



make it ReAAL

Project co-funded by the European Commission within the ICT Policy Support Programme

ReAAL

CIP ICT PSP – 2012 - 325189

www.cip-reaal.eu

universAAL compliance guidelines

[Deliverable D3.3, Revision 1.1]

Key Information from the DoW

Due Date	15-October-2015
Type	Report
Security	Public

Description:

This deliverable is a compilation of the coaching experiences in ReAAL, providing a monitoring and evaluation concept for the adaptation of products and services to the universAAL platform.

Lead Editor: Alvaro Fides (UPVLC)

Internal Reviewer: Ad van Berlo

Versioning and contribution history

Version	Date	Author	Partner	Description
0.1	24-Jul-2015	Pilar Sala	UPVLC	Structure of the document and task assignments
0.2	28-Sep-2015	Álvaro Fides, Michele Girolami	UPVLC, CNR	Aggregated all draft content so far
0.3	29-Sep-2015	Álvaro Fides, Michele Girolami	UPVLC, CNR	Added all available contributions. Rewritten common sections.
0.4	30-Sep-2015	Pilar Sala	UPVLC	Finalized Executive summary. Preliminary release for Review
0.5	30-Sep-2015	Saied Tazari	Fh-IGD	Intermediate version for project review
0.6	25-Oct-2015	Pilar Sala	UPVLC	Restructured content
0.7	9-Nov-2015	Pilar Sala	UPVLC	Updated section 4.1
0.8	11-Nov-2015	Saied Tazari	Fh-IGD	Updated section 2
0.9	12-Nov-2015	Alvaro Fides	UPVLC	Updated section 4
0.95	12-Nov-2015	Pilar Sala	UPVLC	Integrated contributions in final version, ready for internal review
0.96	14-Nov-2015	Pilar Sala	UPVLC	Applied reviewer comments. Final version ready for release
1.0	15-Nov-2015	Helmi Ben Hmida	Fh-IGD	Official release
1.01	2-Mar-2016	Pilar Sala	UPVLC	Revision of structure to address reviewers comments
1.02	15-Mar-2016	Álvaro Fides, Michele Girolami, Alejandro Medrano, Richard Pasmans	UPVLC CNR UPM SmH	Rewritten pilot sections, organized the other sections and provide contributions
1.03	21-Mar-2016	Pilar Sala, Alvaro Fides	UPVLC	Applied reviewer comments. Final version ready for release
1.1	21-Mar-2016	Helmi Ben Hmida	Fh-IGD	Official release

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Executive Summary

This document is the result of the work in task T3.3 Coaching of application providers by platform experts. Within this task the platform technical experts have accompanied the application porting process providing training, coaching and technical support to the application providers and pilot investors with the aim of ensuring that the porting is done with good quality. The ultimate goal is that both the related knowledge is spread more widely and failure risks during deployment and operation are minimized.

A set of instructions has been devised to lead through the process of adapting applications to universAAL, and how this process can be monitored and later evaluated to make sure the adaptation is correct.

To lead pilots through this process, platform technical experts have been assigned to coach and oversee each pilot site, according to several factors such as the proposed piloting concept, the geographical proximity or the shared idiom to facilitate communication.

Coaching basically relies on technical partners having a frequent communication with pilots and their technical staff, through many different means, such as teleconferences, physical meetings or asynchronous support; and it can be originated either by the technical expert, to support basic training or discuss project strategies, or by the application provider, to solve issues encountered during development.

The experiences in coaching have been quite varied, due to the fact that each pilot site has its own context and particularities, some have multiple providers that need coaching, others only one provider, some have more complex deployments than others, etc. For part of the pilots, off-line coaching was provided during the initial phase when companies started learning the universAAL basic concepts, while for others, closer, face to face sessions were organized. In some cases, pilot developers worked mostly autonomously until deploying the application while in others close cooperation was requested including support during deployment.

The lessons learned through this process have been useful to derive development patterns and best practices that will be shared with the community of developers to support anyone interested in creating universAAL compliant applications and services.

.

Table of Contents

1.	About this document	6
1.1.	Deliverable context.....	6
2.	Monitoring and evaluating the adaptation to universAAL.....	8
2.1.	Common universAAL terms.....	8
2.2.	Introduction to the architecture of universAAL.....	9
2.3.	The concept of universAALization	11
2.4.	The adaptation process.....	12
2.4.1.	Analysis and design	13
2.4.2.	Implementation	15
2.4.3.	Test and evaluation	21
3.	Coaching process in ReAAL	24
3.1.	Generic coaching process	24
3.2.	Pilot coaching experiences.....	26
3.2.1.	AJT – SL	26
3.2.2.	AJT – WQZ	27
3.2.3.	BSA	28
3.2.4.	BRM	32
3.2.5.	IBR	32
3.2.6.	ODE	34
3.2.7.	PER	35
3.2.8.	PUG	36
3.2.9.	RNT	40
3.2.10.	TEA	41
3.2.11.	Associated Pilots	43
4.	Generic lessons learned	44
4.1.	Frequently asked questions.....	45
5.	Conclusion.....	48

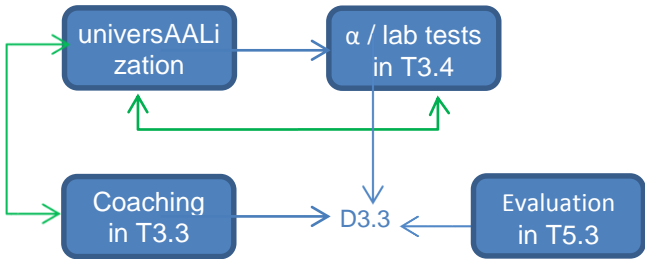
1. About this document

This document is reporting on the process and progress of task T3.3 Coaching of application providers by platform experts. Within this task, the platform technical experts have accompanied the porting process from T3.2 by providing training and technical support to the application providers and pilot investors with the aim of ensuring that the porting is done with good quality. The ultimate goal is that both the related knowledge is spread more widely and failure risks during deployment and operation are minimized.

The document is structured in two main sections. First, starting with an introduction about the terminology used in relation to the universAAL platform, we describe the basics of the platform and the monitoring and evaluation concept for adapting an application to be used on top of universAAL platform.

Secondly, the information about how the coaching process and experiences in ReAAL has been provided, applying the concept described in the previous section, and reporting on the methods and tools used by the different pilots and platform experts, as well as the lessons learned from the process.

1.1. Deliverable context

Project item	Relationship
Objectives	<p>Contributes to the following project objectives:</p> <ul style="list-style-type: none"> O1: deploy at least 7 universAALized applications and services O3: an initial universAAL ecosystem O5: knowledge sharing O8: sustainable exploitation
Work plan	<p>The deliverable D3.3 universAAL compliance guidelines is an outcome of T3.3 Coaching of application providers by platform experts. The relationship with the other parts of the work plan is as follows:</p>  <pre> graph TD A[universAALization] --> B["α / lab tests in T3.4"] C[Coaching in T3.3] --> D[D3.3] E[Evaluation in T5.3] --> D B --> A B --> D </pre>
Milestones	<p>Contributes to the achievement of the following project milestones:</p> <p>MS4 – Final corrective actions</p>
Deliverables	<p>This is a single deliverable; however, it receives input from the experiences in several tasks in addition to T3.3, such as T3.4 Lab Tests, or T5.3 Evaluation execution.</p>

Exploitable results	Contributes to the following exploitable results of the project: Res 6: Guidelines for monitoring and evaluating the adaptation of products and services to the universAAL platform
Risks	Contributes to the clearance of the following project risks: Rk3: Difficulties and delays in the integration with the platform Rk4: Interoperability problems between different subsystems. Rk5: universAAL runtime platform failure during pilots

2. Monitoring and evaluating the adaptation to universAAL

2.1. Common universAAL terms

In order to better understand the following sections, the definitions of the most common terms are provided here:

Concept	Definition
AAL Space	A smart environment (in the sense of a physical space, such as a home, a car, an office, a supermarket, a hospital, an airport, etc.) with defined boundaries of hardware, software, and human resources (in terms of, e.g., access control, multimodal user interfaces, etc.), which is capable of responding to the needs of its users in an adaptive way by providing relevant assistance.
AAL Platform	Usually used to refer to the software that supports the creation of integrated AAL systems by providing a logical layer that facilitates the integration of interoperable AAL applications as well as a standard set of capabilities commonly needed by AAL applications.
AAL Application	Software that implements certain use cases involved in the provision of AAL Services. It may consist of several software modules and / or utilize special-purpose devices (e.g., sensors, actuators, medical devices, etc.).
Application	In this context it is referred as the value provided by the pilot, the “original” software and services that are to be adapted, as opposed to an “AAL Application”, which is already adapted to universAAL.
Application module	Each of the independent software parts that, working together, provide the value of an application in a pilot.
Deployer	Whatever stakeholder tasked with the responsibility of adapting an existing pilot application to universAAL. This may represent stakeholders at different entities and with different roles, such as developers, maintainers, designers, etc.
Ontology	In information technology, an Ontology is an explicit specification of a set of concepts and their relationships within a domain in a formal language that is developed with the goal to be shared by humans and machines as a shared understanding of the domain. Hence, ontologies are explicit and sharable domain models specified in a formal language. Being based on a formal language, ontologies become machine-readable, which enables computer programs to benefit from ontologies in data and information processing. Conventional computer programs usually include domain models, as well; but such models are not suitable for explicit

	<p>sharing because (1) due to linkage to an associated program code, some assumptions in the code find no reflection in the model itself and remain implicit, (2) the formalism of programming language is not powerful enough for creating standalone models, and (3) share-ability remains confined in the scope of a certain technology.</p>
universAAL Middleware	<p>The core part of universAAL that provides the logical horizontal integration layer for creating integrated systems across distributed components and subsystems. It hides distribution and possible heterogeneity of the components and acts as a broker between the communicating parties. This way, it facilitates integration and communication and effectively provides for interoperability among components that may have been developed widely independently from each other. The kind of interoperability enabled by the universAAL Middleware is called “semantic interoperability” because it eliminates syntactical dependencies among the communicating components and reduces their dependencies to shared domain models or ontologies.</p>
universAAL Buses	<p>Part of the universAAL Middleware with the ultimate integration and communication logic that implements the semantic brokerage mechanisms and handles the flow of messages among the communicating parties. For each type of communication, there is a specific bus with its own model, strategy and match-making: (1) the Context Bus for event-based communication based on the subscribe / publish pattern, (2) the Service Bus for the general call-based communication based on the request / response pattern, (3) the User Interaction Bus (UI Bus) for the specific call-based communication when utilizing interaction services, and (4) the Control Bus for the specific call-based communication with regard to configuration management. It is these Buses to which applications connect in the process of integration.</p>
universAAL Managers	<p>A universAAL Manager is a software component that runs on top of the universAAL Middleware as part of the platform itself. universAAL is distributed with a set of such manager components that are necessary for its proper operation or provide relevant basic services and / or events to the other managers and / or to the application layer.</p>
universAALization	<p>Also mentioned as “uAALization”. The process of adapting an existing application to be compatible with universAAL. This is described in more detail in the corresponding section.</p>

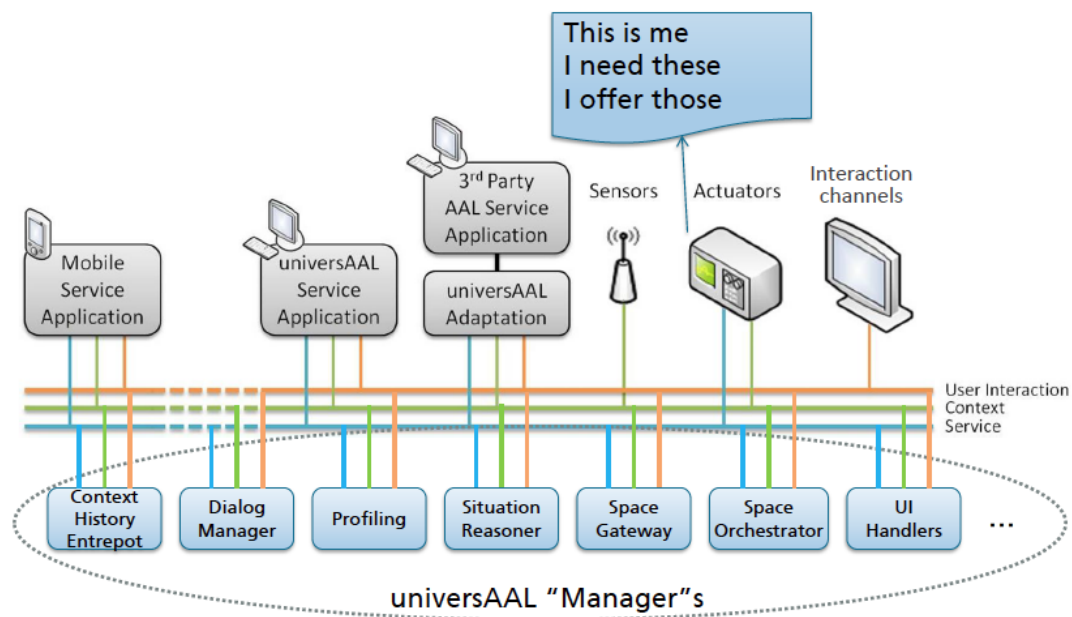
2.2. Introduction to the architecture of universAAL

With its worldwide unique implementation of semantic interoperability for SoA at the level of communication protocol (existing since 2008), universAAL provides an open horizontal service integration layer at the highest abstraction layer, across all possible verticals. By avoiding domain-specific APIs (reduction of all possible

syntactical dependencies to one single message brokerage API), universAAL has a unique future-proof contribution to managing the IoT complexity.

The open source software – distributed with the Apache Software License 2.0 – consists of:

- The distributed implementation of three "group-level" brokers (context, service and UI buses) that ensure integration and semantic interoperability while hiding distribution and heterogeneity.
- A fourth broker supporting configuration management at node level (the control bus).
- A set of "managers" on top of that (providing for shared mechanisms for security, user interaction & accessibility¹, system memory, intelligent & adaptive system behaviour², remote and inter-group (also multiple with or without hierarchies) interoperability, and configuration management).
- A set of concrete ontologies.
- A set of development, deployment, and administration tools.
- Example applications.
- Documentation, wiki pages and training material.



A simplified view of applications, devices and managers connected to the universAAL buses

With its 30+ applications, 100+ services offered by these applications to 5000+ users in 13 pilot sites in eight countries, ReAAL is serving as stress-tester of

¹ The user interaction framework is specifically designed and developed for smart environments and already has an internationally recognized status [IEC/PAS 62883 Ed. 1.0] as an open approach with considerable potential to become an important standard in near future.

² Architecturally, the framework supporting context-awareness and profiling has a very open approach that enables to achieve an intelligent and adaptive system behaviour.

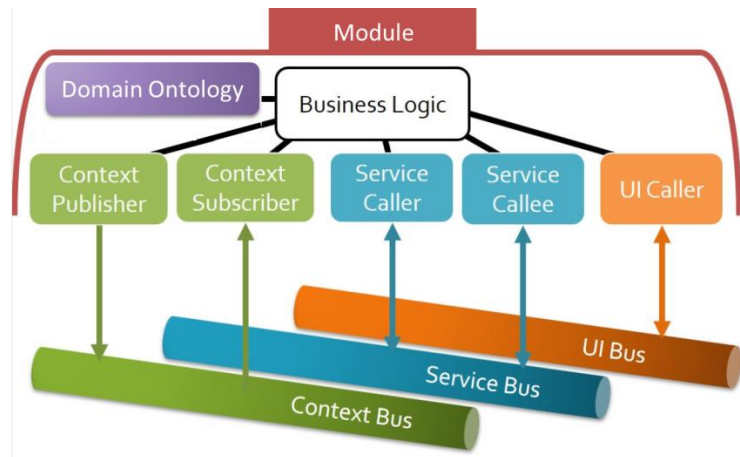
universAAL in real life. The socio-economic impact of universAAL is being evaluated in the first half of 2016 during operation in real life. In other words, universAAL as a domain-independent integration platform is, through ReAAL, getting ready for mission-critical deployments.

The following is a non-comprehensive list of available resources with information about the platform that are available to pilot deployers as well as for any external developer or third party:

- The universAAL project in GitHub (<https://github.com/universAAL/>) holding the current source code and documentation wikis, including some of the main entry points:
 - The main wiki of the 'platform' repository:
<https://github.com/universAAL/platform/wiki>
 - The Developer Handbook:
<https://github.com/universAAL/platform/wiki/Developer-Handbook>
 - The Quick Developer Guide:
<https://github.com/universAAL/platform/wiki/Developer-Guide>
- The universAAL depot (<http://depot.universaal.org/>): A complementary website for developers who want to develop applications on top of universAAL or contribute to its maintenance.
- The universAAL GForge website (<http://forge.universaal.org/>) which still holds the issue trackers, including the ones for generic purpose:
 - Support Issue Tracker:
<http://forge.universaal.org/gf/project/support/tracker>
 - Public Help Forum:
http://forge.universaal.org/gf/project/support/forum/?action=ForumBrowse&forum_id=8
- The universAAL mailing list (<http://universaal.aaloa.org/mailman/listinfo/universaal-dev>) for both users and contributors
- Maintenance sponsors (<http://www.universaal.info/marketplace/consultancy-services/>): Organizations contributing to the universAAL maintenance and providing support, training, and consulting services with regard to application adaptation to universAAL, either for specific runtime environments or for any other specific requirements. These services would be similar to those performed during the coaching process in ReAAL.

2.3. The concept of universAALization

With the term *universAALization* we refer to the integration of all functional components to be deployed with the universAAL software platform prior to their deployment. The goal of the universAALization process is to resolve the dependencies between these components for the exchange of data and functionality based on an open platform in order to achieve future-proof interoperability, adaptability, and extensibility to a wider extent.



The universAALization consists of two major steps: (1) selection and completion of a set of application-related ontologies, and (2) using the application programming interface (API) of the universAAL middleware together with the selected ontologies for the actual integration of the components. An application is considered to be universAALized (or “uAALized”) when it enables (allows or performs) interaction through one or more of the uAAL buses based on an ontological model.

2.4. The adaptation process

The following is an abstracted workflow representing the patterns followed by pilot sites when adapting their solutions to universAAL. It represents a generic step-by-step approach for any *deployer* to convert an existing solution to a compliant universAALized one. A *deployer* is understood as the stakeholder in charge of the implementation of the solution, which could be a pilot, a technical partner, a subcontractor, or a combination of these.

In the following, we summarize the overall workflow which will be described in detail in the next subsections:



1. Analysis and design

- a. *Analysis of initial scenario*: Identify the basic architectural distribution of the existing solution.
- b. *Compatibility with universAAL*: Identify the constraints of the existing solution and identify where universAAL is able to be deployed.
- c. *Selection of universAALization options*: Based on the analysed constraints, choose to introduce universAAL within the solution in the way that provides the desired added value.

2. Implementation

- a. *Design of ontologies*: Create a new, or choose existing ontologies that appropriately model the data needed by the solution.

- b. *Design of application modules*: Determine the structure the adapted application will have by analysing it from different model perspectives.
- c. *Code implementation*: When all design decisions are made, proceed to the actual coding of the application modules.

3. Test and evaluation

- a. *Adaptation evaluation*: Check that the design decisions and implementation have been performed as stated.

2.4.1. **Analysis and design**

The first stage does not require any change yet. It is only an analysis of the existing situation which leads to a final decision-making step that will lay the path to the actual implementation stage.

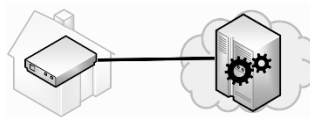
Analysis of initial scenario

The universAAL platform supports both local and server-based (cloud) scenarios, or a combination of these. The first analytical step will be to identify which one of these options corresponds to the existing initial architecture.

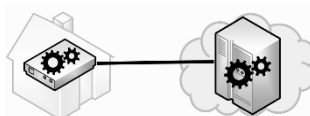
- **Client-based**: All the logic is run in devices deployed at the client side, whether that is in fixed or mobile devices at the Assisted Person’s home or in a mobile device the Assisted Person carries outside home.



- **Server-based**: All significant logic is run in a server hosted by the deployer. It is possible that some amount of processing is performed by devices at the client-side, but only for the purpose of sending the information to the server, not performing any processing on it – such as a gateway gathering sensor data to send them over the Internet on their behalf.



- **Client-Server**: Some data processing or business logic is performed by devices at the client side, while other takes place in a server. This may all be part of the same application or be combined by independent solutions – such as deploying a client-based application *and* a server-based application at the same time.



Compatibility with universAAL

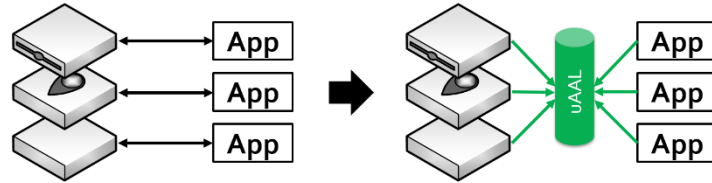
Identifying which one of the above scenarios corresponds to the existing architecture will help answer the following questions. These are aimed at identifying where the universAAL-aware nodes can be deployed (These are devices that can run universAAL).

- **Where are applications currently running?** In the initial architecture, identify in which devices and execution environments the business logic of the applications is being executed – is it a JVM in Windows? In Linux? Is it an Android app...? Which versions?
- **Where could universAAL run?** Right now, the universAAL platform can be executed in devices running Android 2.3 or above, and in Java Virtual Machines (JDK 5 as a bare minimum for the middleware) through an OSGi Container. Identify which of the existing devices in the initial architecture satisfies the requirement. For instance, in the case of OSGi, the hardware requirements are as low as mini-computers like the Raspberry Pi Model-A (700 MHz processor with 256MB of RAM), while for servers the requirements will depend on the number of clients to service and the complexity of the applications.
- **Can applications be universAALized “natively”?** When universAAL can be run in the same device as the initially deployed applications, consider if it would be possible to universAALize these applications directly. This “native” universAALization consists on having the applications communicate with a universAAL instance through the APIs corresponding to their execution environment. Android apps and OSGi bundles can do this, using the uAAL API in Android or in OSGi respectively (details are described in later steps). If this is not possible, alternative methods would have to be used (such as universAAL’s Remote API).
- **How will universAAL instances connect to each other?** If universAAL instances can be deployed in more than one node, determine if and how this communication will happen. Instances in the same IP network can communicate directly to each other. In different networks (or through the internet) they can communicate with the AAL Space Gateway or the Remote API. Or not be connected at all, if that is not necessary for the deployment.

Selection of universAALization options

Answering these previous questions does not yet provide the right choice for universAALizing, but will help taking the final decision on which of the 4 options should be deployed, and how to implement it. These universAALization options provide value to the deployer in different ways; it is how the deployer benefits from having universAAL – these options will only be roughly described here, they are introduced in detail in other sources, like D3.2. It is possible to combine more than one of these options. What options to take will depend on the aim of the deployer, the client/server scenario identified in the first step, and the restrictions identified by the previous questions.

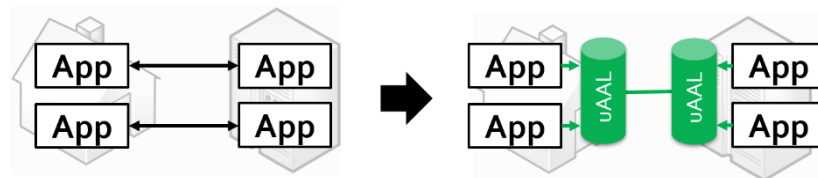
- **Decouple devices from applications:** Instead of hardwiring devices to applications, the communication is abstracted thanks to universAAL, which allows to connect newer devices from other technologies without modifying the applications, and allow other applications to benefit from existing devices.



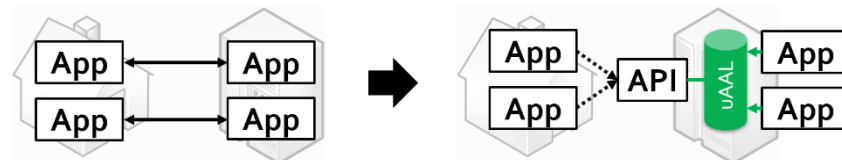
- **Decouple applications from each other:** Instead of hardwiring the communication between applications, it is performed through universAAL, which allows to connect new or different applications providing or consuming the same type of information, without having to change them.



- **Decouple applications from servers:** Instead of hardwiring the communication of client-side applications towards the server side, the communication is performed by universAAL, so for the client application it is like the server application is being run locally.



- **Virtualize applications in the server:** If universAAL cannot run in the client side, an instance of universAAL run in the server can virtualize this client, allowing client applications to benefit from universAAL even if it is not run locally.



To consider that a deployed solution (application, service or a combination of any number of these) has been universAALized, at least one of these 4 options must be implemented.

2.4.2. Implementation

Once it is known which universAALization options to implement, where to run universAAL and how and what to connect its instances to, it will be clear which application modules have to be universAALized (understanding application modules

as pieces of executable business logic, not the overall application, which can be comprised of many).

The process of universAALizing an application module involves making it run in a universAAL-compatible container (Android, OSGi) and perform/allow some interaction through universAAL buses. When an application module cannot run in those containers but still has to be universAALized, the Remote API can be used. In any case, it is imperative that the interaction with universAAL is made in terms of ontologies.

Design of ontologies.

Ontologies are the data model in universAAL. In order to communicate with universAAL the data exchanged must be represented by ontologies. Deployers must analyse the original data model of the application, and extract the basic information needed for the business logic, to a bare minimum, need-to-know basis. This helps identify the meaningful parts of the data that need to be shared (the “semantics”).

Ontologies are application- and hardware- agnostic, which means they should not model things like raw values of sensors, application-specific constants and the like, nor used to stream data (even at low frequencies). It must always be taken into account that other deployers should be able to understand the data model to work with it without knowing anything about how the application or hardware work (even if nobody else will use it in the end). Deployers should aim at having an ontology that is as future-proof as possible to reduce later needs of updates. This follows the criteria for ontologies already defined in documents like D1.2 about interoperability requirements, namely that ontologies should be: reusable, meaningful, extensible and future-proof. Because ontologies are key to interoperability, they are discussed in more detail in that deliverable D1.2.

Once it has been identified what kind of data the application is going to handle (sometimes referred to as the “domain”), it is necessary to inspect existing universAAL ontologies to see if there is already one that can be used. It could be that:

- There is already an ontology that covers all the needs of the application: In that case, use that one.
- There is one that covers the needs of the application but would need more concepts: Use that one and create an extension for it that includes the missing concepts. Ontologies allow extensions: additional modules that can define new concepts that “inherit” or “extend” existing concepts in other ontologies.
- There is none that covers the needs of an application: Create a new ontology specifically for the application, but trying to also make it as generic as possible so that other applications in the future can benefit from reusing it.

There are in-depth explanations on implementing ontologies³ and also reference guidelines on best practices⁴ in the universAAL documentation.

³ <https://github.com/universAAL/platform/wiki/RD-Understanding-Ontologies>

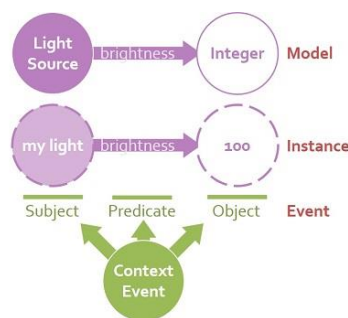
⁴ <https://github.com/universAAL/ontology/wiki/Guidelines>

Design of application modules

When all needed ontologies are identified and ready, the application module must be modified to make use of them and the universAAL APIs for connecting to the buses. But before starting the actual implementation or modification of the code, it is best practice to analyse the software pieces of the modules to be clear about overall structure of the application, what needs to be modified, what needs to be created from scratch, and what “goes where”. This can be studied from different model perspectives:

Data model: It is necessary to define how the ontological concepts will be arranged by each module of the application and how they will be shared through the buses. Each bus has its own format and best practices to follow.

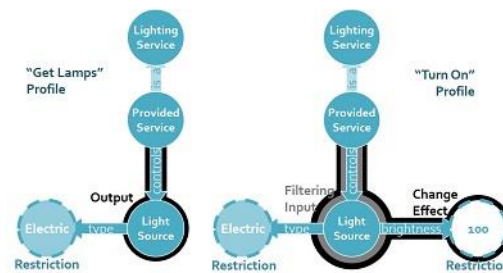
- In the Context Bus, information is shared in the form of Context Events. A Context Event represents a change in the physical context. These changes must be meaningful and relevant to the applications, the bus was not designed to “stream data” or share “raw” sensor data.
- In the Service Bus, it’s Service Requests that are shared, matched against Service Profiles. To build these it is necessary to start from a Service Ontology. Base Service Ontologies are usually included in the related ontology module, but more can be created in the application modules. The module containing the Service Ontology must be available to both requesters and providers of services.
- In the User Interaction Bus the information is shared as Forms. Forms are described in a default ontology included with the Bus. In general its concepts are enough for most applications, but just like any other ontology, they can be extended. However, UI Handlers that render the UI should be prepared to handle any new concept so this is not recommended. When used in declarative mode it can map forms to ontology resources.



Example: A Context Event has to link the right resources in the ontological model

Service Model: Services provided by each application module are defined according to the Service Bus API. The Service Bus implements a Semantic-oriented architecture. This means that rather than specifying explicit, method-like services (like “turnOnLight(X)”) the capabilities of each application module should be described in ontological terms (like “change brightness of a given light to 100 %”). This is described in terms of Service Profiles, and these profiles are invoked by Service Requests. Service Profiles are the equivalent to methods. They represent the operation to perform. Service Requests are the counterpart of Service Profiles. The Requests are matched to Profiles if they are ontologically equivalent. This

means that a Request does not need to be 100% equal to a Profile, only that it “fits” its description.



Example: Profiles are defined semantically over the same shared model of the ontology

Member model: For an application module to share any of the above items to the buses it must implement the appropriate Bus member.

- Context Publishers allow to send Context Events to the Context Bus. They define what type of provider they are and what type of events they publish. The provided events should be described as specific as possible, while leaving room for all the possible variations the application can provide.
- Context Subscribers define what type of Context Events an application wants to receive. These must also be described as specifically as possible. Only the Subject-predicate-Object triple of an event is guaranteed to be delivered. If additional information is required this must be defined in the description of the subscribed events.
- Service Callees handle the execution of a called Service Profile. An application module can have more than one Callee. Each Callee can answer to multiple Profiles. The Profiles are registered when the Callee is constructed, but it is also possible to add or remove them later. A Callee can use its own instance of a Service Ontology to define its own restrictions to narrow its provided services. This is done by creating an extension of the Service Ontology, usually named Provided Service. The Service Profiles are usually defined in this Provided Service class too.
- Service Callers issue Service Requests for the execution of a service and receive the response. One Caller is enough for most application modules, but there can be more. In most cases it is enough with a prebuilt default implementation.
- UI Callers build and send Forms that will be rendered by UI Handlers. Then they will receive the result and input from the user, and must act accordingly.

Looking in to the business logic of each member will determine if different members can be combined into a single member, when the internal logic is similar enough. Similarly, when the internal logic is very complex it might indicate that the member should be divided into different members.

Module model: Determine how the application logic is divided into different application modules, and how these interact with each other. This involves assigning different pieces of business logic, data representation (and/or ontologies), and, in uAAL, also Bus members, to each module. Interaction between modules can happen in any way, not necessarily through universAAL (although at least one will

have to, for the overall application to be uAALized), but doing so will maximize reusability and interoperability with other applications.

A module should be the smallest application part for which any configuration applies. Modules should have self-contained configuration elements (when needed). This means that the configuration of an application is the collection of configuration elements of all its modules, and each configuration element applies only to its module. Some global configurations may be replicated throughout the modules, but this will help increase the reusability of each module.

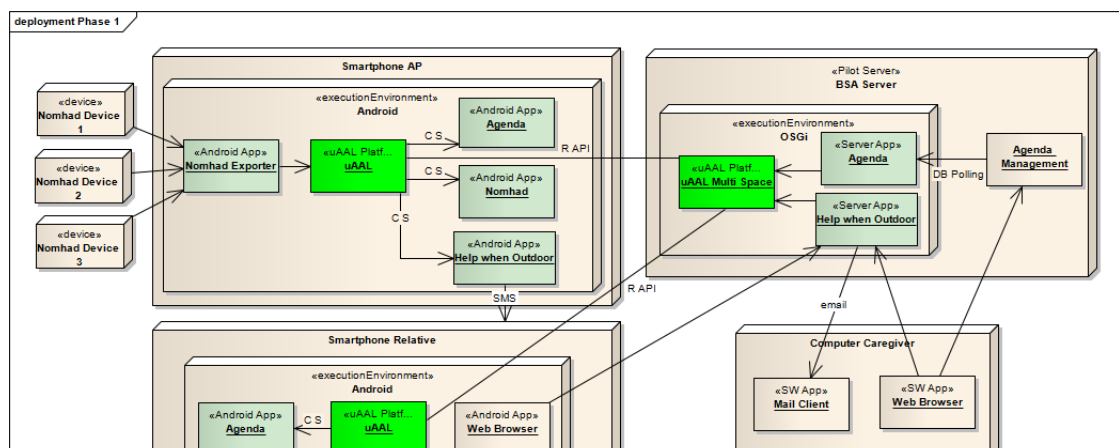
Distribution model: Determine which independent software modules will compose the overall application, and where each of these need to be installed and executed (currently universAAL modules can run either on OSGi or Android Containers, and adaptable to plain Java). It is best practice for instance to have ontologies as container-independent modules, used as libraries by the rest of modules.

Component model: Select which uAAL managers (platform modules offering basic features) each application module will make use of. In particular, for applications that have a connection to hardware devices, it is necessary to determine which LDDI Exporter Manager to use, or if a new one has to be developed. LDDI Exporters are modules devoted to translate between universAAL and the actual devices, through appropriate drivers, dongles and connections. Then the abstracted devices in semantical space can be used by any application sharing the same ontology. The component model of an application will define which kind of devices can be used by which modules, and consider the compatibility issues when the LDDI Exporter is replaced (for example for devices using a different protocol).

Deployment model: Determine how and where each module is deployed, defining for each module:

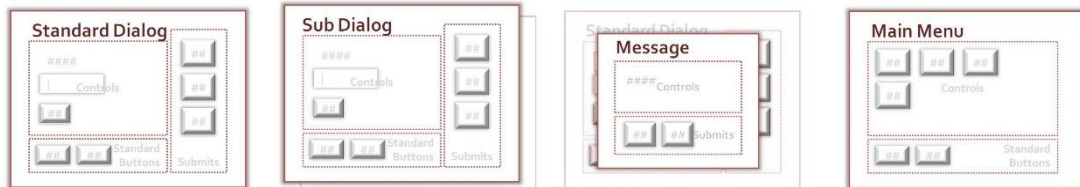
- If it is required in each node running universAAL in the AALSpace.
- If it should be only in one node in the whole AALSpace.
- If it can be optionally deployed multiple times in the same node, or in the AALSpace.
- If it is not required but an optional addition.

Deployment model will therefore define the redundancy requirements of each module.



Adaptation Schemes can display the Member, Module, Distribution and Component models

User Interaction model: For applications that have a direct interface with the user, a proper interaction model must be set up. In general, if an application handles its own UI, it would require no modification when universAALizing, and thus the adaptation is transparent for the user. But if the universAAL UI Bus is used, it is necessary to translate the user interaction model into uAAL forms.



Choose the right Form and how they link to each other

Code implementation

Once it is clear how the software will shape up, the coding process begins. As mentioned before, universAAL has two main native ways of accessing its API, based on the runtime container: in OSGi and in Android. That means that at least one of the software modules of the application (the one/those interacting directly with universAAL) will have to run in these containers. Therefore they will be an OSGi bundle or an Android app, respectively. The creation and best practices for coding these modules will differ based on this.

OSGi: It is logical that each application module being converted becomes an OSGi bundle. Bundles are the basic pieces of software in OSGi. Bundles contain an Activator that starts up the logic. The universAAL platform is comprised of several bundles in OSGi, and these can be used from other bundles to access its core API.

- The universAAL tools and documentation are oriented to managing bundles software as Maven projects. It is therefore recommended that the same is done when adapting application modules, but it is not mandatory.
- If possible, deployers should make sure that the app still works as expected when packed as a bundle and run in OSGi, before trying to universAALize the code. This may not be possible in all cases, depending on the adaptation and the original application, but is a helpful middle step to solve potential problems early.
- Once the bundle is created, create the needed code to access uAAL API and create the needed bus members, following the decisions taken when designing the various model in the previous step.

For in-depth details on how to run bundles in OSGi and connect them to uAAL, check the documentation: <https://github.com/universAAL/platform/wiki/RD-Container>.

Android: In Android, application modules are usually standalone apps (although they can also be libraries). Because of the restrictions imposed by the Android platform, the way the universAAL API can be accessed differs from the typical code writing that is performed for instance in OSGi. But first, there is a small set of additional design decisions to make.

- Choose whether to run universAAL separately, or embedded in your app. The universAAL middleware for Android is packed into a project that can be

either installed as a standalone app in Android, or embedded into another app directly. Each option has its advantages and drawbacks.

- The universAAL platform in Android, whether in a separate app or embedded, needs to be properly configured. While in OSGi this can be done with several configuration files, in Android it must be modified either at build time or at runtime through the adapted application. Something similar happens with the ontologies to be installed.

For in-depth details on how to implement apps in Android that connect to uAAL, check the documentation:

<https://github.com/universAAL/nativeandroid/wiki/Adapting-or-developing-your-app-for-uAAL> .

Remote API: In those cases where an application module that needs to interact with universAAL cannot be adapted to any of the runtime environments where the native API can be accessed, the Remote API can be used. This is a HTTP-based API that can be accessed by any software with Internet access – provided that the deployer has a universAAL instance running that publishes this API.

For information about this API check the documentation:

<https://github.com/universAAL/remote/wiki/Remote-API> .

2.4.3. Test and evaluation

It is recommended that tests are performed through the development cycle of adapting the application to uAAL (unit testing, integration testing... even test-driven development), but the adaptation process does not enforce this in any way. However, there are some features that can help with this task and are encouraged to be adopted by developers when adapting to uAAL: The universAAL platform has some plugins⁵ that enable unit testing bundles as if universAAL was running during a normal execution in OSGi.

As to what to test, here is a list of suggestions that may help identify the targets:

- As usual, check the application and its modules work as intended.
- For cases when an application is being simply adapted to universAAL, make sure the operation is the same as before adapting.
- Test the integration: that all the modules in conjunction make the overall application work as expected despite any possible re-arrangements or divisions due to the adaptation.
- Check that universAAL-based interactions are correct: that the Context Events are correctly sent and received, UI Forms are correctly rendered, and, especially, that Service Calls find their match. The latter is of special importance since the adaptation to a semantic-oriented service framework usually leads to some mistakes (more on this in the section about experience).
- In general, remember to make use of all the tools available in universAAL, such as the bus testers⁶, the log monitor⁷, or the command-line options.

⁵ <https://github.com/universAAL/itests/wiki/Integration-Tests>

⁶ <https://github.com/universAAL/samples/tree/master/integration.tests>

⁷ <https://github.com/universAAL/tools/wiki/Log-Monitor>

As a final recommendation, use the logging system (log4j⁸) thoroughly and with a good strategy. This not only helps during testing but also provides a good way to audit the operation of the application and helps solving future unexpected bugs after deployment. It is also helpful when evaluating the proper universAAL adaptation, as explained below.

Adaptation evaluation

All the above steps, since the beginning of this section, are the theory about how to compliantly adapt an existing application to universAAL. If a deployer follows the steps therein, it is almost guaranteed that the application can be compatible with universAAL (and therefore with other uAAL-compatible applications).

Once the goal is reached, it is necessary to make sure that these steps were indeed followed and everything was compliant, developed and working as expected. The starting point of such validation is a statement indicating what parts of universAAL are being used by each application. This can be done with the following form:

Usage of uAAL API

<Check-mark the applicable options (At least one. Multiple choices possible)>

- Using universAAL runtime with the Java-OSGi API of the middleware
- Using universAAL runtime with the Java-Android API of the middleware
- Using universAAL runtime with ASOR (Java from within JavaScript)
- Using other runtime with the Java Remote-API of the middleware

Usage of the middleware buses

<Check-mark the applicable options (At least one. Multiple choices possible)>

- Context bus (for publishing events and / or subscribing to events)
- Service bus (for calling and / or providing services)
- User Interaction bus (for using universAAL's UI description package and leaving the rendering to situation-aware UI handlers)

Additional middleware features used

<Check-mark the applicable options (Multiple choices possible)>

- Multi-language support
- Configurability API (mainly management of config parameters and config home directories for storing files and resources and accessing them)
- Logging mechanisms
- Multi-tenancy support (server-based usage of universAAL to connect to and serve several homes)

⁸ <http://logging.apache.org/log4j/2.x/>

- Functional Manifest (each module can contain a digitally signed “functional manifest” that is used for getting user consent – similar to the Android permissions system that gets active when you decide to install a new app, then Android lists the permissions that the app claims to need and you decide if you will install the app or not)
- The AAL Space Management API (info about the available middleware instances, installed modules, etc.)
- Serialization and parsing API (currently only for RDF Turtle syntax)

Using universAAL “Manager” components (platform services)

<Check-mark the applicable options (Multiple choices are possible)>

- Context History Entrepôt (CHE) services (querying data gathered in the home)
- Profiling services, including storage and retrieval of data that describes users, objects, locations (builds on CHE)
- Resource Manager (important only if at the middleware level, you plan to use the UI bus; in that case, you can achieve a higher level of adaptability if you let the Resource Manager store your media objects that you want to be used when interacting with the user)
- Situation Reasoner services (builds on CHE – the Situation Reasoner can store SPARQL CONSTRUCT-queries as rules to automatically generate new context events whenever certain conditions hold; this can be used to recognize situations; e.g., if you want that a context event is published whenever the user is sleeping, a solution could be to tell the SR to publish this event whenever in the night the user is in the sleeping room in the bed and the lights are off)
- The Drools Engine (a second reasoning engine using the JBoss rules)
- ASOR Scripting (ASOR stands for AAL Space Orchestrator; with ASOR scripts, you can create composite services – combinations of existing services – and define rules that specify the automatic reaction of the system to certain situations)

After validating the asserted functionality of each application, the universAAL components can be validated by checking the log files generated by the system. Each interaction with the universAAL buses or other modules leaves a particular log message (if configured to do so) that is traceable in these log files. For design decisions that are not traceable by logs, these are checked by examining the installation and deployment of the different modules of the application, any Adaptation Schemes and finally, performing user tests to check that the application indeed performs as indicated (generating the aforementioned logs in the process).

3. Coaching process in ReAAL

Coaching process in ReAAL was established with the goal of supporting application providers in their way toward successfully porting selected applications to work on top of universAAL platform.

To facilitate this process, platform experts (partners with expertise in open platforms) have been assigned to coach each pilot site in order to provide training and technical support to the application providers and pilot investors in all the matters related to the adaptation work, with a twofold aim: from one side to ensure the good quality of the porting and on the other side to widely spread the knowledge about the platform. These are also often referred to as “technical partners”.

The matching of platform experts and pilots have been done according to several factors, such as the proposed piloting concept, the geographical proximity or the shared idiom to facilitate communication. The assignments are as follows:

Pilot site	Platform expert
AJT - SL	Fh-IGD
AJT - WQZ	Fh-IGD
BRM	SINTEF
BSA	UPV
IBR	UPV
ODE	MEDCOM, (UPM)
PER	TRIALOG
PUG	CNR
RNT	SmH
TEA	UPM
Associated Pilots	TRIALOG, (UPV)

Selection of different modalities for such coaching activities have been done according to the concrete needs of each pilot site, and in case of need a backup coach have been assigned to complement the expertise.

3.1. Generic coaching process

Each pilot required specific strategies, suited to the pilot partner and technical partner and the particularities of the applications being adapted (each of which is described in the next section). But it is possible to abstract a set of generic steps that are applicable to all coaching experiences for all pilots:

Coaching was performed by technical partners having a frequent communication with pilots and their technical staff, through many different means. There are two types of events in which coaching was performed, identified by the originator of the communication:

- Coaches would arrange meetings with coached people during the “initial steps”, in order to introduce pilots to the technicalities of the platform, but

also during any generic update on the strategies of the project, like for instance a revision of used ontologies to enhance the adaptation to universAAL.

- Coached people would arrange meetings with coaches whenever a technical problem appeared during the adaptation, or when further explanations or support were needed after one of the coach-triggered meetings.

Meetings were conducted through several means:

- **Teleconferences:** These would take place between individual coaches and their assigned pilot staff, or all together (although the latter was less usual). This was the most common type of coaching sessions. The tools used were mostly Skype and GoToMeeting, often using the screen-sharing features.
- **Physical meetings:** Whenever needed, usually on demand of the coached people; in addition, coaches would take advantage of physical presence of pilots in the regular project meetings proactively, be it plenary meetings or training events.
- **Asynchronous support:** Not actual meetings, but asking for help and getting response, through emails, issue trackers or support forums, making use of collaborative documents or sharing source code.

The archetypical process by which a coach would assist a pilot to perform the adaptation to the universAAL platform usually followed these steps:

1. Get in touch with the pilot representatives, which would point the coach to the staff and developers responsible of the actual adaptation.
2. Point the developers to the universAAL documentation and have them read the basics to understand the platform. Then have them try to run the examples.
3. Understand the application and proceed to designing an adaptation scheme and strategy. In ReAAL practice, the result of this step has been documented in D3.2a.
4. At this point developers would start implementing the adaptation. At any point in time, if they found any trouble, they could ask the coaches for help. This could include:
 - a. Explain, if needed, how to turn existing code into universAALizable components (OSGi bundles/Android apps).
 - b. Assist in designing and developing ontologies if needed. Check that the ontologies designed by pilots (if any) are correct, compliant with the principles of the platform, and build on existing ontologies.
 - c. It may be necessary to assist in code development, especially with the Service Bus. To check that wrappers work as expected (received events, matching calls...)
5. Assist with any advanced manager that may be used (CHE, Remote API, Gateway...), pointing to the right documentation, providing examples or helping with the code.

6. Finally, once the pilot application is in theory adapted, the adaptation evaluation described at the end of section 2 is performed, to certify that it was correct. This was achieved mainly by task 3.4 “Field Lab Testing and Validation”.
 - a. The pilots were requested to use the checklist from section 2 and assert which components applied to the individual application. This was documented and sent to the field lab together with the equipment (if any).
 - b. Then the logs were examined as described in section 2. In the cases of BSA and RNT a logging server was setup at Smart Homes premises which allowed their cloud based applications to store a copy of their log file on the logging server. This allowed checking those logs without needing to have access to their restricted servers.

Since the pilots have no requirement to open-source their code, no direct inspection is possible and therefore the field lab tests had to resort to these indirect confirmations. In the case of design decisions, the material to analyse was the Adaptation Schemes included in Deliverable D3.2.

The whole process of all this evaluation and assessment is described in detail in Deliverable D3.5.

3.2. Pilot coaching experiences

3.2.1. AJT – SL

The Smart Living pilot in ReAAL has been coached by Fraunhofer IGD.

Performed coaching activities

An SL developer attended a universAAL training on January 16, 2014, in Barcelona. Then, two initial virtual meetings were organized, the first one for understanding the Smart Living system architecture better and the second one concerning the universAALization strategy.

In Two following face-to-face meetings, a concrete universAALization scheme was first agreed upon, as documented in D3.2a. Based on this scheme, it was obvious that SL had to make use of two new universAAL features, namely the Remote API and the scripting language ASOR (AAL Space Orchestrator). Therefore, a complementary training on these features as well as refreshments on other basic features, such as ontologies and the usage of the communication buses, were the second topic in the F2F meetings. Last but not least, the concrete ontologies in the context of the SL universAALization scheme were discussed thoroughly so that SL could start to develop them.

Monitoring the pilot adaptation

As a result of the F2F meetings, Fraunhofer IGD took over to demonstrate the universAALization of the SL services with one complete example. After delivering this example and discussing its details, the communication mode was changed from mainly meeting-based to mainly email-based in terms of continuous support, not only during the development phase, but also in the test and deployment phases.

After finishing the universAALization process at SL, a specific physical meeting was organized in order to discuss the criteria for selecting a concrete application from another pilot site as the “imported application” at SL. As a result, a ranked list of recommended alternatives was created so that SL can choose from among them.

Feedback from the pilot

- The coaching experience with SL showed that it is not always obvious how to benefit from universAAL when a simple closed solution not depending so much on third party components is going to be ported to universAAL without having any concrete plans to make use of the cross-domain data analysis and service composition potential.
- In case of SL, the flexibility in future extensions and experimentation with service composition have been identified as the main motivation for making use of universAAL.
- In addition, once more it was realized that it was very important to put the right emphasis on the design and development or selection of the right ontologies fitting the application context.
- A recommendation that can be made to the universAAL developers is to consider alleviating the complexity of the development environment in order to improve its acceptance by new developers.

3.2.2. AJT – WQZ

Concerning the WQZ pilot side, it has coincided that the technical partner and the supplier of AJT are the same (FhG).

Performed coaching activities

In this context, the main coaching activities can be summarized as follow:

- Common meeting with WQZ pilot site and its original technical supplier (inHaus) to discuss the original system architecture
- 1 day internal meeting to agree on the universAALization schema and the universAAL system architecture
- F2F discussion to extend the original ontology, thus to fit the WQZ “world” requirement: addition of the Posture and the activity ontologies, same extension of the Physical word and the AAL space ontologies.
- Support for the conception and design of the WQZ created rules as an extension of the original system
- Support of the system extension with 2 new protocols (SIP and SMTP) add to the original PLC one
- Continuous support in case of bugs during the whole lifecycle of the WQZ pilot site

Monitoring the pilot adaptation

The coaching, development and monitoring process has taken place internally between the “FhG coaches” who present the main developers of the universAAL

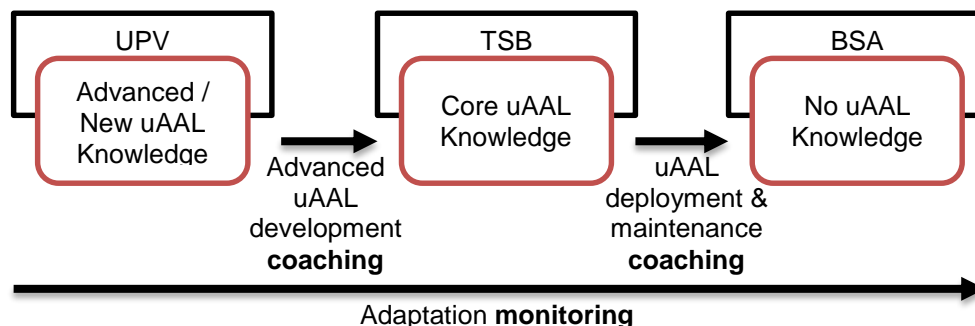
platforms and the “FhG-WQZ developers” who have universAALized the home management and the CapFloor applications.

Feedback from the pilot

- It is easier to learn about the platform while being in touch with a core member of the universAAL developer teams.
- It is very important to learn about the platform and the related component before starting any (pre)development steps
- Training is a pillar toward a successful use of the Platform and its component
- The use of the platform at the beginning needs some effort to learn the basics. In the meantime, the ability of developing applications on the top of the platform will quickly evolve... As the platform is very rich in number of features and help methods, the technical team must well learn about the platform and looks for the available features before deciding developing his owns.
- The other very important lesson learned was mainly related to the required level of development to be able to experience the Platform advantages. In fact, while developing the initial modules based on the platform in order to satisfy the minimum requirement of the services, the developer experiences very small advantages compared with the original development environment. Once the minimum set of applications is created, the service provider, same the developer experience a huge advantage in using the platform to extend, adapt, adjust and transfer the created application.
- The re-activation of the uStore and the uCC as main two components for facilitating the marketing and the deployment of universAAL based solutions

3.2.3. **BSA**

The coaching was performed by UPV to the company subcontracted by BSA, which was in charge of actually developing the applications and their adaptation to universAAL. This company (TSB) already had experience on dealing with universAAL, since they participated in the original project that developed it.



At the same time, TSB had to deal with the technical staff of BSA remotely when it came to deployment of the actual applications and their maintenance. This represents some kind of two-staged coaching, where UPV coached TSB on

advanced, specific details on universAAL (especially those developed during ReAAL, like the new features introduced by WP2), and then TSB coached BSA staff in deployment, execution and maintenance of the universAAL instances and applications.

There were several particularities and special considerations to take into account in this pilot, which made the adaptation to universAAL, and as a result the effort needed for coaching, a bit more demanding.

Performed coaching activities

As usual, coaching was performed with a combination of face-to-face and online coaching support. The first stage of coaching presented above was made easier by the fact that both UPV and TSB are familiar with at least the core of universAAL, and because they are located in the same city. The second stage was eased by having the BSA staff worry only about deployment issues, not development ones.

- Coaching content:
 - UPV > TSB: Because of the architecture of the proposed adaptation, and because TSB was not familiar with the new features introduced in the platform in ReAAL, the main topics of these coaching sessions were:
 - Adaptation to universAAL based on Android
 - Multitenancy
 - Remote-API
 - Keep-alive mechanisms
 - Advanced container deployment configuration
 - TSB > BSA: Because BSA staff main involvement was the deployment and maintenance of the solution, the main topics in which TSB coached them were:
 - Container execution and compatibility issues
 - Advanced container deployment configuration
 - Logging management
 - Android application distribution
- Face-to-face coaching:
 - UPV > TSB: Because of the proximity of both partners, it happened often that developers from one would visit the premises of the other. This has the advantage of speeding up the coaching process, being able to better share resources and respond to feedback. But there is the drawback of schedule incompatibilities.
 - TSB > BSA: More than 350km separate TSB and BSA, so holding physical meetings would be challenging. The strategy was to take advantage of other physical meetings, such as plenaries and workshops, to support the online coaching meetings. Also, during dedicated physical meetings TSB staff moved to Badalona to assist

with the deployment of successive versions and special technical support. They would take advantage of these meetings to coach BSA staff in “out-of-band” topics, such as deployment tips, that were not the main subject of the meeting.

- Online coaching:
 - UPV > TSB: There was plenty of online training material that was given to TSB for all the new topics they should be coached in (presentations, wikis, code samples...). In addition to this “passive” coaching, telcos and email communications were the main and continuous way of providing coaching sessions online.
 - TSB > BSA: Just like from UPV to TSB, telcos and email were the main communication channel, in addition to direct phone calls for special cases and staff. Regarding “passive” material, TSB elaborated dedicated documents for BSA for some of the topics described before.

Monitoring the pilot adaptation

After supervising the first design steps of the adaptation process and elaborating the Adaptation Scheme, it was clear that this would be first pilot to attempt the combination of universAAL on Android devices in the user side, connecting to universAAL in the server side through the Remote API. This means it was also the first to discover bugs in the operation of the systems involved. It must be noted that the Remote API was created by request of the ReAAL project and had not been tested before. As these bugs were discovered during the development of the adaptation of the pilot applications, they would be solved by WP2 as soon as possible.

But a consequence of the above is the divergence in versions and even codebase (for the universAAL Android App) in the pilot against the official trunk. Bug fixes can only be promptly provided through SNAPSHOT versions, but SNAPSHOTS are prone to introduce failures and should not be used in deployment. Nevertheless, the pilot had to use early SNAPSHOTS to get rid of these bugs that would otherwise block their operation.

In general, ontology design has to be careful due to being like the “API” in universAAL, which means that critical changes in the ontologies may require changes in the apps using them. In this pilot ontologies, this was particularly crucial given the imposed inability of updating one of the applications (NOMHAD) once it was deployed at end-users’ homes. This adapted application has to remain the same regardless of released versions of the platform due to a limitation in the remote-updating functionality of the original commercial application – no changes can be made after it was initially deployed unless technical support people goes to each end-users’ homes to do it manually, which is not scalable, of course.

During the step of adapting application modules to each container, the particularities of the Android version of the middleware made it at first more difficult for this task. But not because of universAAL itself: Even if adapting an Android app can be a tricky task, some Android apps being used in this pilot were already adapted or developed precisely by universAAL project: the Help When Outdoor and the Oximeter/BloodPressure drivers of NOMHAD.

In the case of the Agenda application, it was developed by TSB from scratch for ReAAL, designed from the ground up to be universAAL-based. The fact that it is a new application increased the number of bugs encountered during its development and deployment (as usual with new software) which cannot be, when discovered, identified as being caused by either the application logic or universAAL platform.

In all cases, the adaptation of the Android apps was assisted by UPV in order to speed up the process – especially taking into account that UPV created the Android version of universAAL. However, this in turn was detrimental to the coaching of the pilot in that the partners did not undergo the same level of self-learning other pilots went when dealing with the Android version.

Another obstacle deploying in Android is the particularities of the OS itself, regarding libraries, cloud messaging (used by Remote API), packaging, publication or updates. The Android version of universAAL was initially designed to be an independent application that works as a hub for the apps in the device. But to facilitate the deployment and try to avoid some of the obstacles described before (especially the updates), it was turned into a library so it could be embedded into one of the applications (the Agenda in particular). This is something that will benefit any future Android application that has to be adapted to uAAL, providing more options for adaptation. It also allowed BSA apps to follow a more modular approach that improved account handling as a result. This issue could also be considered part of the pilot feedback section below.

Feedback from the pilot

There were a set of problems because of the particular architecture of this pilot that other pilots may have not experienced. When adapting some services without having a serious intention of making complex (or just simple) data exchanges between the applications, you don't have to deal with many of the problems found here. Although the interoperability is not too big, it seems that it has implied a non-linear increment of the complexity of the applications.

Although the deployers (in this case TSB for the development and BSA for the deployment) started the project with a good perspective of the quality of the version of the platform, they report having underestimated the impact of working with universAAL in some early steps, namely:

- Knowing the new versions of the universAAL platform developed in ReAAL.
- Working with the less known Android version of the middleware
- The fact that the platform is not just only the middleware and managers but also a complex development and deployment environment
- The fact that this environment has to be fine-tuned in order to work in a large deployment scenario and not just ad-hoc set-ups

Then, during the development of the adaptation itself, a set of issues, obstacles and problems were encountered, feedback for which is being reported here:

- The platform needs to be friendlier for developers and support their daily work. TSB was close to exhaust resources due to the estimation ignorance of working with (not in) universAAL.
- Working with the uAAL middleware Android app is complicated. A lot of time was wasted chasing simple bugs due to typos or missing parameters that the

Android uAALization process requires to be very strict, and gives no clue about when failing. Some editor tools for this would have been helpful.

- Given the complex infrastructure, which involves Android devices, servers, databases, etc. it is difficult for WP2 support tasks to reproduce errors, and be able to identify bugs.
- Due to the common issue of Android fragmentation, the uAAL Android app works in some devices and not in others. These had to be identified.
- The multitenancy feature of the Android version of the platform requires Google Cloud Messaging. This is yet another system to be managed by the deployers.
- The possibility of including the uAAL middleware as a library in Android apps rather than a separate app (as described in previous section) should have been provided by universAAL from the start, not as a result of the problems it caused to this pilot.
- Overall, most of these problems could be avoided by providing good tools for developers so they don't feel left alone with a big number of disparate technologies.

3.2.4. **BRM**

The coaching of Baerum pilot site was performed by SINTEF. Before withdrawing from the project, BRM pilot site decided to import two of the applications from BSA, Help when Outdoor and Agenda, changing them to be tailored to BRM context and complement it with an additional application for monitoring users at home.

As the application provider was the same as in BSA case, the technical coaching that SINTEF performed was in supporting the pilot in defining the needed changes required to tailor the applications to the piloting context as well as in the requirements of the technical infrastructure needed to set up the deployment environment.

3.2.5. **IBR**

UPV coached directly the developers and responsible people of Ibermática, without major particularities. The physical distance between partners was not much relevant because the setup was considerably less intricate than the other UPV-supported site (BSA). Therefore, the coaching was performed primarily through online methods, but also taking advantage of pre-scheduled face-to-face meetings (plenaries, workshops) for occasional coaching.

Performed coaching activities

As mentioned before, the majority of coaching activities were performed online consisting mainly of:

- Skype-based telcos: For direct communication and coaching sessions, monitoring, and problem resolutions.
- Email discussions: For asynchronous versions of the above.

- Issue-tracking: Some of the issues encountered by Ibermática were reported through the issue-tracking system, were other technical partners could help in addition to UPV (for instance http://forge.universaal.org/gf/project/support/tracker/?action=TrackerItemEdit&tracker_item_id=440).
- Diagram-sharing: For the design phases of the adaptation, Enterprise Architect and other diagram-like tools were used to share the ideas and proposed designs.
- Code-sharing: Either sending directly the packaged code (due to pilot applications not being open source) or linking to the open source (in case of universAAL code in repositories).
 - UPV > IBR: Code examples on how to use uAAL, Code samples and snippets to be reused by Ibermática, corrections on Ibermática code.
 - IBR > UPV: Code samples and snippets of their applications being suggested or requiring to be debugged when a problem is found.

Monitoring the pilot adaptation

Ibermática adapted to universAAL some applications that originally ran in OSGi and Android. The adaptation was followed according to the steps described in section 2, with special emphasis on the following topics:

- A general coaching session about how universAAL works was introduced to Ibermática, focusing in particular on how applications can be made to run in the same environment (both the OSGi and Android options in their case) and interact with the buses.
- How to design ontologies – in particular studying which exact ontologies to use, and how to extend them. As a consequence, Ibermática identified existing ontologies that could be reused, but required minor extension to cover all the data model of one of the applications. Developers were assisted when needed when creating this extension.
- Since the device used at the client-side (Assisted Person's home) was a Raspberry Pi, Ibermática needed particular support for running OSGi and universAAL in its Operative System. It had always been perfectly possible to run universAAL in such a device, but a set of configuration options are required for it to work properly. The feedback from this issue helped refine the universAAL documentation on this topic, which will help future developers.
- Another of the Ibermática applications runs in Android, so the same support was needed for running the Android version of universAAL. As described in the experience with BSA, adapting the application to universAAL in this context is a bit trickier, so help was provided to adapt the code. For this, UPV had the advantage of dealing with a more complicated adaptation in BSA, so some of the feedback from that pilot helped shortening support on Ibermática side. It also helped that it was UPV the partner that had developed the Android version of universAAL.
- Ibermática uses its own sensor network, for which a new LDDI exporter had to be developed. UPV coached Ibermática on the techniques on developing

such exporters and provided examples to follow from existing exporters of other technologies.

- After the initial phases of adaptation to universAAL, when it was requested that pilots should go through it looking to enhance such adaptation, some discussions took place to agree on possibilities for improvements, but determined that with the introduction of their application running in Android there was enough usage of universAAL features. The way Ibermática addressed the usage of the existing ontologies also proved that universAAL was being used as intended.
- Later, during the process of determining imported applications, the technicalities of importing were discussed and a set of applications was determined that would be compatible with the existing pilot infrastructure.

Feedback from the pilot

In some aspects, Ibermática pilot was the ideal starting point for universAAL adaptation, since it was already running on supported environments (OSGi and Android). This helped in many ways during development, and made Ibermática a pilot with very few blocking issues. But still there were some major topics where the adaptation process could have improved:

- Additional support was needed to run universAAL in the Raspberry Pi, since the distribution packages available were not prepared to run in it “out of the box”. Specialized distribution packages would be helpful for common devices such as the Raspberry Pi.
- Adaptation of Android apps to universAAL is cumbersome and should have special tools to help in this process.
- Documentation on developing LDDI exporters should be more explicit and perhaps be oriented to teach by example. So far it mostly contains descriptions of existing LDDI Exporters (e.g. ZigBee, KNX...) but there are no generic instructions on developing exporters for other technologies, only generic design decisions that led to the existing exporters.
- It was necessary in some cases to share “diagram-like” information. A tool would have been useful – instead the discussions relied on screen sharing and using whatever tool at hand.

3.2.6. **ODE**

Coaching for ODE pilot have been performed by MedCom, however, as it had no previous experience with universAAL, UPM has been assisting in the more complex topics, such as the ontology design, what has been translated in a coaching more difficult from a managerial point of view.

Performed coaching activities

An on-site two days training event was organized and carried out by universAAL experts, to support the application providers in setting up the development environments and understanding the basics of the platform.

From that moment on, the coaching was performed according to the usual methods described in the beginning of the section. Whenever a more complex issue has

arisen that the main coach was not able to provide support, the backup coach was involved to speed up the solving process. In particular, additional support has been requested to setup universAAL server and environment, and showing how to use the platform without using Eclipse, as well as modelling ontologies for the pilot applications.

Monitoring the pilot adaptation

ODE managed to achieve adaptation by developing preliminary ontologies, however after the expert coaches reviewed these ontologies many changes were proposed. Preliminary ontologies were very technical, falling the category of data models rather than full semantic framework for the system. Common mistakes included the direct translation of interfaces to ontological concepts, this method is not always the most advisable, since normally interfaces represent a very specific interaction between two components rather than the description of the interaction itself or the context in which the interaction takes place. One way to pick up the ontologies where not correct was the presence of strings encoded in an object serialization standard (in this case JSON) which is indicative that the objects contained could be modelled by the ontology itself. Another give away was the use of fixed length arrays, where each index had different meaning, this is not an ontological model as it cannot assign different meaning to each index; the correct way to do this would be to model independently each index as its own concept and then grouping them in another container concept (if needed). Typical mistakes also included the disregard of existing ontologies that could help develop a more complete modelling of the solution.

ODE was always opened for suggestions and changes, in fact the new ontologies were developed using all experts' suggestions; they modelled two types of step counters (something that was challenging, even for experts), and included existing ontologies such as the device or health profile ontologies. The new model helped developers and managers in the ODE pilot appreciate the value of universAAL, as their system is now compatible with other applications. The new model made the use of the platform far easier, as the lower level details were hidden on top level business logic.

ODE development was very proactive, and required little assistance in the implementation phase. Probably due to previous experience, but fundamentally due to the new ontological model.

Feedback from the pilot

The most important lesson learned is that coaching needs to be done by a universAAL expert, otherwise common mistakes and pitfalls translate into the final product.

3.2.7. PER

The case of the PERCHE is quite specific compared to the others. Its late arrival prevents them to propose the same process (a complete universAALization of existing application) but a new approach based on the "imported App" concept. Therefore, the goal of PERCHE was not directly to universAALize existing Apps but to study how imported universAALized components (and Apps) can be integrated into an existing AAL service.

TRIALOG, the pilot leader, has a strong experience with AAL applications development (for instance they were the technical leader of the FP6 MonAMI) and also with universAAL as developers of AAL Application. Therefore, the analysis performed in July on the Agenda and HWO applications design has concluded to decouple and re-implement the User management and interface part (independent on universAAL) and to keep as it is the universAAL architecture of the App. After a first period of deployment and based on the user feedbacks, it is possible that Trialog in collaboration with his subcontractor will go deeper into universAAL stuff to extend the current functionalities of the Apps.

3.2.8. PUG

The Italian pilot site is hosted in the Puglia region whose headquarter is located in Bari⁹. This pilot is considerably complex for tree main reasons:

1. The process followed by Puglia region to recruit users and companies is achieved with public procurements.
2. The high number of companies involved.
3. The high number of AAL services provided by the companies.

In particular, the Puglia region decided to recruit users and companies via open procurements. This process guarantees a deeper penetration of the universAAL platform within the Italian market of AAL services. In fact, the open call guarantees that:

- Most of regional companies become aware of the existence of an open platform targeted to AAL services;
- Companies interested to participate to the initiative are evaluated according to some criteria.

The process for recruiting companies involved four steps:



The procurement has been published in 2014 and, at the end of the process, 6 companies answered positively to the public call. Such companies now join the ReAAL consortium as associate vendors. The companies are:

- Ingel
- SteelMinds
- Virtech
- Cupersafety
- eResults
- BioResult

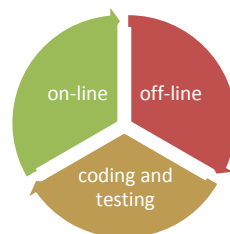
⁹ 41.108811, 16.915067

Moreover, the services (or modules) developed by the companies have been clustered in three main areas of interest: Safety-at-Home, Home Activity Monitoring and Easy Home Control. The following table reports the name of the modules developed by each of the companies:

Area of interest	Module	Company
Safety-at-home	Safehome	Ingel
	Electrosafe	Steel minds
	iHelp	Virtech
Home Activity Monitoring	Indoor Monitoring System	ERESULT SRL
	Environmental Monitoring System	ERESULT SRL
	Omniacare Health Check	BIORESULT SRL
Easy Home Control	NewDom	CUPERSAFETY SRL

Performed coaching activities

The number of companies involved as well as the number of modules developed by the companies led CNR-ISTI (as coacher of the Italian pilot) to approach the coaching activities with a structured approach. The coaching is split in three phases closed in a loop:



- Off-line training: during this phase the companies started learning the universAAL basic concepts including the available tutorials, examples and the API documentation. Such resources have been mostly prepared in the context of the universAAL project. CNR-ISTI collected in one single document a step-by-step guide to browse the universAAL documentation¹⁰. In particular, the document refers to the following main resources:
 - The Developer Depot: CNR-ISTI recommended specific documents to read such as universAAL primer basic and advanced;
 - The GForge platform: CNR-ISTI explained the organization of the universAAL projects (middleware, ontologies, LDDI, support etc.) and how to join such sub-projects (as shown in the next figures). CNR-ISTI also suggested specific wiki pages to read as well as the tutorials already available.

¹⁰ The document has been written in Italian language.

- Slide-sets prepared for the Italian companies summarizing the core concepts of the universAAL platform such as: use of the platform, basic tutorials, use of the ontologies and some examples of adaptations.
- On-line training: during this step CNR-ISTI prepared a simple mechanism to allow the companies to ask for questions, suggestions and requesting for live seminars. In particular, the following mechanism was adopted:
 - A company first opens a ticket on the Italian tracker¹¹ hosted on the GForge platform. The tracker collects the tickets only from the Italian companies. For every new ticket, CNR-ISTI receives an email notification so that it is possible to understand the kind of question, get ready with the proper documentation and (if needed) to group together different tickets concerning the same topic.
 - After the reception of the email notification CNR-ISTI answers to the companies posting a message on tracker or, if necessary, via skype calls. This last option (Skype calls) has been the most useful and practical mechanism to provide support to companies.
 - In addition, CNR-ISTI has prepared two live seminars concerning the some basic concepts of the universAAL platform as well as some live demos to better explain how to use the universAAL API.
- Coding and testing: during this step the deployers code their modules so that to adapt to the universAAL platform.

After such training steps, each company had still the possibility of requesting for further support to the universAAL development community.

During the coaching activities 5 tickets were opened as shown in the figure below, asking for different level of support:

- Email exchange
- Skype calls
- Coding of specific examples
- On line seminars

ID	Sommario	Priorità
445	[Italy Pilot Sites - SteelMinds] Training: Context History Entrepot	3
446	[Italy Pilot Sites - SteelMinds] Training: Reasoner Application	3
447	[Italy Pilot Sites - SteelMinds] Support request for ontology writing	3
452	[Italy Pilot Sites - SteelMinds] Training: Application Ontology and uAALUtils library use	3
453	[Italy Pilot Sites - SteelMinds] Training Request for Application Ontology and Service Profiles	3

Monitoring the pilot adaptation

¹¹ http://forge.universaal.org/gf/project/support/tracker/?action=TrackerItemBrowse&tracker_id=261

CNR-ISTI evaluated the progresses of the deployers along with the universAALization with two methodologies (ranked according to the effectiveness):

- Scheduled telcos: this has been the primary way to understand troubles and misunderstandings of the companies. The approach followed consisted in asking to deployers specific questions, for example:
 - (Concerning the platform)
 - If you want to run universAAL starting automatically specific bundles how would you modify the Karaf feature?
 - How to run universAAL in DEBUG mode?
 - How to verify if context messages generated by your application are correctly sent out?
- Validation and analysis of the adaptation schema proposed: CNR-ISTI reviewed the adaptation schema proposed by deployers. In particular:
 - CNR-ISTI evaluated qualitatively the integration with the core universAAL features (e.g. use of the context-bus, use of the service-bus, use of the AAL Space concept, ontologies used);
 - CNR-ISTI suggested, when needed, changes to the adaptation schema. Also tried to involve expert coaches of a specific topic to support the Italian companies.

Feedback from the pilot

From the coaching activities it is possible to draw some conclusions and lessons learned:

- Deployers who are approaching to a new platform (such as universAAL) require to be coached with an off-line phase during which they can read and test the platform alone. This phase allows to the deployers to become familiar with the universAAL APIs and terminology to be used and to try to figure out how their applications can be adapted without any specific suggestion. It is considered that this effort has the advantage of pushing the deployers to really understand the basic concepts of universAAL.
- On line seminars and Skype calls are the most effective tools for answering to questions from companies. Email exchange are helpful only if they refer to specific and simple troubles. Moreover, the email exchange is not a *debugging* tools, this means that coaches should avoid to write long emails, rather they should set up a quick telco.
- The quality of the documentation is the key-factor for the success of the universAALization.
- Deployers are often sceptic in using the ontologies as an interoperable language. At the beginning of the training, deployers asked several times the potentialities of such language to their products.
- Coaches should highlight the benefits of the whole platforms but, more importantly, also the benefits of single components of the platform. As for example, even if a company does not consider fruitful to adopt the Karaf

OSGi runtime within their development process, they might consider only to use the busses as way to exchange context information at home. Under this respect, the coacher plays a crucial role for the promotion of the universAAL platform.

3.2.9. RNT

Smart Homes (SmH) was RijnmondNet's (RNT) coach for this project. RNT is not a technical party itself and were not able to understand the technicalities of the platform. The lack of technical personnel was solved by inviting the supplier Almende to the coaching meetings between SmH and RNT and also to regular project meetings, e.g. during the Paris workshop. Although none of the six suppliers had any experience with universAAL from the beginning of the project, Almende took the lead in trying to understand universAAL and was able to set up a test environment and getting familiar with universAAL. The other suppliers were:

- MedicineMen
- Netmedical
- Mibida
- MindDistrict
- Curavista

Performed coaching activities

During the adaptation phase Smart Homes organised regular physical and virtual meetings informing each partner on any new technicality from WP2 technical meetings, discussing blocking issues and sharing experiences, which Almende experienced using universAAL

The feedback from these meetings were reflected back to the consortium through SmH in the technical workshops and technical teleconferences that were organized by WP2 lead.

This resulted for example in better, understandable online documentation and a quick-start guide for every new developer who wants to properly set up a universAAL developing environment. At a later stage of the project, the internet forum of the platform was used increasingly to ask questions by RNT's suppliers. The questions posted were directly answered by the main developers of universAAL, which in most cases are members of WP2. At a later stage, as more serious errors and bugs were found in the applications of RNT the internet forum was considered overhead. Questions were asked directly to the main developers of the platform by e-mail.

Monitoring the pilot adaptation

The corrective actions that were assigned to RNT by the management of the project made a more intense appeal to SmH as coach. During this period a strict time-schedule was created and a document with 4 milestones. Due to the strict time limitations RNT and SmH had at least weekly teleconference meetings by phone in order to make sure all milestones were reached in time and with good quality. SmH was responsible to setup meetings for discussing the ontologies with all suppliers

amongst others. Every week a fixed timeslot was allocated for an open question round between suppliers and developers.

Feedback from the pilot

The forum of the platform is vital for knowledge sharing across pilots within ReAAL but also across the world. In a lot of cases this was seen as overhead by the RNT suppliers and the path of e-mailing directly from supplier to developer was used as it was found quicker and easier. Especially when the deadline of releasing the application came near. This meant, in most cases, that SmH as coach was not involved in heavy discussions over any issues or bugs that occurred as SmH were not listed message address list. Although this was addressed by SmH, to the suppliers, it was frequently forgotten.

RNT, SmH and all six suppliers had no experience in/with the universAAL platform. Almende and SmH started to investigate the possibilities of the platform when the project started in 2013. The task of SmH was however not to adopt any applications to the universAAL platform (so called universalization process) and therefore SmH never gained any in-depth practical knowledge that was needed for detailed coaching of all six suppliers

Almende quickly gained hands-on knowledge as they were in fact implementing universAAL in their application. As Almende was the first to do so of the six suppliers, they could more easily help the rest of the suppliers with the start of their universAALization process. Any questions, issues and bugs from the other suppliers during the universAALization were first tried to be solved internally by Almende.

Any other questions that were too difficult for Almende to solve were passed on to the main universAAL developers. The role of SmH as coach changed gradually from a technical knowledgeable role, towards more organizational role, e.g. keeping track of deliverables that needed technical input from the suppliers, etc.

3.2.10. **TEA**

Performed coaching activities

UPM was the assigned coach for TEA pilot site. Being located in the same city made it easy to follow up on the progress and profit from physical meetings to provide a more effective support. Although most of the coaching was done by UPM some inquiries were derived to UPV. These questions were due to the usage of the Android universAAL container by TEA's system, and the expert on this area being associated with UPV.

In the case of TEA, the coaching proceeded without major issues according to the usual steps described at the beginning of the section. After the initial phases of adaptation to universAAL, suggestions for improvement were discussed and adopted in the context of the importing application process.

Most of the coaching activities revolved around trying to solve minor issues, misconceptions and misunderstandings that arose during the training and co-design sessions.

For example the first misconception, is about security. Without careful explanation during the introductory sessions of the platform, trainees understood that the buses of the platform operate at universal scale (probably confused by the name of the platform). Privacy and security concerns, under this false assumption, took them

back from counting on the platform features for many application areas (especially those sensitive to privacy issues). Thankfully this misconception was resolved early on.

Another difficulty was conveying that platform can be used as a tool for developing features within application (by using the platform to connect the modules of the application), and not just between applications. This concept of using universAAL as base for the system architecture of the application will also have useful consequences, for example:

- It will help optimize the design of the application itself by being able to use the features of the platform
- Inner links are “exposed” for other applications to use and exploit the first application to different degrees.

Monitoring the pilot adaptation

TEA developers had some experience with ontology design, which helped a lot while explaining the platform and especially during the discussions of the ontology design. This background enabled trainees to grasp quickly many complex concepts, that otherwise would have been difficult to be included in the final adaption. This understanding together with good disposition helped the monitoring process, so much so that coaching services were hardly required, and the coacher had the confidence that more detailed monitoring was unnecessary.

Yet during adaptation minor misunderstandings still appeared, for example, instead of reusing existing ontology concepts, by adding properties, TEA developers had made a copy to customize of these concepts, effectively duplicating concepts (and code) unnecessarily. Thankfully, the semantic features of mapping ontological concepts can be used to unite once again the duplicated concepts, without having to recode the ontologies, which would have consumed much of the already scarce resources.

Feedback from the pilot

- During Training it is critical to convey a broader understanding of the platform. universAAL is very complete, complex and some trainees will pick only the key concepts they feel the platform can be used for, and disregard the rest of the properties and components of the platform. The name alone of the platform will lead to preconceptions of the trainees. The coacher must be aware of these preconceptions, and try to debunk them as soon as possible when identified on trainee’s train of thought.
- When dealing with developers that have a good understanding of how to model ontologies, it is easier to convey all the semantic functionalities of the platform. It is particularly useful to explain the intricacies of how services may or may not be matched, and how to use ontological modelling to the advantage of the design of the application.
- The coacher has to know when to compromise on design concepts proposed by trainees. Even if the coacher believes there is a better design concept, other considerations like effort, time, human resources devoted to the adaptation should be accounted.

- Sometimes the good understanding of the trainee leads to misunderstandings. Even a trainee shows the best disposition and knowledge the trainer must not disregard basic concepts and keep track of the progress.
- It is also important to keep a look on the updates on the development, especially during the ontology implementation, as ontologies are the basis of the whole universAAL platform, their optimization is critical to the success of the universAALization and for the capability of implementing the different showcases correctly.
- Generally, the most efficient way to coach a team which requires platform knowledge is using continuous communication.
- Getting in touch with the experts of each topic, once a minimal knowledge is acquired, speeds the adaptation; while making the learning curve a bit more accessible.
- When the coacher is open to questions and participation on the application design, in most cases, helps trainees to get insight of the process of how to get the most advantage of the platform; with the additional advantage of getting a real-life example, and expert advice on design.

3.2.11. Associated Pilots

In February 2015, four Associated Pilots joined the ReAAL project with the goal of complementing and extending the experience of the member pilots and providing additional users. For them, TRIALOG is acting as their coaching partner, backed up by UPV.

The universAALization process has been active from March to July for 3 pilots (IMA, SCUPS, and NCSR) and until October for EIC-IL.

Two full coaching days for the associated pilots and associated vendors were organized in Eindhoven by end of February 2015, where all the associated pilots participated and therefore were able to start the design process of their universAALized applications. Raw videos with the theoretical approach are here:

<https://cloud.igd.fraunhofer.de/owncloud/index.php/apps/files/?dir=%2FuniversAAL%20Training>

The project **members** coaching them (mainly Trialog and UPV for the universAAL expertise) guided them through a step-by-step process inspired by the previous WP1 and WP3 works. First an overall template architecture document has been proposed. This template is based on the interoperability requirement deliverable D1.2 and the D3.2a. The architecture of each application is described per software components and per distributed nodes when required. Within this architecture, universAAL middleware is inserted where functional decoupling is suggested by the designer as well as a first description of the data involved. After discussing about the reasons of this decoupling, pilots started to work on the ontology. As for other pilots, it has been proved that the choice of the ontologies (reusing an existing one or creating a new one) is one the most important decision to ensure a reasonable adaptation.

4. Generic lessons learned

As it can be seen from section 3.2, the coaching experiences are as varied as the pilots themselves. Just like the coaching process can be abstracted to a generic set of common steps applicable to everyone, there are some issues that have been encountered across several pilots. These identify aspects that should be improved in general by the universAAL platform to facilitate adaptation in the future, regardless of particular specific issues that will always arise in any adaptation process.

- Developers approaching to a new platform such as universAAL need first an off-line phase during which they can learn about the platform and the related component before starting any (pre)development steps. The quality of the documentation is key at this stage to keep the motivation of the developer.
- As the platform is very rich in number of features and help methods, the technical team must well learn about the platform and looks for the available features before deciding developing his owns
- The learning curve is steep at the beginning and need some effort to learn the basics. However, the ability of developing applications on the top of the platform quickly evolve with the practice.
- Training and coaching are key activities to ensure the success in adapting/developing universAAL applications, but they need to be done by a universAAL expert, otherwise common mistakes and pitfalls translate into the final product.
- The most efficient way to coach a team that requires platform knowledge is using continuous communication. The forum of the platform is vital for knowledge sharing but on-line seminars and Skype calls are most effective tools for answering to questions from developers.
- There is the need to ease/customize the universAAL applications development environment to maximize its acceptance from the developer side. Developer support tools for ontology design, test and debugging, as well as for configuration, deployment and maintenance are crucial for wide adoption of the platform.
- Main topics for support has been requested to setup universAAL server and environment, and showing how to use the platform without using Eclipse, as well as modelling ontologies for the pilot applications.
- It is very important to study which exact ontologies to use, and how to extend them, to achieve an expressive and generic ontology that fit the pilot/application needs, before starting any implementation activity.
- For the coaching team, it is important to keep a look on the updates during development, especially during the ontology implementation, their optimization is key to the success of the universAALization and for the capability of implementing the desired level of interoperability.
- Typical mistakes have included the disregard of existing ontologies that could help develop a more complete modelling of the solution as well as the definition of very technical ontologies, falling in the category of data models

rather than full semantic framework for the system. Common mistakes included the direct translation of interfaces to ontological concepts.

4.1. Frequently asked questions

Here are some of the recurring doubts that the pilots have come across. They are not issues with the platform itself, but very specific topics that the coaches had to explain often. These have been brought up to the attention of the coaches or other support members through many sources: Support forums, Issue Trackers, Developers mailing list, direct contact with coaches... (Not counting bugs or feature requests).

The intention here is not to answer the questions themselves but how they were answered and why the issues they represent were found so often. The questions are sorted by frequency (more frequent first) but in a very loose way: it is difficult to determine actual amount of occurrences given all the different sources of the questions, and the fact that some may be related to others, or from different sources.

How can I migrate existing ontologies to universAAL (and back)?

There were many requests for explaining how to move from existing OWL files (files for defining ontologies) to universAAL-compatible ontology representations (OSGi bundles), and also the other way around. This is a topic of interest because in the cases where developers already know ontologies, they usually have some existing OWL files they would like to use. And another reason for the question to pop up is that the process that allows this is a bit convoluted, requiring separate steps with some of the tools of the AAL Studio plugins – whose documentation was hard to find and follow. In addition, there were a few bugs, too.

The actions taken to minimize these issues were to update and fix the tools, rework the documentation, and lead to the right links in the wikis. However, since a full refactor of the migration tool has been planned (whenever there is time), the usual recommendation has been to implement the code of the ontology manually – taking the chance to refine the ontology. Which leads to the next frequent question.

Example forum link: http://forge.universaal.org/gf/project/support/forum/?forum_action=ForumMessageBrowse&thread_id=120&action=ForumBrowse&forum_id=7

How do I model a new ontology?

Ontologies are one of the most common entry barriers for using universAAL, as they require changing the way developers think about data models. This is why developers have asked for support on how to design the ontologies they need and implement them in universAAL-compatible bundles.

Regarding design, it is only possible to assist developers on a per-case basis, and give general advices (some are found in this deliverable, for instance). Regarding implementation, developers were redirected to appropriate documentation links.

Example tracker link: http://forge.universaal.org/gf/project/ontologies/forum/?action=ForumBrowse&forum_id=126

Where can I find an example and how can I run it?

Some developers expect to find an example that is ready to download and run. This is a common practice with all libraries, platforms and open source projects. In

universAAL there is the “Lighting Example”, but documentation is often focused on code development, not on how to simply run it.

New documentation pages were created that explain the minimum steps needed to get a universAAL instance running, and how to add modules (such as the lighting example) into it.

Example forum link: http://forge.universaal.org/qf/project/support/forum/?forum_action=ForumMessageBrowse&thread_id=135&action=ForumBrowse&forum_id=8

Where can I find the module [X] and how do I use it?

There were several questions asking for information about specific modules (managers like CHE or Gateway, ontologies, examples...). These are usually solved by redirecting developers to the right wiki page that explains the module. After the rework of the documentation pages this should be easier to find by developers themselves.

Where do I start?

In some cases new developers do not know exactly where to start reading to know how to use universAAL. This is part of the ongoing trouble with documentation clarity. After the rework of the documentation wiki pages, they are usually redirected to the GitHub Platform repository Wiki page, which is now highlighted in all places as the main entry point for technical documentation.

How do I prepare my own deployable instance of universAAL with my applications?

The final step in any deployment of the pilots is to actually deploy the universAALized applications. In this case it means preparing a Karaf folder with every needed bundle already installed, or, in case of deploying on Android, packaging the APKs.

There are already some documentation pages dedicated to these issues, but since every pilot is considerably different from each other, some required more specialized instructions, case by case.

How do I convert my existing application code to something I can run in universAAL? And how do I deploy it?

This is part of the implementation phase of the universAALization process, explained in this deliverable too. It is part of the coaching tasks to assist pilot developers with this. Again, because existing code varies so much from pilot to pilot, there is no generic answer.

Example forum link: http://forge.universaal.org/qf/project/support/forum/?forum_action=ForumMessageBrowse&thread_id=148&action=ForumBrowse&forum_id=7

What hardware sensor technologies does universAAL support?

One for the aspects of universAAL that raises more attention from developers is its support for sensor networks and domotic devices. Each hardware technology requires an adaptation (an “Exporter”), and in the universAAL platform there are already some existing exporters, listed and explained in the LDDI repository documentation.

Example forum link: http://forge.universaal.org/gf/project/support/forum/?_forum_action=ForumMessageBrowse&thread_id=125&action=ForumBrowse&forum_id=7

Can I run universAAL in this particular execution environment or device?

One of the questions, described in this document, that need to be answered when planning the universAALization of applications is if it is actually possible to run universAAL in whatever device will be used for deployment. The universAAL platform can run in most JVM, so any device with a running environment capable of this should be able to run universAAL (including Android). Of course, there are particularities with every environment and these have to be solved one by one.

How do I use multi-tenancy to connect universAAL instances remotely?

One of the features that were added to the platform specifically for ReAAL was multi-tenancy, which allows to connect or virtualize several “tenants” (or remote nodes) to a central server instance. Because it was made during ReAAL, until all documentation was ready, pilots would have to ask for support on how to make use of it.

Example forum link: http://forge.universaal.org/gf/project/support/forum/?_forum_action=ForumMessageBrowse&thread_id=163&action=ForumBrowse&forum_id=8

How do I run the Android version of universAAL?

Some pilots deploy universAAL in Android devices. Because it is quite different from the usual Karaf-OSGi deployment, it requires special care. Developers were redirected to the specialized documentation, and such documentation was also improved.

How can I make sure a universAAL service call will succeed?

Because service calls are semantic, it is often possible that calls will fail to find the service they were looking for if the call or the service were not properly defined. It is one of the most common issues found while coding universAAL applications. Coaches were available on demand when developers found these issues and helped debug and solve the problem.

Can I develop universAAL applications on Mac?

Some use Mac computers to code. In the past, there have been issues when installing the recommended IDE (Eclipse) for developing with universAAL. These were due to issues with one of the build tools (Maven), which is not the responsibility of universAAL platform. There are other ways to code the OSGi bundles (or Android APKs) that don't require this IDE and build tool, but the universAAL documentation cannot cover all possibilities. However it seems that latest versions of the IDE may no longer face this problem. As for running universAAL itself for testing, it should work just like Linux.

Example forum link: http://forge.universaal.org/gf/project/support/forum/?_forum_action=ForumMessageBrowse&thread_id=159&action=ForumBrowse&forum_id=8

5. Conclusion

One of the advantages of the universAAL platform is that it can be adapted in many different ways to many different scenarios, but in turn, it makes it difficult to define a holistic compliance methodology. There are, of course, certain “checks” that need to be certified in order to consider an application as being adapted to universAAL (universAALized), and these are reported in this deliverable. But beyond these, there is plenty of room for pilot developers to choose how exactly, and for what purpose, they incorporate universAAL into their solutions and how to benefit from it.

Just like there is no one-size-fits-all process for the adaptation to universAAL, there is no unique script that a coach can follow to give support to all pilots, given the differences between them. Experience showed that the most practical approach (and probably the best) was providing on-demand support through direct communication channels. While it is thought that face-to-face meetings (e.g. in plenaries) or workshops are fruitful (and they are), their limited duration and the effort required to attend the meetings make them an exceptional occasion, used for exceptional issues.

As a conclusion, pilots required a dedicated coaching support tailored to their specific architecture and intended usage of the platform. The compliance of their adapted solutions had to be evaluated starting from a common set of minimal requirements and continuing through each specificity of their deployment, to determine that the adaptation was correct and useful. In return, each pilot contributed with their feedback, which will be useful when establishing an open source community around the universAAL platform.