adaptive web components implemented in another Task, T202.6.

Additionally, as shown in Figure 11, AsTeRICS ARE communicates with URC by including the URC Remote Component as an AsTeRICS model component within the ARE.
Ecosystem infrastructure for smart and personalised inclusion and PROSPERITY for ALL stakeholders
www.prosperity4all.eu

Figure 11: The 3 main platforms under the GPII

Figure 12: NFC Technology is used for user identification
6.1 The Adopted Integration Schema

After examining a number of integration possibilities between the three platforms [20], we have resulted in the interaction schema mentioned below as the most appropriate one that is able to offer good communication between the platforms, remote platform management and dynamic exchange of information. Figure 13 shows the final integration schema.

![Final integration Schema](image)

In this integration/interaction schema, ARE and MyUI communicate with the URC using the URC-HTTP protocol. This protocol is offered in JavaScript. UCY implemented the URC-HTTP protocol in Java for the needs of the ARE-URC communication. To achieve URC remote sensor/actuator management/communication from within the ARE, a URC remote controller (namely *URC actor component*) was developed as an AsTeRICS model component that runs within the AsTeRICS model that is able to communicate directly with the URC remote sensor/actuator via the URC-HTTP protocol (see Figure 14). The result is enabling access to remote URC sensors/actuators and other H/W from within an AsTeRICS model, meaning that remote URC sensor information (e.g. the temperature of a remote thermostat) or actuator process (e.g. switch on a heater) can be initiated from AsTeRICS model components and reused as well throughout a number of AsTeRICS models.

MyUI Adaptive User Interface also communicates with the URC via the URC-HTTP protocol.

The communication between ARE and MyUI is bi-directional and is conducted in a different manner for each direction. The ARE → MyUI communication is loose and stretches over the operating system. The AsTeRICS model running in the ARE is emulating default operating system events like mouse or keyboard interaction to control the MyUI Adaptive User Interface running on the same machine. What is achieved is to enable the usage of MyUI Adaptive User Interface to accommodate the preferences of a user in terms of the user interface (e.g. large fonts and medium images), while at the same time enabling AsTeRICS
smart accessibility and AT functionality (e.g. handling the computer mouse via head movement through a camera). A RESTful based communication between ARE and MyUI would be feasible as well, however, such a communication would include real-time polling of information from the ARE, significantly affecting the network. More appropriate solutions would include a persistent HTTP connection between the two platforms, which is out of the scope of this work and is considered as part of future work.

MyUI → ARE communication is conducted via an appropriate MyUI Adaptive User Interface that is able to manage the AsTeRICS ARE through a RESTful communication. Via this interface, MyUI Adaptive User Interface is able to initiate commands to the ARE, as well as receive relevant updates of the ARE’s application state and runtime information.

6.2 RESTful based communication between the AsTeRICS ARE and the other platforms

Figure 14 shows the RESTful based communication between the AsTeRICS runtime and the two other platforms, mainly URC and MyUI. Any program will be able to use the REST interface to communicate with the ARE. In the same figure the Server-Sent Events (SSE) Support can also be depicted as the “Push state Messages”.

Prior to this work, the communication means from the AsTeRICS ACS to the AsTeRICS ARE and vice versa was conducted via the ASAPI protocol, which was replaced in the context of this work by a more easy to implement and adopt, as well as more flexible RESTful interface. The RESTful implementation is developed on the server side (ARE). The REST functionality allows communication of the outside world with the ARE, providing thus the ability to manage the ARE by specifying a group of actions that ARE can perform during runtime, such as to start a model, to stop a model, etc.

However, the actual model that is being run cannot be adapted during execution. The model and all its components (sensors, processors, actuators, ports, etc.) have to be predefined at development/design time. This adds complexity when one (or more) of the components of the model being run in the ARE runtime needs to be an outside component, i.e. a component of another platform. An example is depicted in Figures 11 and 14, where a thermostat component handled by the URC platform needs to be included within an AsTeRICS model that is running on the ARE. This can be accomplished only by predefining at design and development time the thermostat component and its connections to other components in the AsTeRICS model. Let us name this AsTeRICS thermostat component as “Ather” component. This component, while running on the ARE, must be able to communicate with the UCH platform in order to manage the actual thermostat sensor. In Figure 14 this communication is shown as “Cross Component Communication”, since the Ather component (which is in this case the URC actor component), while being an AsTeRICS model component, must communicate with the actual thermostat sensor.
To achieve cross component communication, an AsTeRICS Ather component was developed that is able to communicate via REST through the UCY developed Java based URC-HTTP protocol with URC. This component is able to be used within any AsTeRICS model and can connect to other AsTeRICS model components for inner model communication, while at the same time it includes the appropriate REST functionality to communicate with URC. UCY has specified generic guidelines as to how a developer can design and implement a URC actor component in ANNEX II, but the development will be left upon the developer.
In the following subsections the details of the integration mechanisms between the three platforms are presented covering both the theoretical and the technical aspects.

6.3 Integration of ARE and UCH (Cross Component Communication)

The ARE-URC integration takes part on component level, meaning that when a scenario defines a URC sensor to be managed, a corresponding component should exist in the ARE. This ARE component will be capable to monitor/manage/coordinate the URC remote sensor over the network.

To communicate with the URC server and components, a Java library that implements the URC-HTTP protocol was a necessity. Due to the fact that such a library did not exist, UCY took over the responsibility to create one that provides a complete set of functions which will cover all possible scenarios demanding ARE-UCH integration.

Among other capabilities, the UCH-communication Java library (UCH jClient) allows ARE components to:

1) Discover URC server’s location in the local network with a zero-configuration networking (zeroconf) implementation (see Figure 15). With JmDNS [21], the UCH jClient is searching in the local network for the UCH zeroconf service. If found, the UCH jClient is able to retrieve the IP address of the machine that hosts the server and proceed with the communication procedure.

![Figure 15: Local network example](image)

2) Make out-of-session URC-HTTP requests to discover which URC components are running on the UCH server and retrieve all the available information for each one of them (static information). Using the UCH-communicator object, any Java client can query the UCH server repository and obtain the UIlist, an xml formatted list with the running URC components descriptions. Every element of this list, contains information needed for the communication, such as the component id and its Socket Description. Below a snippet of code is depicted that offers this capability.
3) When needed, the UCH-communicator object can create session objects which are following the URC-HTTP protocol as described in the URC documentation [12]. Every session object can send in-session requests to monitor/change the state of the URC components using a unique session id. The following is an example of how to create a session object:

```java
String sessionId = uchCommunicator.createSession_getRequest(httpUrcURI);
UchSession session = uchCommunicator.getSession(sessionId);
```

More information on the UCH Java library with the complete set of functions for ARE-UCH integration can be found in ANNEX II of this document.

6.4 Integration of MyUI and UCH

The MyUI runtime contains two concepts for binding user interfaces to application data and functionality. Data Acquisition Functions are used as data sources for information to be presented by the generated user interface. Accordingly, Application Functions represent application specific actions to be triggered by the user interface. In the implementation of the MyUI runtime both types of functions are provided via implementations of the base class AaimService. Therefore, this base class serves as the appropriate integration point for the communication with the UCH.

The services are themselves implemented in JavaScript and executed in the web browser. Hence, the existing URC Webclient JavaScript Library [16] builds the foundation for the implementation of a generic UchService as shown by the class diagram in Figure 16.
An example of a service function that may be used as a *Data Acquisition Function* is presented in Listing 1. The `queryTargets` method uses the function `getAvailableSockets` provided by the URC Webclient library to return a list of all targets provided by the UCH instance. The list may be filtered to only contain targets that match a given socket definition. Another method that can serve as data source is `currentState` which uses the URC `getValue` function to fetch the current values of the socket variables of a specific target.

```javascript
queryTargets(socketName) {
    let allTargets = org_myurc_urchttp_getAvailableSockets();
    if (socketName) {
        return allTargets.filter(function(target) {
            return target.socketName === socketName;
        });
    } else {
        return allTargets;
    }
}
```

**Listing 1: Implementation of the `queryTargets` method in the `UchService`**

*Application Functions* on the other hand typically perform actions or change data. An example for this type of service methods in the `UchService` is `setState` (see Listing 2). It is implemented using a call to the `setValues` function of the URC Webclient.

```javascript
setState(socketName, targetId, values) {
    org_myurc_webclient_init([socketName], 0);
    let elements = [];
    for (let val in values) {
        elements.push({
            elementPath: "/" + val,
            operation: "S",
            value: values[val]
        });
    }
    org_myurc_webclient_setValues(socketName, elements, targetId);
}
```

**Listing 2: Implementation of the `setState` method in the `UchService`**

### 6.5 Integration of MyUI and ARE

To allow asynchronous services, the `AaimService` implementation in the MyUI runtime is based on the concept of promises [22]. Return values of service methods are wrapped into promises while promises returned by service methods are forwarded unchanged. This enables the `AreService` implementation to use the asynchronous ARE REST API (see section 3.4) to provide *Data Acquisition Functions* (see `listStoredModels` in Listing 3 as an example) as well as *Application Functions* like starting and stopping models.
listStoredModels() {
    return new Promise(function(resolve, reject) {
        listStoredModels(function(data, httpStatus) {
            resolve(data);
        }, function(httpStatus, message) {
            reject(new Error(message));
        });
    });
}

Listing 3: Implementation of the listStoredModels method in the AreService

The capability of the MyUI runtime to load AsTeRICS models into the ARE and start their execution via the ARE REST API leads to the possibility of automatically provide AT interaction mechanisms realised as AsTeRICS models and web-based user interface patterns aligned to each other. Designers may define both parts in conjunction and treat them as one interaction solution. Figure 17 sketches the general communication sequence between the AsTeRICS Runtime Environment (ARE), MyUI and the GPII Preferences Store as a source of user preferences.
6.6 Communication Routes between AsTeRICS ARE, MyUI and URC

In scenarios where all three frameworks AsTeRICS ARE, MyUI and URC are included, we can identify two communication routes that can be used.

6.6.1 ARE → MyUI → URC

In this scenario, the communication between the ARE and MyUI is being conducted over the operating system. The AsTeRICS model running in the ARE provides accessibility and AT interfaces to control default operating system events such as mouse or keyboard interaction in order to control the MyUI Adaptive User Interface. The URC-HTTP protocol is being used to enable communication between MyUI runtime and URC. The REST calls must be integrated in the Abstract Application Interaction Models as the final application functions being executed. To simplify the communication between the MyUI runtime and the UCH, the JavaScript Library implementing the client side of the URC-HTTP protocol is used. The Abstract Application Interaction Models must be created at development time by a developer or during runtime by parsing the Socket Descriptions being available in the UCH.
6.6.2 MyUI → ARE → URC

This scenario requires that an appropriate MyUI Adaptive User Interface exists that manages the AsTeRICS ARE through a RESTful based communication. Through this interface, MyUI Adaptive User Interface is able on one hand to initiate REST calls to the ARE to handle models, such as “Start Model”, “Stop Model”, “Pause Model”, “Load Model”, etc., and on the other hand to receive (and display if needed) relevant updates of the ARE’s application state and runtime information.

Following, when an AsTeRICS model has been started and is running, a RESTful communication between the ARE model components and the remote URC sensors/actuators is being established. The AsTeRICS model is then able to retrieve and or push information from/to the remote URC components. This functionality is provided through the URC actor component built within the AsTeRICS model (see Section 6.1). A URC actor component can be designed and developed for use with any URC remote sensor/actuator, and can be used by any AsTeRICS model, provided that the corresponding model component interconnections are being properly defined. This can serve as a good example of Software and Hardware reuse accomplished via the Integrated Runtime Environment, where a URC actor component (that handles a remote URC sensor) is designed and built once, but reused in many AsTeRICS models of different purposes without further coding needed. In addition, as the AsTeRICS platform enables a non-expert AT designer to easily model or reuse existing AsTeRICS models to provide the necessary functionality to the user, it can be easily inferred that having URC actor components available to be freely used within AsTeRICS models provides added value to the platform, and the ability to the non-expert AT designer to freely access remote URC sensors/actuators as well.

6.7 Setting up the Integrated Runtime Environment

To setup the Integrated Runtime Environment the reader is referred to ANNEX III of this document “Runtime Environment Setup Manual”. The manual describes thoroughly any prerequisites for installation, a quickstart guide, as well as the S/W and H/W installation steps needed to successfully install and configure the Integrated Runtime Environment. It also provides a few examples to ensure proper installation.
7 Use Case Scenario: Assisted Living

In this section we describe a use case scenario in order to examine how the integration approaches mentioned above can indeed be used effectively in real life situations.

7.1 Personas

We have used the personas described in D103.1 [23], which were created aiming to reveal the behaviour of various stakeholders in the ecosystem. For the purposes of this deliverable, we have used the following personas:

Mrs Moroz

Mrs Moroz has had a stroke a year ago, which paralyzed the right side of her body and took her speech away. Fortunately, she still has good head control which enables her to use a variety of different technologies. For example she is competent in using a head mouse control, a mouth stick and a chin control for her electric wheelchair. She spends a lot of her time in an Assisted Living dormitory.

Nicholas Gallo

Nicholas Gallo, an 11 year old, was diagnosed with cerebral palsy. He lives in a small town with his parents and younger brother. He moves around in an electric wheelchair, which he controls via a joystick. His hands need to be placed on the joystick by a care-person and then he can use it with very small movements of his fingers. However, using the joystick is exhausting for him. That’s why he prefers to control the mouse with (limited) head movement whenever possible. Nicholas also spends time in the Assisted Living dormitory.

Vasili Moroz (Care Giver)

Vasili Moroz is Mrs Moroz husband, 78 years old and is also the care giver of Mrs Moroz in the dormitory.

James Olsen (AT Developer)

James is an independent developer working for the past 15 years in the area of accessibility. He is interested in supporting people regarding accessibility by using open source AT and he is very interested in supplying the Assisted Living dormitory with AT to facilitate the people with moving limitations that spend time there. He is currently working with the DSpace tool and he is excited to discover various solutions, technologies and applications for accessibility within DSpace that he can use as is or extend to meet the needs of the people in the
dormitory. He and a small group of passionate AT developers, like experimenting with AT S/W and H/W and find the idea of being AT enablers for the dormitory fascinating. Their future plans include starting a company specializing on providing full AT solutions based on open source platforms, such as the DSpace, therefore the dormitory is an excellent opportunity to start applying AT solutions and test them with real users.

7.2 Environment/Available Technology

Mrs Moroz and Nicholas Gallo spend much of their time in assisted living, a really nice and warm dormitory that provides accessibility. Specifically, Mrs Moroz spends her weekdays there, while Nicholas spends only a few hours on most of the week days. The dormitory has a separate bedroom for every resident and a common room for getting in contact with other people. The common room has two different areas. One is equipped with a TV set and the other one with tables so that people can sit together.

The common room, as well as each bedroom, is equipped with the following:

- Philips HUE lights (a lamp that changes colours remotely controlled by URC – it is a remote URC actuator) so that it can be illuminated to individual colour wishes.
- An electricity outlet (Wöhlke Websteckdose that is remotely controlled by URC – it is a remote URC actuator as well) connected with a fan heater that can be turned on/off from the network
- Both Philips HUE lights and the electricity outlet in a room can be controlled from a central control panel within the room, a computer screen that can adapt to users’ preferences

7.3 Personalized Technology Control

The central Philips HUE light and the fan heater can be controlled via the computer screen. All three people, Mrs Moroz, Nicholas and Vasili should be able to control them as follows:

Mrs Moroz:

- needs to use the headmouse feature with sensitivity set to “normal”: requires normal head movements to operate the mouse. Keeping still for four seconds over something selects it.
- due to her wheelchair the distance between her and the control panel is about 1.0m to 1.5m

For Mrs Moroz the following is needed: “normal” headmouse control + increased button size.
Nicholas

- needs to use the headmouse feature with sensitivity set to “increased”: even with limited head movements the mouse can be operated. Keeping still for four seconds over something selects it.
- due to his wheelchair the distance between him and the control panel is about 1.0m to 1.5m

For Nicholas the following is needed: “**sensitive** headmouse control + increased button size.

Vasili Moroz

- wants to use a mouse or touchscreen
- needs to see as many information as possible (e.g., lights + fan heater)

For Vasili Moroz **no assistive technologies are needed + normal button size.**

**General required adjustments:**

- increased font size/less information for bedridden people due to increased distance between display and user
- different input technologies: “**sensitive**” headmouse control, “**normal**” headmouse control

**Room parameter control**

Every room has its own control panel which must adjust to each resident’s preferences for that room. The control panel in the room enables controlling the Philips HUE lights and the fan heater in a personalized manner.

### 7.4 Enabling the Scenario

James Olsen has undertaken the job of supplying the Assisted Living dormitory with AT to facilitate the people with moving limitations that spend time there. James was thinking that what he really needed as a developer was a clever, easy and fast way to integrate a number of different technologies from AT, Smart Interfaces, Smart Sensors and Accessibility domains to be able to facilitate the many different, as well as very specific needs of the habitants. Before becoming aware of DSpace, he could not think of a solution to his problem. He could of course use many different technologies and frameworks, but the installation, maintenance and support costs, as well as the time needed for his developers’ team to learn...
all the different technologies had made him reluctant to proceed. Not to mention the many
different types of S/W and H/W licenses he would need to manage.

A few weeks now he has been accessing a new environment that is promising solutions to
problems such as the one he is facing. The DSpace is an open source platform that provides
various solutions, accessible components, tools, parts, and building blocks for developers to
browse, search, learn about, and use. After spending some hours familiarizing with DSpace,
he has effectively used DSpace’s search and indexing tools to search for a solution to his
problem, retrieving as a potential solution the Integrated Runtime Environment. This
environment seemed promising as it combined support for diverse and individually tailored
AT, access to a variety of services and remote sensors/actuators, as well as usage of flexible,
runtime adaptive user interfaces that match dynamic personal needs of the user.

James felt that this Integrated Runtime Environment is promising and that it can very well be
the best ecosystem offering solutions for technologically supporting the dormitory. He and
his developers team had spent some time with the installation and developers’ manuals to
learn the full potential of the tool, and after some experimentation, they decided to actually
use it experimentally at the dormitory (as a first test to support Mrs Moroz and Nicholas
Gallo, therefore technologically facilitating 3 rooms: their private rooms and the common
room). Provided that the Integrated Runtime Environment is open source and therefore free
for use, the costs for their experiment included only the cost for purchasing three cameras,
three Philips HUE lamps, three electricity outlets Wöhlke Websteckdose, three fan heaters
and three PCs with large screens.

7.5 Scenario Description

Scenario Part 1

Mrs Moroz is sitting in her bedroom where she has set the Philips HUE lights and fan heater
according to her current needs: “light green” light and fan heater “on”. At some point she
uses her electric wheel chair to move to the common room to watch TV. Since nobody has
been there for a while, the temperature is lower and the lights are out, so Mrs Moroz needs
to switch on the fan heater, as well as control the lights in the common room. She uses the
control panel in the common room to make the necessary changes so she needs to use
increased button size (to be able to read characters from a distance) along with “normal
headmouse control”. The system makes the necessary adjustments to assist Mrs Moroz.

Scenario Part 2

After some period of time, Nicholas also enters the common room. He has been out with his
parents and brother. He feels hot so he thinks of switching off the fan heater. Also, he wants
to read a new book his brother gave him, so he asks Mrs Moroz’s permission to make the
room a little bit brighter by changing the colour of the lights to something brighter, as well as less hot by switching off the fan heater. Mrs Moroz doesn’t mind so Nicholas approaches the control screen. Due to his wheelchair the distance between him and the control panel is about 1.0m to 1.5m, so the system increases the font size and enables “sensitive headmouse control”.

Scenario Part 3

When Vasili Moroz enters the common room 2 hours later, he feels the room is a bit cold and he wonders how come the temperature is so low. He approaches the control screen in the common room and through the device normal interface (common font size, etc.) he switches on the fan heater. He then asks Mrs Moroz and Nicholas whether they had any problems with the technological equipment in the dormitory: the Philips HUE lights, the fan heater, the touch screen and the TV set. Mrs Moroz mentions that she would prefer the headmouse control to be even less sensitive so that she can handle it better. Vasili Moroz communicates with James Olsen (the AT developer) and informs him about Mrs Moroz’s wish. James then connects to the dormitory machines via remote connection, accesses the AsTeRICS ARE and adjusts the headmouse sensitivity according to Mrs Moroz’s needs and preferences. He then informs his AT developers group of the problem for consideration for future developments: a good idea would be to provide an easy to use interface for the user to be able to real-time adjust mouse sensitivity for the headmouse control of AsTeRICS.

7.6 Components, Tasks and Roles

In the above scenario it is assumed that the GPII has access to the particular user needs and preferences (NPs) for each of the three protagonists that are active in a particular room, as well as important context parameters such as the user position in terms of the room she is currently in. GPII is able to identify users by using NFC technology: the user has a unique NFC tag that is readable by the NFC reader connected to GPII, informing the latter of who was identified in the proximity. MyUI and ARE are represented within GPII as “solutions” that special components of the GPII such as the Matchmaker and the Lifecycle Manager can start or stop depending on user needs, preferences and the context.

The three parts of the above scenario can be developed via using any of the two distinct communication routes between AsTeRICS ARE, MyUI and URC, as described in Section 6.5, each route providing different results and therefore advantages.

7.6.1 ARE → MyUI → URC

In this case the communication between ARE and MyUI is being conducted over the operating system, while MyUI remotely controls URC sensors/actuators. The AsTeRICS
model running in the ARE provides accessibility and AT to control operating system events such as mouse or keyboard interaction in order to control the MyUI Adaptive User Interface.

**Scenario Part 1**

As Mrs Moroz enters the common room, GPII detects her entering the room. Mrs Moroz needs to switch on the fan heater and control the lights, therefore she uses the control panel (screen) to make these changes. Based on Mrs Moroz’s NP (Needs Preferences), GPII needs to specify “increased button size” to MyUI so that Mrs Moroz is able to read characters from a distance. MyUI will start as a GPII “solution” with parameters “increased button size”, while it will also initiate and maintain connection with URC. Moreover, GPII starts the ARE (on the PC in the particular room – same PC that GPII is running), and (i.) deploys and (ii.) starts the corresponding model responsible for the “normal” headmouse control actuator. Via the headmouse control Mrs Moroz can use normal head movements to interact with the increased button size UI of MyUI (Figure 18) to control the remote URC actuators: change the colour of the Philips HUE lights (see Figures 19 and 20) and switch on the fan heater (Figure 21 depicts the electricity outlet Wöhlke Websteckdose that controls the fan heater).

![Figure 18: MyUI increased button size UI for usage with the AsTeRICS headmouse model](image-url)
Ecosystem infrastructure for smart and personalised inclusion and PROSPERITY for ALL stakeholders

Figure 19: Mrs Moroz can change the colour of the Philips HUE lights through the MyUI UI via head movements

Figure 20: The Philips HUE lights in different colours

Figure 21: The electricity outlet Wöhlke Websteckdose that controls the fan heater
Scenario Part 2

When Nicholas enters the common room, he needs to switch off the fan heater and also to change the colour of the lights to something brighter. Nicholas approaches the touch screen. The GPII, based on Nicholas’s NP (Needs Preferences), understands that due to his wheelchair, the distance between him and the control panel is increased (about 1.0m to 1.5m), so the system needs to increase the font size and also enable “sensitive” headmouse control. MyUI will again start as a GPII “solution” with parameters “increased button size” and initiate/maintain connection with URC, while the ARE will be initiated by GPII as a solution and (i.) deploy and (ii.) start the corresponding model that utilizes the “sensitive” headmouse control for Nicholas to use. Nicholas will then be able to use very limited head movement (as a limitation of his illness) to control the mouse and interact with the increased button size UI of MyUI to control the remote URC actuators: change the colour of the Philips HUE light and switch off the fan heater.

Scenario Part 3

When Vasili Moroz enters the common room, he approaches the touch screen and needs to adjust various settings through the device normal interface (common font size, etc.). The GPII knows Vasili Moroz’s NP (Needs Preferences) and is able to initiate MyUI and ARE as solutions to facilitate Vasili Moroz. In this case, MyUI will specify common font size, while the ARE will not start a particular actuator/sensor since one is not needed for Vasili Moroz’s case. Vasili Moroz is able to turn on the fan heater via accessing the remote URC actuator through MyUI interaction.

7.6.2 MyUI → ARE → URC

In this case the MyUI Adaptive User Interface is used to manage the AsTeRICS ARE, while the ARE handles the communication with the URC remote sensors/actuators. At first AsTeRICS ARE is started by GPII, with the appropriate AsTeRICS model initiated as well according to user NPs (e.g. the headmouse model), while MyUI is also initiated by GPII according to user NPs (e.g. increased font size). Through an appropriate MyUI user interface, the user is able via a simple click of a button to change the AsTeRICS model so that to be able to handle the remote URC sensors/actuators needed by using AT. The AsTeRICS model is then able to retrieve and or push information from/to the remote URC components.

Scenario Part 1

As Mrs Moroz enters the common room, GPII detects her entering the room. Mrs Moroz needs to switch on the fan heater and control the lights, therefore she uses the control panel (screen) to make these changes. Based on Mrs Moroz’s NP (Needs Preferences), GPII Ecosystem infrastructure for smart and personalised inclusion and PROSPERITY for ALL stakeholders
needs to specify “increased button size” to MyUI so that Mrs Moroz is able to read characters from a distance. MyUI will start as a GPII “solution” with parameters “increased button size”. Moreover, GPII starts the ARE and i. deploys and ii. starts the corresponding model responsible for the “normal” headmouse control actuator. Via the headmouse control Mrs Moroz can use head movements to interact with the increased button size UI of MyUI to change the AsTeRICS model via a simple click of a button so that she is able to handle the remote URC sensors/actuators needed by using AT. Now, Mrs Moroz can handle the remote URC sensors via head movements as follows:

- Philips HUE light: head movement continuously changes between available colours. By keeping still for 4 seconds Mrs Moroz can return to the “normal” headmouse control model to continue with other tasks. The colour of the lamp is then the last selected colour. For more details see ANEX III section 5.2.

- Switch on the fan heater: mouth movement (opening) changes between “switch on” and “switch off”. When the fan heater is set to the desired state, Mrs Moroz can return to the “normal” headmouse control model with a specific head movement (turning head to the right) to continue with other tasks. For more details see ANEX III section 5.2.

**Scenario Part 2**

When Nicholas enters the common room, he needs to switch off the fan heater and also to change the colour of the lights to something brighter. Nicholas approaches the control panel. GPIII, based on Nicholas’s NP (Needs Preferences), understands that due to his wheelchair, the distance between him and the control panel is increased (about 1.0m to 1.5m), so the system needs to increase the font size and also enable “sensitive” headmouse control. MyUI will again start as a GPII “solution” with parameters “increased button size”, while the ARE will be initiated by GPIII as a solution and i. deploy and ii. start the corresponding model that utilizes the “sensitive” headmouse control for Nicholas to use. Nicholas will then use “sensitive” headmouse control to interact with the increased button size UI of MyUI to change the AsTeRICS model via a simple click of a button so that he is able to handle the remote URC sensors/actuators needed by using AT. Now, Nicholas can handle the remote URC sensors via head movements as follows:

- Philips HUE light: head movement continuously changes between available colours. By keeping still for 4 seconds Nicholas can return to the “sensitive” headmouse control model to continue with other tasks. The colour of the lamp is then the last selected colour. For more details see ANEX III section 5.2.

- Switch off the fan heater: mouth movement (opening) changes between “switch on” and “switch off”. When the fan heater is set to the desired state, Nicholas can return to the “sensitive” headmouse control model with a specific head movement (turning...
head to the right) to continue with other tasks. For more details see ANEX III section 5.2.

**Scenario Part 3**

Vasili Moroz does not need AT. Therefore he would prefer to control the fan heater via the MyUI interface by using the touch screen, rather than via head movements through AsTeRICS.

**7.7 Results**

Mrs Moroz and Nicholas Gallo are excited with the many interaction possibilities their new equipment provides. They really like the very easy and straightforward way they can handle the lights and fan heater to bring them to suit their needs and preferences. And they feel excited knowing that more devices can also connect in the future to provide even more cool solutions, making their lives at the dormitory easier.

However, the two end users are not as excited as James Olsen. James is happy that their first experiment was successful, satisfying thus the users, but his real excitement stems from the fact that:

- He has managed to perform a successful technological experiment at the dormitory while *lowering development costs to a minimum*: both equipment and development time for his team were minimal compared to the development time and equipment they would need to spend without using the Integrated Runtime Environment.
- The *potential* of this technology is great: he can use it to address a number of different problems in AT and smart adaptive UIs. Moreover, the platforms that constitute the Integrated Runtime Environment can be extended to facilitate more and diverse user needs, as well as to be integrated with even more platforms and applications.
- He and his team are now eager to start their own company specializing on providing full AT and Smart UI solutions based on open source platforms, such as the DSpace. They believe that they can begin by using and extending the Integrated Runtime Environment and potentially use more technologies from the DSpace to try to *achieve better market access for their company*. 
8 References

[6] OSGi - The Dynamic Module System for Java, Online: http://www.osgi.org/Main/HomePage
[8] AsTeRICS code (Github Repository), Online: https://github.com/AsTeRICS/AsTeRICS
[18] MyUI Developer Documentation, Online: 
http://myuidocumentation.clevercherry.com/overview/basic-conceptst

[19] Lukas Smirek, Gottfried Zimmermann, Christos Mettouris, Marios Komodromos, 
Achilleas Achilleos, George A. Papadopoulos, Daniel Ziegler and Michael Beigl. 
"Accessible Control of Distributed Devices: Supporting People with disabilities by 
Providing Adaptive Interaction", In Proceedings of the First International Conference on 
Universal Accessibility in the Internet of Things and Smart Environments (SMART 

Report for T203.3, D203.1

[21] JmDNS java library for a zeroConf implementation: 
http://jmdns.sourceforge.net/

https://tc39.github.io/ecma262/#sec-promise-objects

[23] Prosperity4All: Deliverable D103.1 “Foundational Design for Prosperity4All”