30 APRILE 2018

Evaluation of universAAL solution

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

www.NESTORE-coach.eu

DELIVERABLE ID:	WP6/D6.2/TASK NUMBER 6.2
DELIVERABLE TITLE:	Evaluation of universAAL solution
RESPONSIBLE PARTNER:	ROPARDO
CONTRIBUTORS:	Gabriela Candea, Marius Staicu, Ciprian Candea (ROPARDO), Filippo Palumbo (CNR)
NATURE	Report
DISSEMINATION LEVEL:	PU
FILE:	D62_Evaluation of universAAL solutionV1_Final
REVISION:	V1
DUE DATE OF DELIVERABLE:	2018.04.30
ACTUAL SUBMISSION DATE:	2018.04.30
CALL	European Union's Horizon 2020 Grant agreement: No 769643
TOPIC	SC1-PM-15-2017 - Personalised Medicine

Document History

REVISION	DATE	MODIFICATION	AUTHOR
0.1	2018.03.06	Document Creation, initial TOC	Gabriela Candea, Marius Staicu, Ciprian Candea (ROPARDO)
0.2	2018.06.11	Document v0.2	Gabriela Candea, Marius Staicu, Ciprian Candea (ROPARDO)
0.3	2018.06.19	Document update	Filippo Palumbo (CNR)
1.0	2018.06.28	Final version	Gabriela Candea, Ciprian Candea (ROPARDO)

Approvals

DATE	NAME	ORGANIZATION	ROLE
2018.06.25	Sandro Repetti	NEOS	Reviewer
2018.06.28	Ciprian Candea	ROPARDO	WP Leader
2018.06.28	Giuseppe Andreoni	POLIMI	Scientific Coordinator

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

Short Abstract

This document is the results of work in task T6.2 Evaluation of universAAL solution. Within this task the technological partners from NESTORE reviewed and analyses the official universAAL sources and documentation, realized a Proof-of-Concept. Based on learned lessons and intersecting them with NESTORE requirements are presented conclusions and NESTORE approach relating universAAL support.

Key Words

universAAL, Proof of Concept, technical assessment, IoT, Message Bus

Table of Contents

1. Executive summary 8

2. NESTORE requirements 10

3. universAAL: what and how? 13
3.1. Introduction 13
3.2. Middleware 14
3.3. uAAL Applications 14
3.4. Managers 15
3.5. Semantic Services 16
3.6. Implementation in UniversAAL 16
3.7. The software 17

4. NESTORE Proof of Concept 19
4.1. Hardware components 19
4.2. Software configuration 21
4.3. Development Configuration 21
4.4. uAAL Setup 22
4.5. Proof of Concept details 22
4.6. uAAL Challenges 25
4.7. Conclusions relating POC 26

5. Proposed alternative 28
5.1. NESTORE & uAAL 28
5.2. NESTORE IoT 29
5.2.1. IoT Deployment 31
5.3. NESTORE Message Bus 32
5.3.1. Using NESTORE Message Queue as uAAL buses 34
5.4. Application Integration 38

6. Conclusions 40
6.1. A) Non-Technical 40
6.2. B) Technical 41
6.3. C) NESTORE & uAAL 42

7. References 43

8. Annexes 45
8.1. Annex 1 45
8.2. Annex 2 47
8.2.1. NEOS 47
8.2.2. TUD 47
8.2.3. HES-SO 48

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

Table of Figures

Figure 1 Runtime Support Platform – layered model 14
Figure 2 Runtime Support Platform 15
Figure 3 RDF statement 16
Figure 4 An Application running in the container connects to the buses using its own wrappers 17
Figure 5 NESTORE PoC Hardware components 20
Figure 6 Software components 23
Figure 7 Main Sequence Flow 24
Figure 8 NESTORE & uAAL 28
Figure 9 NESTORE - vCare integration 29
Figure 10 A typical IoT solution 30
Figure 11 IoT Components - by Eclipse Foundation 30
Figure 12 Deployment using wotAgent 32
Figure 13 Sender - Receiver 33
Figure 14 Request/response messaging with dedicated response queues for each sender 34
Figure 15 The integration with universAAL is equivalent to “talking” to its buses with specific roles 35
Figure 16 A generic MQTT topic 36
Figure 17 Separation of concerns in different MQTT topics and their relationships with uAAL buses 36
Figure 18 Sequence diagram of the announce phase for discovering a newly deployed sensor 37
Figure 19 Sequence diagram of the interaction of the NESTORE modules with the uAAL buses 37
Figure 20 Driver offer interface with temperature sensor 45
Figure 21 Driver offer interface with relay (actuator) 45
Figure 22 Reasoner observes the “world” and can take various decisions based on its rules programming 46
Figure 23 It offers a REST/JSON standard interface over HTTP to integrate with external users 46

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 769643

List of Acronyms

AI – Artificial Intelligence

AJAX – Asynchronous JavaScript and XML

API – Application Program Interface

CORBA – Common Object Request Broker Architecture

CRUD – Create, Retrieve, Update, Delete

CSS – Content Style Sheet

HTML – Hypertext Mark-up Language

HTTP – Hypertext Transfer Protocol

ISO – International Organization for Standardization

IoT – Internet of Things

MOM – Message Oriented Middleware

MQTT - MQ Telemetry Transport

OASIS – Organization for the Advancement of Structured Information Standards

OWL –Ontology Web Language

REST – Representational State Transfer

RMI – Remote Method Invocation

RPC – Remote Procedure Call

SOA – Service Oriented Architecture

SQL – Standard Query Language

uAAL - the universAAL project

UML – Unified Modelling Language

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

W3C – World Wide Web Consortium

WSDL – Web Services Description Language

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

WWW – World Wide Web

XML – eXtensible Mark-up Language

1. Executive summary

On the T6.2, NESTORE consortium carried out the following actions:

- Analysis of the UniversAAL platform features, requirements and implementation, both at theoretical level and the development of a preliminary proof-of-concept.
- Contact the UniversalAAL IoT Coalition (uIC) for verifying the analysis and setting up the collaboration for the NESTORE platform development in accordance with the rule to have a "quality certified" and reliable system to be deployed to the end-users for the pilot. uIC has not been registered as a legal entity entitled to sign the subcontracting agreement.

NESTORE Requirements Validation against UniversAAL platform:

- UniversAAL IoT is a middleware from where each project can start building their own technical platform blended with ontologies objects. The actual state of the art of UniversAAL components present older release of Java (1.5), Karaf (2.2.9) and OSGI [web3]. These versions have some contradictions on UniversAAL wiki/Github because while the installation instructions have been update to reflect newer versions (Java 8, Eclipse 4.6) the prebuilt distribution still has Karaf 2.2.9 which does not support Java 8… "Java 8 is not supported by Karaf 2.x.y, hence the UniversAAL release might not work properly. Java 8 is supported by Karaf 3.0.0+, so select the appropriate version accordingly". No easy way to get latest running version except building from sources.
- UniversAAL communication between nodes is bounded by the multicast network; it requires a complex set up (additional software) to overcome the limited multicast support available when run over Internet (GAN – global area network). This reduce dramatically its usage on the NESTORE use cases, where the target is aged users with limited technical skills and resources.
- UniversAAL application are written as OSGi bundles. Due to this approach it is expensive to integrate existing open source modules or non-Java modules i.e. chatbot's or other AI modules/algorithms, leading to a huge impact on NESTORE development

Administrative and Legal Issues:

- UniversAAL is based on very big code base, complex architecture leading to potential for bugs & misconfigurations. The support community on GitHub is small (4 users at the date of our investigation) and the dynamic in the code support is low implying a "strong" effort of the consortium in keeping alive the platform development instead of concentrating in NESTORE objectives
- Some of the code sections are released under proprietary license. NESTORE project will be developed under the open policy of the EU funding
- NESTORE consortium can't benefit from uIC support due to the missing legal registration of the uIC. NESTORE was not allowed to receive support from one of the UniversAAL key partners, who are all part of the uIC

Proposal that are presented on the D6.2 are summarized as follow:

1. NESTORE project will define the methodology and complete the ontology of the human model for the elderly and will publish it as scientific outcome either of the NESTORE public repository or in the UniversAAL GitHub site.

2. NESTORE project will define the methodology for the realization of a UniversAAL “bridge” or Manager (as reported in the Grant Agreement) and will implement and deploy that bridge in the NESTORE platform. The ontology represents the first side of the UniversAAL Manager. Additionally, NESTORE project will define a parallel JSON structure mirroring the ontology and compatible with it and more usable on any technological architecture. This module represents the second side of the UniversAAL Manager.

3. A restart of the relationship with some members of the coalition will be tried to have specific training and a reliability assessment of the provided UniversAAL solution implement in NESTORE project in accordance with the mitigation measure no.10 in the Grant Agreement.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 769643

2. NESTORE requirements

The focus of present deliverable is the evaluation of universAAL accordingly with platform requirements described in D6.1, monitoring system requirement defined in WP2 and user requirements defined in WP7.

Trust	Privacy/security/ reliability of NESTORE
	The system's management of data is transparent. Users are aware of where data goes and how it is used. The user has control over their privacy settings (in a usable manner and who has access to the data)
	Health-related data is accurate and has the option to be viewed in real-time by users of the system (which might be more appropriate for certain health conditions)
	Users have the capacity to turn off analytics (on/off switch)
	The system will translate health-data into contextualized user-centred feedback appropriate to its audience
	The system should be robust and withstand everyday use
	The system should not compromise or effect other health-technologies of the user (e.g. pace-maker)
	Charging (power) requirements should not interfere with lifestyle
	Software updates should not alienate access.
Cost	Affordability
	NESTORE should be scalable (e.g. inclusive core features with option to purchase/add additional functionality)
	Costs should not be prohibitive to the specified user group
Fits my life	NESTORE must be user-centred and responsive to reflect the needs and preferences of the end user including considerations of ergonomics
	Interface should be clear, concise and visually appropriate with the opportunity for manual customization by the user reflecting their own preferences and style
	The visual interface (e.g. Icons) should be easily recognizable, provide consistent look and feel
	Single sign in (log in once rather than multiple times)
	The system should provide a creative user-friendly solution for log in and resource access
	Users settings should be remembered throughout the platform

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

	When the user changes the interface settings the interface should be updated immediately and continuously
	Resource/functions should be accessible and usable by all users of the system. Features should include interaction modes for touch and voice alongside high visibility settings appropriate to personal needs (e.g. high contrast for people with visual impairment etc.) and language settings/options Language should be clear and meet the needs of individuals with varied literacy skills
	The system should be responsive and adapt itself to the environment and user needs
	The system should have the capacity to be used beyond the user's physical home environment

Table 1: User needs

NESTORE platform is composed by multiple devices, software components, serious game and users that involve different communication technology, described in D6.1.

Requirement	**Wearable monitoring device**	**Environmental monitoring device**	**Decision Support System**	**Virtual Coach**	**Game**	**Tangible interface**
Internet connection	NO	Yes, environmental monitoring system sends data to the cloud continuously.	Yes, DSS runs both locally and in the cloud	YES, Virtual coach received data from DSS	YES, for data synchronization with other NESTORE modules.	Yes, it receives data from DSS
Cloud storage	YES, if history is required. GDPR compliance required.	YES, if history is required. GDPR compliance required.	YES, required for long term analysis.	NO	YES, for leaderboard and game data synchronization,	NO
WiFi	NO, it requires too much power and battery.	YES, if device is connected to the plug.	-	-	-	YES, if the device is connected to the plug
Zigbee/Z-Wave/Thread or other low energy communication	YES, but requires router or dongle in order to connect wearable to PC or mobile devices	YES, but requires router or dongle in order to connect wearable to PC or mobile devices	-	-	-	YES, but requires router or dongle in order to connect wearable to PC or mobile devices. Tangible Interface could be the dongle for this kind of communication technology
Bluetooth Low Energy	YES, it could connect directly	YES, it could connect directly	-	-	-	YES

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

	with PC and mobile devices. BLE allows for low data-rate. Only small data could be sent in order to save battery.	with PC and mobile devices.				
Offline working	YES, wearable could record data without any connection active and then download data when connected	YES, environmental sensors could be connected only sometimes and send data while connected.	DSS requires full-time internet access for real-time support (e.g. assess current progress in personal goals or exercises); all the other module requires a irregular connection.	YES, virtual coach uses offline DSS locally	YES, it works with local data and then re-sync when connected	YES, it works with local data and then re/sync when connected.
Mobile phone (iOS and Android)	Bluetooth Low Energy is the most common type of communication with wearable devices	Environmental sensors are always in the home, it can be connected with Bluetooth Low Energy, WiFi or other low energy communication technology	Run both locally on mobile platform and on the cloud	-	-	Tangible Interface can be a modified mobile phone with hardware output, or embedded devices like Arduino Board, Raspberry Pi or Android Things.

Table 2: Summary of NESTORE platform requirements with modules and notes

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

3. universAAL: what and how?

3.1.Introduction

universAAL – the universAAL project (http://www.universaal.org) is a large research project funded by the European Commission under the 7th framework program. It aims at developing an open platform and reference specification for Ambient Assisted Living (AAL) solutions [web1]. AAL refers to intelligent systems supporting people for a better, healthier and safer life in their preferred living environment (AAL Space). AAL applications [web2] as envisioned by universAAL are manageable, distributed and context-aware services; they cooperatively use the context information gathered by the devices and sensors embedded in the AAL Space. The OSGi framework with several of its service specifications is used as middleware to build the software infrastructure that hides the heterogeneity and distribution of resources and enables context sharing. Moreover, OSGi is also used as intelligent gateway to integrate Wireless Sensor Networks.

universAAL was aimed to be an open platform and reference specification for Ambient Assisted Living (AAL) solutions [1]. AAL means intelligent systems supporting people in their preferred living environment (AAL Space). universAAL see AAL applications as manageable, distributed and context-aware services; these uses together the context information gathered from devices and sensors embedded in AAL Space.

Definition: UniversAAL is an open-source software platform for Ambient Living where various, heterogeneous technical devices may be connected to a single, unified network. (The MS Windows and Apple MacOS platforms are only able to handle homogeneous technical devices.)

The devices involved in universAAL are sensors or actuators. Sensors provide the system with information about the current state of the environment (contextual information) such as: pressure sensor, motion sensor, brightness sensor, camera, clock, others. Actuators can be used by the system to influence the current state of the environment (such as: heater, TV, electric window).

universAAL IoT is an open platform for the integration of open distributed systems of systems, with development history expanded on several research projects such as EMBASSI, DynAmITE, PERSONA, universAAL, ReAAL [web 1], which has led to the creation of the initial universAAL IoT ecosystem. universAAL IoT is distributed with the Apache Software License 2.0, available under https://github.com/universAAL/ .

The universAAL platform is called a Platform, because it is more than just a software layer that lies between operating system and the applications in a distributed computer network (Middleware) and offer:

- Runtime Support (Implementation of the Execution Environment)
- Development Support (a suite of SW tools for supporting the SW developer)
- Community Support (a suite of SW facilities and technical infrastructure to assist end users, service providers and developers in community-building).

The platform can logically be divided into various layers: Middleware, Managers, Applications.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

3.2. Middleware

The Middleware is the only part of the universAAL platform that needs to be installed on every technical device that wants to actively participate in the system's communication; such a device is named an universAAL node.

The Middleware (Figure 1) hides the heterogeneity and distribution of the nodes; the applications that run on a node just communicate with the Middleware, they may be unaware of the actual device they are running on.

The Container lets the Middleware logic execute in different devices with plain Java (computers or embedded systems running OSGi, or Android smartphones). The Peering part is responsible for interconnecting and communicating of the Middleware.

The Communication part is the ultimate logic of the Middleware and enables the flow of the universAAL semantic across nodes by defining specific-purposes Busses (Context, Service and User Interface busses). Each bus handles its own specific strategy, semantics, reasoning and match-making.

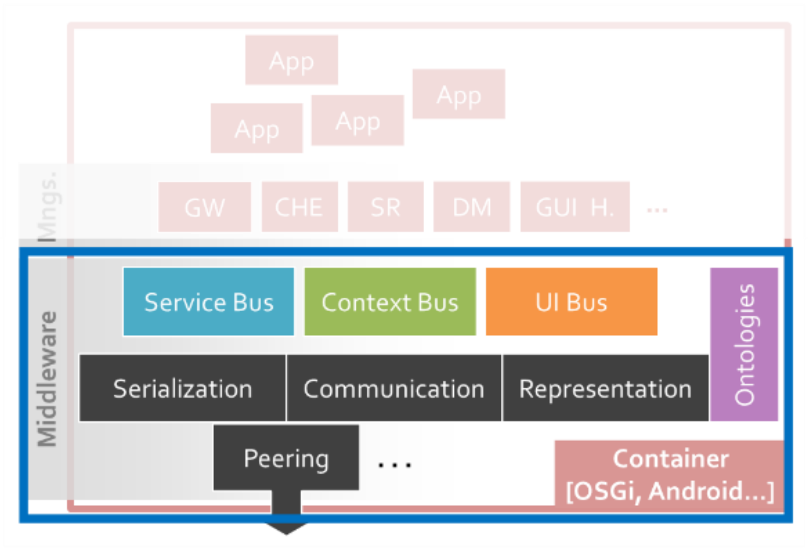


Figure 1 Runtime Support Platform – layered model

Regarding the buses responsibilities of middleware, it's outlined:

- The Context Bus is responsible for sharing contextual information – i.e. sharing knowledge that is used dynamically adapt services from application to the user and conversely. Examples of context are: identity, location (geographical data), status (temperature, ambient illumination, noise level) and time.
- The Service Bus is responsible for sharing the access to services, i.e. sharing functionality
- The User-Interaction Bus is responsible for sharing information to active user interaction.

3.3.uAAL Applications

uAAL Application is any piece of software that can run on the Container. The application platform must offer the possibility to run applications on multiple heterogeneous devices. The challenges arise from: independent

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

development and production of consumer items, ability to exchange data depends on networking protocol (switching and routing), access protocol (synchronization, FEC) and Data representation (compression, encryption), multiple application domain, multiple standards for each application domain and multiple profiles per standard.

The challenges for the platform is regarding the devices (mobile devices can be switched on and off, can fail to restart) the applications (can come and go, can be installed, updated and uninstalled, can fail to restart) this lead to the conclusion that the universAAL cannot be restarted for changes in a device/an application, but the platform and the application should auto-adapt to any change.

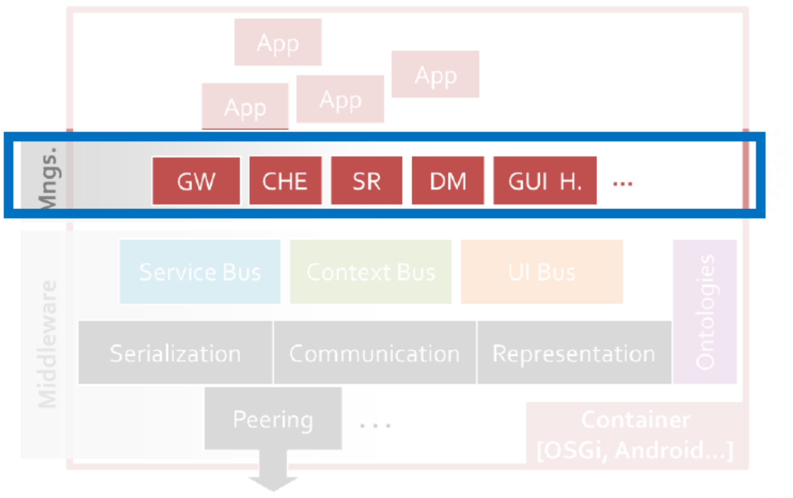


Figure 2 Runtime Support Platform

3.4. Managers

The Managers of Runtime support platform can be considered low level applications. Some are tied to certain Buses, while others are more widely used (Figure 2).

There are different Mangers are mandatory for the platform to work while some must be present only in one Node but not the rest. They usually also provide functional APIs to the above final apps.

Examples of managers:

- Context History Entrepôt: It provides services for consulting the history of events and obtaining the current context information status.
- Profiling & Space Servers and Editor: Used to obtain context (profile and space) information.
- Situation Reasoner: Based on a set of rules and conditions it composes new higher-level events based on those
- Dialog Manager: It takes care of maintaining the uAAL Main Menu interface and initiating the application interfaces listed there.
- UI Handlers: Refer to the “User Interaction Handlers”

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 769643

- Resource Server: applications can store multimedia resources here that can be later referenced
- Remote Import/Export: These managers act as proxies that can represent external web services as universAAL services inside the space, or publish universAAL services of the space to the outside as web services.

3.5. Semantic Services

Knowledge is shared in uAAL in the form of Ontologies. Ontologies can be represented in some standard format such as serialized RDF, but in universAAL Ontologies have a code representation so they can be handled by the Middleware. Ontologies are grouped in domains representing a single field of knowledge such as Devices, Health or Furniture. Resources are how the ontologies are represented, being in the Middleware representation like a class, and identified by a URI.

The purpose of ontology inside universAAL is: distribution of knowledge (Context Bus in uAAL) and sharing of functionalities (Service Bus in uAAL). Ontologies are made up of classes, properties, and data types. Every ontology has a uniform resource identifier URI.

Example of RDF statement for universAAL (http://ontology.universaal.org/Lighting.owl#LightSource) is presented in the Figure 3.

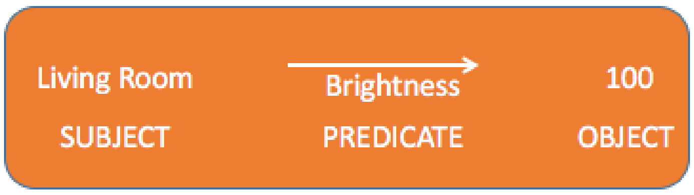


Figure 3 RDF statement

3.6.Implementation in UniversAAL

Messages inside universAAL are secured using a common symmetric cryptographic key. The distribution of messages can be:

- direct (node to node),
- multicast (node to set of nodes), or
- broadcast (node to all nodes).

```
public class LightSource extends PhysicalThing
{
public static final String MY_URI =
" http :// ontology . persona . ima . igd . fhg .de/ Lighting . owl # LightSource ";
public static final String PROP_AMBIENT_COVERAGE =
" http :// ontology . persona . ima . igd . fhg .de/ Lighting . owl # ambientCoverage ";
public static final String PROP_HAS_TYPE =
" http :// ontology . persona . ima . igd . fhg .de/ Lighting . owl # hasType ";
public static final String PROP_SOURCE_BRIGHTNESS =
" http :// ontology . persona . ima . igd . fhg .de/ Lighting . owl # srcBrightness ";
}
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

These messages are classified according to patterns of communication (publish/subscribe, general request/response, or request/response for interacting with human users); thus, there are different “buses” (Figure 4), each serving its specific purpose using an appropriate own brokerage strategy, where a bus is a message broker in charge of routing the messages to the appropriate software modules on appropriate nodes based on its brokerage strategy.

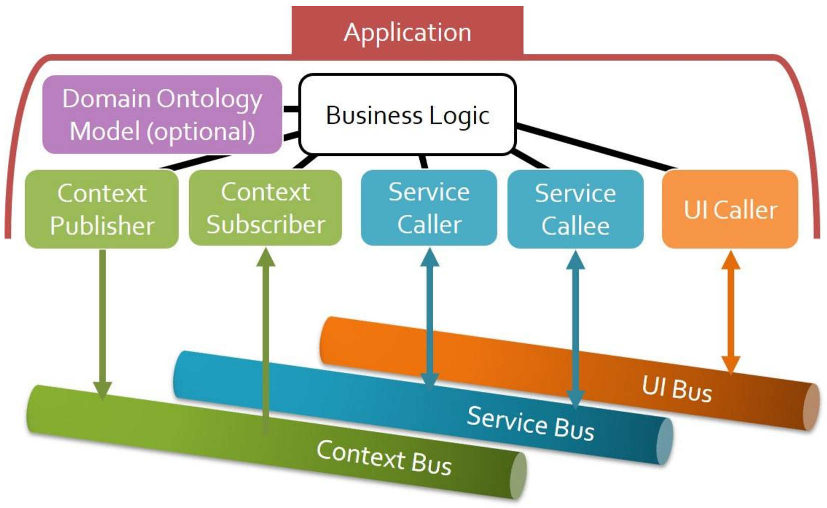


Figure 4 An Application running in the container connects to the buses using its own wrappers

3.7.The software

The software necessary to install and run the universAAL platform can be downloaded and tested from https://github.com/universAAL/middleware/wiki .[web12]

Table 3 universAAL platform

LINKS OF INTEREST	
PAX COMPOSITE BUNDLE	scan-composite:mvn:org.universAAL.middleware/mw.composite/x.y.0/composite
KARAF FEATURE	uAAL-MW
JAVADOC	https://universaal.github.io/middleware/mw.pom/apidocs/index.html
MAVEN SITE	https://universaal.github.io/middleware/mw.pom/index.html

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 769643

REPOSITORY STATUS	https://github.com/universAAL/platform/wiki/Repository-Status
MAVEN RELEASE REPOSITORY	http://depot.universaal.org/maven-repo/releases/org/universAAL/middleware/
MAVEN SNAPSHOT REPOSITORY	http://depot.universaal.org/maven-repo/snapshots/org/universAAL/middleware/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

4. NESTORE Proof of Concept

This chapter is structured in several sections as following:

- Hardware components
- Software components
- Development
- uAAL Setup
- Proof of Concept details
- Challenges
- Conclusions

Within NESTORE project, we aim to analyses the possibility of usage and integration of universAAL platform inside NESTORE system.

universAAL is the result of several European research projects focused on creating an open platform and standards which will make it technically feasible and economically viable to develop Ambient Assisted Living solutions.

While leveraging state of the art technologies, the innovation brought about by NESTORE relies, first, in its integration capacity, meaning we can see NESTORE according to different perspectives:

1. as a platform, providing a set of basic services that can be leveraged by different apps, potentially offered by different stakeholders;
2. as a private companion, offering personal and personalized support according to the user needs and interests;
3. as a social support, easing communication with family and friends as well as with careers, but also able to propose external service offerings based on “group” interests.

Looking at NESTORE as a platform, we aim to offer an open system that re-uses and integrates capabilities such as those provided by universAAL.

To perform the analysis as close as possible to the project goals, technological partners from NESTORE consortium built a Proof-of-Concept modelling a common use case: keeping a constant temperature in a room. The approach is generic and can be applied to other use cases from the IoT universe.

4.1. Hardware components

Hardware configuration was choose to be representative for a home automation scenario. There is a gateway (home automation hub) device coordinating the IoT nodes from the location. Communication between the hub and the nodes is done over a private WiFi network (not shared/used by any other devices). External communication is done through an Ethernet link (wired).

The hub is run on an embedded ARM Linux SBC (Single Board Computer) with Allwinner A20 dual-core CPU.

The IoT node runs on an evaluation board for ESP8266 with a relay and temperature sensor (connected to board's GPIO pins).

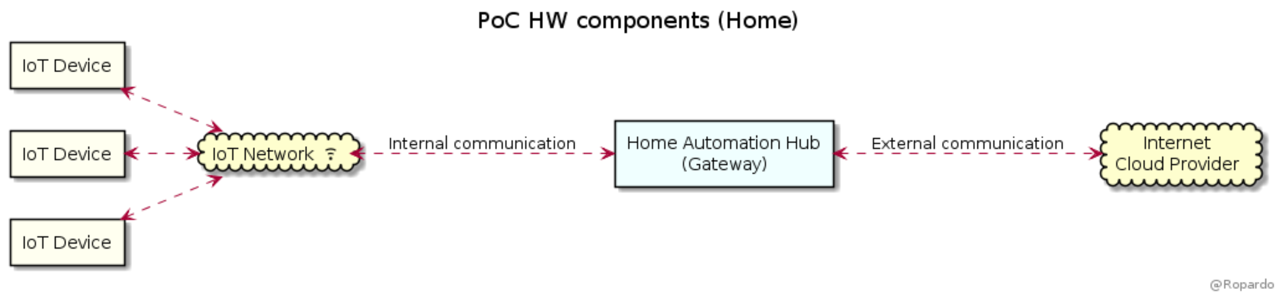


Figure 5 NESTORE PoC Hardware components

Both the hub and IoT node are open source hardware which can configured and modified for specific needs (no hardware modifications for the proof of concept).

The automation hub hardware configuration includes[1]:

- Allwinner A20 dual core Cortex A7 CPU – 1GHz
- 1 GB DDR3 RAM
- 1 GBit Ethernet
- Native SATA support
- LiPo battery backup
- MicroSD card connector
- 5V DC

The IoT node uses ESP8266 (low cost, low power, wifi 2.4GHz)[2]:

- ESP8266EX
- 2MB SPI Flash memory

[1] https://www.olimex.com/Products/OLinuXino/A20/A20-OLinuXino-LIME2/open-source-hardware
[2] https://www.olimex.com/Products/IoT/ESP8266-EVB/open-source-hardware

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

- 5V DC

4.2. Software configuration

The automation hub runs a Linux distribution for ARM SBCs (Armbian Jessie based on standard Debian 8). Beside the latest Debian 8 packages we installed the dependencies for universAAL and IoT packages:

- Java 1.8.162
- Apache Karaf 3.0.8
- Mosquito
- Python 3.5.5

For the IoT node we used the Arduino core for ESP8266 together with a couple of Arduino sketches for temperature sensor:

- OneWire
- DallasWireless

4.3. Development Configuration

The configuration used for development and implementation the Proof-of-Concept includes a mix of older and newer Java versions. We tried to use the newest working version while doing minimal changes to universAAL packages. We succeeded in having a functional self-compiled environment by using:

- Java 1.8.0_162
- Java 1.7.0_80
- Eclipse Indigo SR 2
    - uAAL Studio plugin
- Karaf 3.0.8
- Nexus 3
- Maven 3.5.2
- Arduino IDE

Java 1.8 was used for running Karaf and compiling universAAL middleware (keeping compatibility with Java 1.5). Java 1.7 was used for running Eclipse Indigo together with the uAAL Studio plugin.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

Karaf 3.0.8 (the oldest still maintained Karaf version) was used instead of 2.x used in universAAL due to dependencies needed for PoC (API) – too old and buggy/not working in 2.x. We tried to use Karaf 4 but failed during deployment of universAAL (most likely additional changes are needed in middleware code to make it perform flawlessly on newer OSGi containers – but we did not deeply investigate the root cause of the problems).

4.4. uAAL Setup

Development environment was setup following instructions from:

- https://github.com/cstockloew/platform/wiki/DG-Quick-setup-and-start-guide.

Build instructions:

- https://github.com/universAAL/platform/wiki/Building-from-Source

The universAAL repository did not have the latest builds for all components. Some modules were at 3.4.2_SNAPSHOT, some of them were at 3.4.1. This resulted in impossibility to run the samples (versions not found) and the decision to have an independent repository.

The middleware was built from sources in order to have an NESTORE Nexus repository for the latest code level (3.4.2_SNAPSHOT).

Build was not working out of the box and we had to disable unit-testing (at least one-unit test failed so we skipped them to get a minimal working version compiled). The entire process of building and publishing the artefacts takes quite a long time and this can be seen as a potential issue if code changes are needed in the platform (for whatever reason).

Once the internal repository was populated the tested samples worked fine (after adjusting the source repo, maven configuration etc., but this is just standard java development patterns).

Setup of the Eclipse plugin encountered no issues but its usage is kind of limited. The generated code does not follow the latest changes in the middleware (rename of the classes) and running of the Run Config seems to be no longer supported (February 2018)[3]. This resulted in the need to manually modify the code templates and to run the modules directly in Karaf. For pure Java development we don't see the need to use the plugin. No assessment was done for ontology development and its specific needs.

4.5. Proof of Concept details

We use a similar approach with uAAL structure (multi-module maven project) and the Eclipse project structure contains:

[3] https://github.com/universAAL/platform/issues/45

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

- hvac.pom – used for tracks common libraries versions (uAAL platform)
- hvac.shared – allow common code to be reused through the other modules
- hvac.driver.temperature – represents uAAL driver for temperature sensor
- hvac.driver.relay – represents uAAL driver for relay (actuator)
- hvac.reasoner – represents decision making code for actuator activation based on temperature input
- hvac.api – REST/JSON api for observing the system status and configuring various parameters

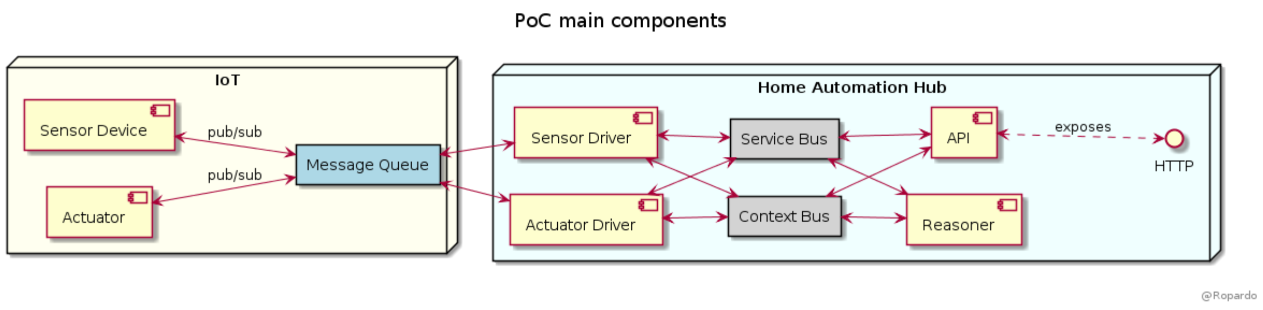


Figure 6 Software components

The modules presented above were generated with uAAL wizard but required changes to work with latest uAAL platform. They follow a common structure (they're OSGi bundles) so it would be straight forward to create a template bundle and then just reuse/duplicate it for something specific (without the need to use/update from studio wizard).

Figure 6 illustrates the components setup and usage for the IoT domain and for the home automation hub domain. The IoT consists in the sensor device (temperature sensor) and actuator (relay) whose code runs on the IoT node.

The communication of sensor data and actuator commands/status is done using a message queue (implemented with Mosquito running on the linux node). The message queue is checked (subscribed) by the sensor and actuator drivers. The sensor and actuator have separate drivers implemented as universAAL/OSGi bundles. Each driver subscribes to specific topics on the message queue and generate events on the context bus in the universAAL environment. Both drivers expose services on the service bus to receive commands (sensor driver has setup commands, actuator driver has on/off commands for the relay).

In addition to the drivers the hub runs the reasoner and API. The reasoner listens for temperature events on the context bus and decides to switch the relay on or off in order to keep the temperature in a narrow band. The API module listens for temperature and relay events and keeps a status of the system that can be publish over HTTP to an external client (uses standard REST/JSON interface).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

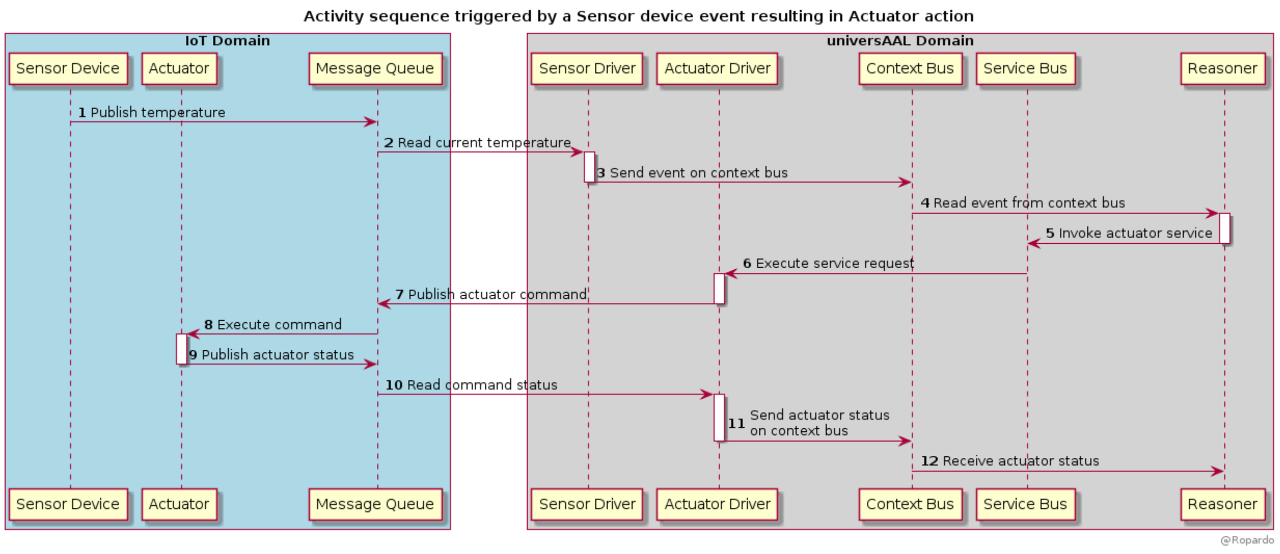

Activity sequence triggered by a Sensor device event resulting in Actuator action
IoT Domain
universAAL Domain
Sensor Device
Actuator
Message Queue
Sensor Driver
Actuator Driver
Context Bus
Service Bus
Reasoner
1 Publish temperature
2 Read current temperature
3 Send event on context bus
4 Read event from context bus
5 Invoke actuator service
6 Execute service request
7 Publish actuator command
8 Execute command
9 Publish actuator status
10 Read command status
11 Send actuator status on context bus
12 Receive actuator status
@Ropardo

Figure 7 Main Sequence Flow

A typical closed-loop (feedback) automation sequence is detailed in Figure 7:

- temperature sensor publishes its value into message queue on a topic dedicated to the sensor (uses its hardware ID) (step 1)
- sensor driver sees the temperature value (it's subscribed to the sensor's topic), reads it (step 2) and publish an universAAL event on the context bus (step 3)
- the reasoner is subscribed to context bus and receives the new temperature value (step 4)
- if the value is outside the temperature range to be kept then it decides to change the actuator state (on or off depending on the current state and temperature value); for this it calls a service using the service bus (step 5).
- The actuator driver is invoked for the request (step 6) and publish the actual command on the IoT message queue (step 7)
- The IoT actuator receive the command (it's subscribed to the message queue) (step 8), execute it and generates and event with the actual state of the physical actuator (in the eventuality the command was rejected by the actuator) (step 9)
- The actuator driver received the actual status of the actuator (step 10) and publish an event on the context bus (step 11)
- The event is received by the reasoner (step 12) which will update its internal model (and will use the new state during the next temperature read/event).

Each uAAL bundle has a similar code structure:

- Each bundle has an Activator (required by OSGi specification) that setup/tear-down the bundle
- Events are received / emitted from context bus using a Publisher/Subscriber mode (based on ontology definition)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

- Services are offered by registering Service Profiles (describe what's offered)
- Services are full filled on service call requests
- Services are invoked filling in a Service Request
- Parameters can be Java primitives or uAAL ontology entities (Derived from Resource)
- Any other Java library can be used (if it needs UI, like AWT/Swing/SWT then the Karaf container must run on a platform supporting the required UI – i.e. no embedded deployment).

The API module is used to provide a standard interface (REST/JSON over HTTP) to the internals of the universAAL system. It allows the query (GET) and update (PUT) of various values:

- GET / - retrieves JSON with full system status
- GET /temperature/current – retrieves the current temperature
- GET /temperature/target – retrieves the target temperature set in the system
- GET /temperature/threshold – retrieves the temperature threshold for starting/stopping the relay
- GET /temperature/delay – retrieves the delay between readings
- GET /status – retrieves the status of the system (if is enabled or disabled)
- GET /relay – retrieves the status of the relay (if it's on or off)
- PUT /temperature/target – Modifies the target temperature with 1 degree (+ or -)
- PUT /temperature/threshold – Modifies the threshold using 0.1 degree steps (+ or -)
- PUT /temperature/delay – Modifies the delay between reads in 1second steps (+ / -). When the delay is 0 the sensor is disabled (won't send any more temperature reads).
- PUT /status – Enable or disable the overall system status (when disabled no reasoning is performed)
- PUT /relay – Sets the relay status

4.6. uAAL Challenges

The implementation of our PoC using the uAAL outlines that for this kind of work one needs above-average resources able to handle complex environment of universAAL for all life-cycles: setup, development, deployment, maintenance. The platform could not be approached as just another Java library or solution because it required compilation from sources, script changes to be able to run on newer software components, looking through code to pinpoint what went wrong or was used incorrectly, nexus setup for using it etc. (with a

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

common java library or platform one gets the binary, read the documentation or samples, setup some maven links and generate/compiles the project using it).

During the development process there were random Karaf (uAAL) crashes happening after several bundle deployments without no overall restart. This indicates our uAAL setup is unstable for complex development.

The runtime performance on embedded devices (likely to be used as automation hubs) was poor during startups: the Karaf with uAAL deployed into it took between 90 and 120 seconds for the startup, as comparison, the Karaf container without uAAL only took 10-20 seconds. This can be an issue for real-life usage where the users expect fast startup (couple of seconds) after power on (and it adds the need for a feedback mechanism to the users, to make them aware of the current device status – crashed, powering up, running…).

4.7. Conclusions relating POC

The main objective of the activities reported in this chapter was to learn and understand universAAL platform and the way it can support NESTORE architecture development. After the deep analysis conducted within T6.2, some key points have been found relevant to understand the involvement of such technology.

First, there are some strong concerns about UniversAAL documentation and project roadmap, since there hasn't been a wide adoption among community. Moreover, it is not clear whether any integration task could be developed by UniversAAL development group, supporting NESTORE partners within the strict time frame planned by the project.

UniversAAL requires to install Karaf and Java environment on each node, this create a very restrictive constraint on the kind of device it can be installed too which in turn raise issues on the size, cost and battery usage of the devices.

UniversAAL provides no synchronization or data/message persistence mechanism between nodes, this means the synchronization has to be implemented manually and with some custom mechanism external to the platform when needed. Furthermore if a node goes down for any issue (network, manually rebooted, ...) and it misses a message from a bus there's no recover mechanism provided by universAAL, which result in a very poor fault tolerance

In addition, some strong constraints from an architectural point of view raised more concerns:

- when installed on discrete devices, it has been found that there is no support for a wide range of nodes, so a communication mechanism must be put in place such as an MQTT channel between home IoT nodes and a central hub running UniversAAL

- authentication of home central hub must be managed outside UniversAAL because there is no device provisioning provided by the framework

- UniversAAL relies on outdated libraries (provided by Apache foundation) and it is not up to date with latest releases; this can introduce security vulnerabilities that will need a full recompile (and potential code changes) of project from source code to include bug fixes and increase support

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

- there is no support for communication errors or retrieval: developer must assure the node is in a consistent state and handle any error to fail back whenever an error occurs

Another drawback of the universAAL platform is represented by offering of two buses for events (context) and commands (service); communication is done transferring serialized (as RDF strings) java objects representing ontological entities.

Thus, from technical point of view, it is clear after the PoC implementation that same functionality (as in this PoC) can be achieved with a lightweight message queue using JSON data.

In conclusion, proposal is to split NESTORE architecture from UniversAAL message bus and adopt a different platform to handle nodes and support the widest adoption. Then, an extension to NESTORE can be obtained leveraging UniversAAL as an integration layer within NESTORE platform, assigning this task to UniversAAL group, under strict timing since they have the knowledge about their system and can effectively translate ontology and data from NESTORE data mode to UniversAAL model.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

5. Proposed alternative

5.1. NESTORE & uAAL

NESTORE objective is to deliver the Coach activities; to do this the uAAL platform was proposed as candidate to be used as support platform. It is not the NESTORE project scope to create / define a new IoT platform but to use an existing one.

Using uAAL to deliver NESTORE Coach(es) requires investing a considerable effort into uAAL platform (to update to latest java version as well latest Karaf and used libraries). It was found challenging to simply build the uAAL from the GitHub sources, due to the complexity of the environment and choosing of default configuration (unit test crash, additional small changes needed). Even for the samples published on the GitHub we found out that because of the source code consistency with available binaries from uAAL maven repository it’s not possible to just run them. You need to adapt, change, build from sources, this make even harder to adopt the uAAL without substantial effort unaccounted in the project proposal. Some parts of online documentation were not updated and synchronized with the actual source code.

All the NESTORE coaching applications are involving different subsystems that partners developed with different programming languages that are not compatible with uAAL requirements (only Java or something compiling to JVM bytecode). For Coaching activities different AI modules that are running Python or Go are used, for device management is used PHP / NodeJS, etc.

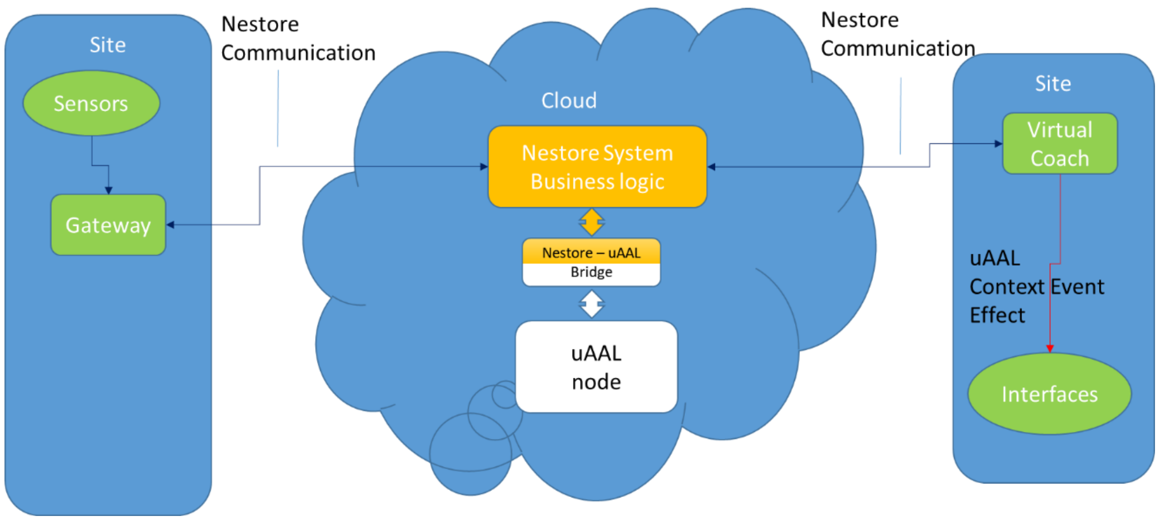


Figure 8 NESTORE & uAAL

To support the uAAL by the NESTORE project a dedicated bridge NESTORE-uAAL is proposed to be built, to allow the communication and data exchange between the two systems (Figure 8 NESTORE & uAAL)

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 769643

Supposing that different projects are using cloud based architecture, we can consider to a multi-deployment scenario in which components of different projects are installed on the same site.

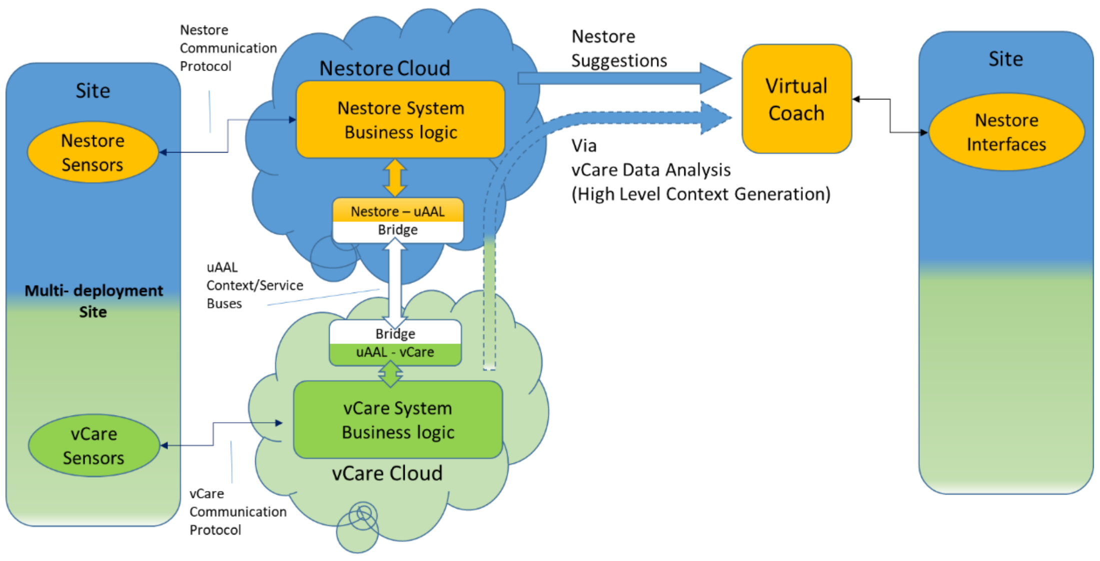

Nestore Communication Protocol
Site
Nestore Sensors
Multi- deployment Site
vCare Sensors
Nestore Cloud
Nestore System Business logic
Nestore – uAAL Bridge
uAAL Context/Service Buses
Bridge uAAL - vCare
vCare System Business logic
vCare Cloud
vCare Communication Protocol
Nestore Suggestions
Via vCare Data Analysis (High Level Context Generation)
Virtual Coach
Site
Nestore Interfaces

Figure 9 NESTORE - vCare integration

Each component sends context information to its own cloud, then information is elaborated in higher level context that is shared through uAAL context buses among the cloud-net. (Figure 9 NESTORE - vCare integration)

For example, the NESTORE-uAAL bridge would translate the semantic data written according to a uAAL ontology in the proper data format used by the NESTORE DSS to generate new advises for the Virtual Coach.

The NESTORE bridge architecture will follow the experience gained on the presented proof of concept. The software artefact that resulted from the POC will be utilized as a starting point for the development phase. On the bridge development the ontologies that are defined on the WP2, under the task T2.5 will represent the knowledge backbone for the NESTORE – uAAL integration.

5.2.NESTORE IoT

A typical IoT solution is characterized by many devices (i.e. things) that may use some form of gateway to communicate through a global area network to enterprise back-end servers that are running an IoT platform which helps integrate the IoT data into the existing enterprise (Figure 10). The roles of the devices, gateways, and cloud platform are well defined, and each of them provides specific features and functionalities required by any robust IoT solution.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

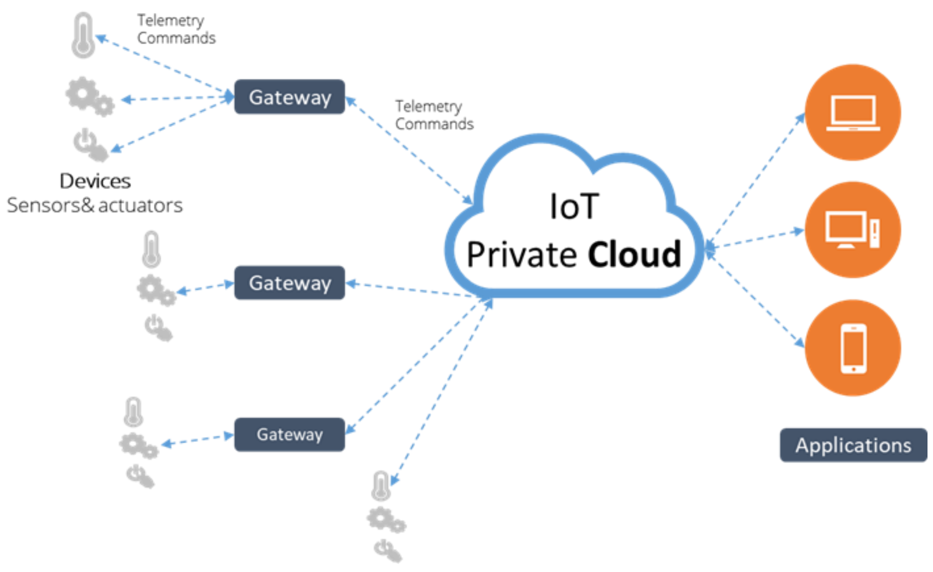


Figure 10 A typical IoT solution

An IoT Solution requires substantial amount of technology in the form of software, hardware, and networking. (Figure 11)

When implementing an IoT solution scalability is an important parameter to be considered; how to support many users at latter stage but have a small enough footprint when the project is in inception phases.

The web has become widely successful nowadays because it's simple to learn and use; it emphasizes loose coupling between servers, browsers, and applications. The simple and clearly defined programming model of HTTP makes it possible for anyone to change pieces of the system without breaking the whole system.

The Web of Things is a specialization of the Internet of Things where web development principles that made web so successful, applies it to embedded devices to make the latest developments in the Internet of Things accessible to as many developers as possible.

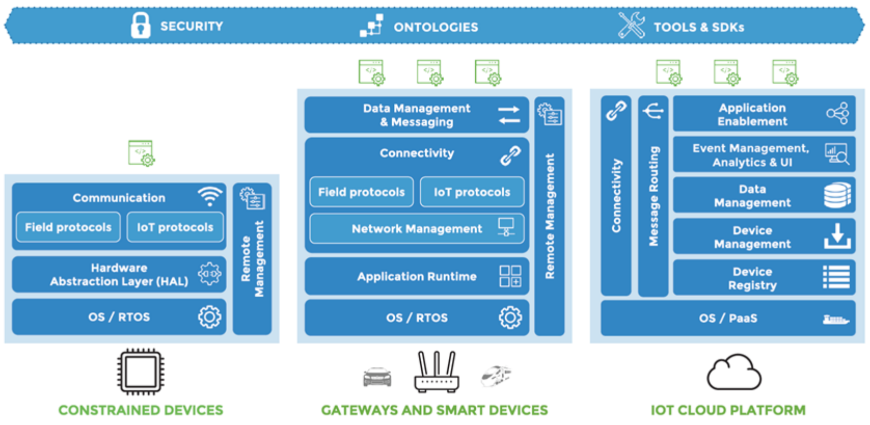


Figure 11 IoT Components - by Eclipse Foundation

Cross stack functionality need to be considered for any IoT architecture:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

- Security – Security needs to be implemented from the devices to the cloud. Features such as authentication, encryption, and authorization need be part of the solution stack.
- Ontologies – The format and description of device data is an important feature to enable data analytics and data interoperability. The ability to define ontologies and metadata across heterogeneous domains is a key area for IoT.
- Development Tools and SDKs – IoT Developers will require development tools that support the different hardware and software platforms involved.

NESTORE recognize the need of an IoT system to allow data acquisition from the users to the NESTORE cloud where Coaching algorithms are implemented.

Based on the presented constraints of the uAAL, NESTORE project will implement the IoT system around a well-supported open source and light system – **Home Assistant** (home-assistant.io).

User management functions presented by the system:

- Mange devices per user – personal devices as well wearable
  - Register / Un register
  - View status

With the proposed IoT platform NESTORE is able to manage the user Assets. Define the user assets and assign devices / sensors to the user in a simple way will support end users to understand and maintain his IoT universe.

5.2.1. IoT Deployment

Deployment of the IoT can be realized around the wotAgent (Web of Things Agent) that can be implemented at different levels.(Figure 12)

- On the private cloud – then sensors must be technically able to send data to the cloud wotAgent
- At the level of sensors network – then the wotAgent will be implemented on a local gateway device
- On mobile device – the mobile device is responsible to acquire data from other specialized sensors or devices and the mobile wotAgent is transferring data to the cloud

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

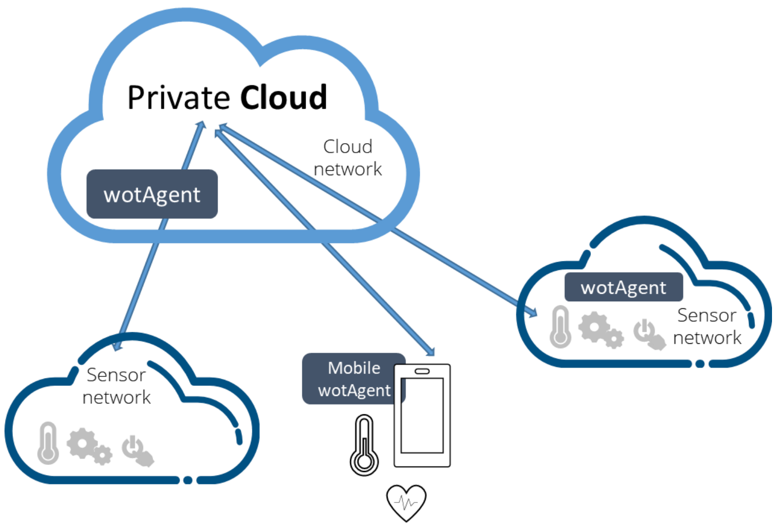


Figure 12 Deployment using wotAgent

First scenario supposes that:

- The device / sensor is using a communication protocol that is secured and Internet enabled
- It is a VPN between the device/sensor network and Cloud

Device side wotAgents are running on the same network with the sensors and can run on a dedicated gateway, mobile phone, router, etc. . The NESTORE wotAgent will take in consideration the power needs in respect to the running device and network as well the computation needs – memory, processors.

The wotAgent will be a cross cooperation work distributed between WP3 and WP6.

5.3. NESTORE Message Bus

Messaging is a key strategy employed by NESTORE cloud distributed environment. It enables NESTORE applications (i.e. ChatBot, LogIn/Register, etc.) and services (i.e. DSS services) to communicate and cooperate. It also sustains a scalable and resilient solutions. Messaging supports asynchronous operations, enabling NESTORE to decouple a process that consumes a service from the process that implements the service. The NESTORE message bus enables a message to be posted to a queue via HTTP (actually always HTTPS) and this will ensure a simple and strong security model as well a universal accepted transport protocol represented by HTTP(S).

A message queue receives messages from an application and makes them available to one or more other applications. In the NESTORE architectural scenarios, if application A needs to send updates or commands to applications B and C, then separate message queues can be set up for B and C. A would write separate messages to each queue, and each dependent application would read from its own queue (the message being removed upon being dequeued). Neither B nor C need to be available for A to send updates. Each message queue is persistent, so if an application restarts, it will begin pulling from its queue once it is back online. This helps break dependencies between dependent systems and can provide greater scalability and fault tolerance to applications.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

The modern message queue implementations (such as Apache MQ) support having a single message be read by multiple endpoints. Messages become “invisible” to applications that have read them for some period of time before actually being removed. During this time, the message can still be read by other applications. This blurs the line between queues and buses, especially as it pertains to the 1:1 correspondence between queues and destination applications (Figure 13).

Using message queue technology NESTORE will implement the Service Bus that will support the IoT data transmission to the NESTORE cloud as well it supports the NESTORE specific applications to communicate.

Inspired by the uAAL concepts, the “Context Bus” as well “Service Bus” will be supported by the NESTORE bus. Implementing these two buses enables the communication support for the NESTORE-uAAL bridge.

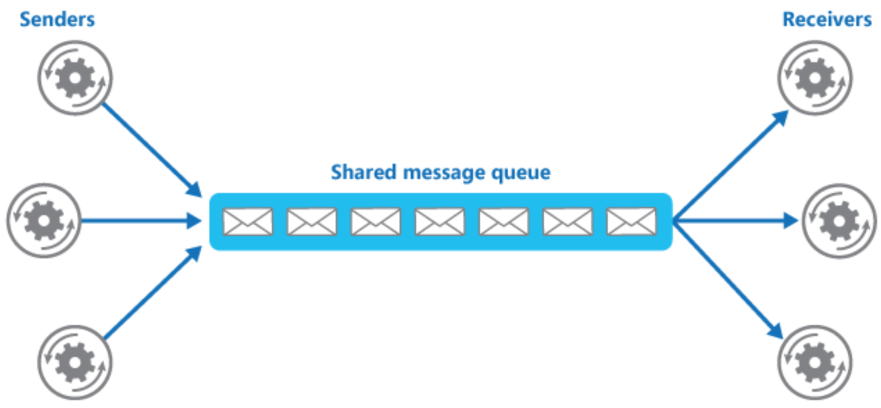


Figure 13 Sender - Receiver

The so-called “Context Bus” will handle "event-based" communication channel, which means that an application forwards information to this bus without taking interest in the existence of actual recipients. This may sound strange at first, but it is due to the nature of the messages that are shared via the Context Bus, as this bus is used for publishing information about the state of the environment and/or the assistive system. This kind of information is called "context" and it is used by the assistive system to recognize situations in which it is supposed to act. A typical "context provider", an application that publishes context information onto the “Context Bus”, would be an application talking to a sensor (such as a temperature sensor). At regular intervals, this application forwards the values it receives from the sensor to the “Context Bus”, maybe after having preprocessed them in a certain way. It is obvious that such an application does not need to know who is receiving its messages, that is, who the "context subscribers" are that have registered themselves to the “Context Bus” as recipients for this specific kind of sensor related messages (if there are any at all). The application's job is all done with having shared the information that it has acquired from the sensor.

The so-called “Service Bus”, on the other hand, is "call-based". In this context, the term "service" shall mean the provision of a functionality, or in other words, that someone is doing something for somebody else (we are well aware that "service" is one of those heavily overloaded terms that almost everyone seems to interpret differently, though). An application that offers a service (= can do something) announces this by registering a corresponding service profile, that is a description of what it is capable of doing, with the Service Bus. This kind of application is called a "service callee" (because it gets called by another application). NESTORE application will publish their profiles that will be used in the deployment phase.

For implementation of the “Service Bus” (Figure 14) a sender posts a message to a queue and expects a response from the receiver. This pattern is used to implement a reliable system where confirmation that a message has been received and processed must be assured. If the response is not delivered within a reasonable interval, the sender can either send the message again or handle the situation as a timeout or failure.

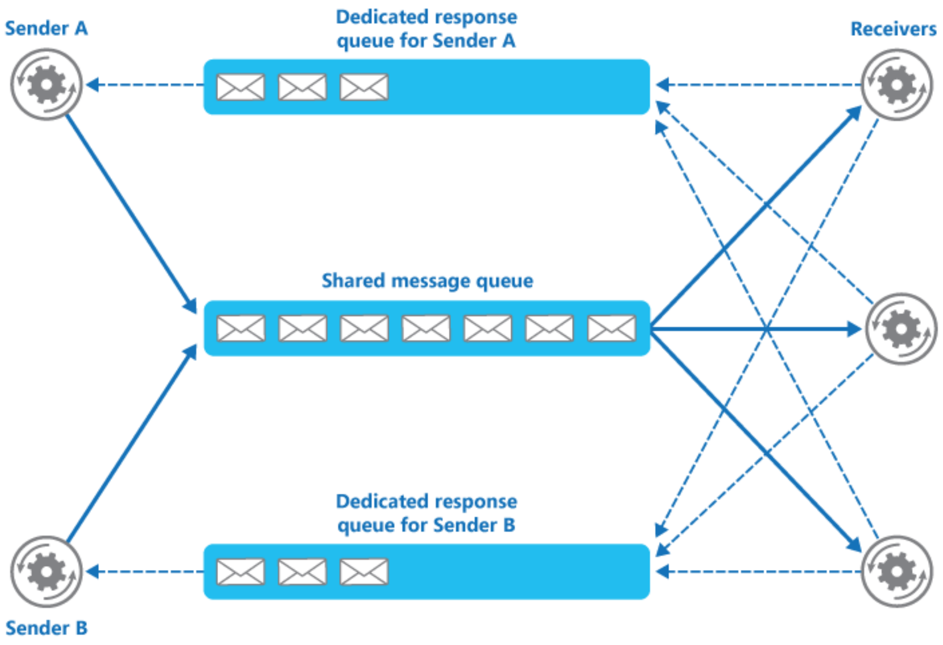

Sender A
Dedicated response queue for Sender A
Receivers
Shared message queue
Dedicated response queue for Sender B
Sender B

Figure 14 Request/response messaging with dedicated response queues for each sender

An exemplary service callee for NESTORE would be an application that can provide an advice, i.e. how many calories are in the plate deducted from picture taken by user, and it would register the appropriate service profile with the Service Bus (stating something like "I can provide number of calories"). The counterpart to the service callees are applications that require a service, the "service callers". They send a service request to the Service Bus, asking for a specific service (as in "I need number of calories from picture"). It is up to the Service Bus to then find one or more matching service profiles to the service request and, if a match is available, to forward the request to the corresponding service callee(s). The Service Bus is different to the Context Bus in that it is call-based, meaning the caller (= the service requester) expects an answer to its message - and might temporarily cease its execution to wait for that. Once the service callee has completed the job it was asked to do, it puts his answer, the "service response", on the Service Bus (as in "Find here number of calories"). The Service Bus makes sure that the answer really reaches the service caller application, which can then proceed with its execution (knowing that the lights in the living room should be off).

Messages carry a payload as well as metadata, in the form of key-value pair properties, describing the payload and giving handling instructions to Service Bus and applications. NESTORE will allow application to send / receive messages encoded as MQTT for the IoT sub-system as well payloads encoded as JSON objects for the inter application communication.

5.3.1. Using NESTORE Message Queue as uAAL buses

Our objective is to design and develop a system compliant to the uAAL results in terms of open platform and reference specification for the AAL scenario. Within the universAAL ecosystem, an AAL Space is intended to be the physical environment – such as the home of an assisted person – in which independent-living services are provided to people that need any sort of assistance. In NESTORE, the reference scenario is more oriented to the person rather than the environment, enlarging the AAL space and opening a research challenge from the architectural point of view. In such an extended virtual ecosystem, hardware as well as software components can "live" while being able to share their capabilities. In this space, the universAAL platform can be still used to facilitate the sharing of three types of capabilities: Context (data based on shared models), Service (control) and User Interaction (view). Therefore, connecting components to the universAAL platform is equivalent to using the brokerage mechanisms of the uAAL platform in these three areas for interacting with other components in the system. Such connectors, together with the application logic behind the connected component, are called altogether "AAL Applications". Figure 15 shows the universAAL reference architecture with the adaptation mechanism used to integrate 3rd party services, as in the case of NESTORE applications.

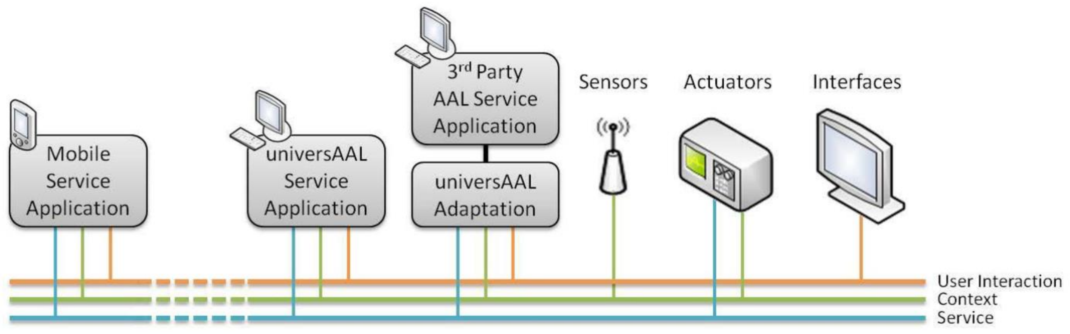
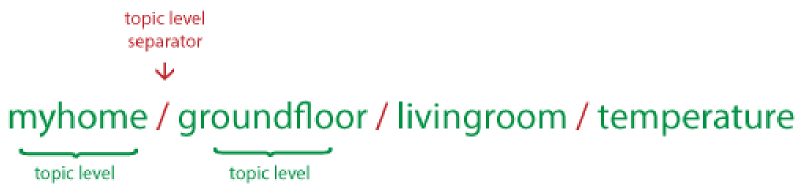

Mobile Service Application
universAAL Service Application
3rd Party AAL Service Application
universAAL Adaptation
Sensors
Actuators
Interfaces
User Interaction
Context
Service

Figure 15 The integration with universAAL is equivalent to "talking" to its buses with specific roles

As stated in the DoA, it is our intention to be aligned to the universAAL results, to reuse the Open Source software released by universAAL as much as possible, and to share the use cases based on the NESTORE Virtual Coach system. In this way, we can enrich the universAAL platform with the technological requirements deriving from the NESTORE project. In NESTORE, devices installed in the user's home are abstracted with the Web of Things (WoT) approach reusing existing and well-known Web standards (REST, JSON, MQTT). In this view, the integration with universAAL is equivalent to "talking" to its buses with specific roles. In particular, we use MQTT for creating the shared message queue among the NESTORE module and uAAL buses.

MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple, and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium [web4].

MQTT is based on the principle of publishing messages and subscribing to topics and they are used to decide on the MQTT broker which client receive which message. A topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. A topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator in Figure 16).



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

Figure 16 A generic MQTT topic

In order to "talk" to the uAAL buses, we need to separate the concerns of each topic reflecting the functionalities of the relative uAAL bus. We use the root topic level "serviceBus" to discover new sensors and services in the NESTORE ecosystem for each user (identified by the subtopic "uid"). Each subtopic will act as *handler* for the relative sensor/service. We use "contextBus" for collecting all the messages produced by sensors and services in the NESTORE/uAAL ecosystem of each user and the "uiBus" to send and receive message from the primary UI with the NESTORE user: the virtual coach.

Figure 17 shows the interaction between the uAAL buses and the NESTORE applications by means of the above-mentioned aggregation of MQTT topics.

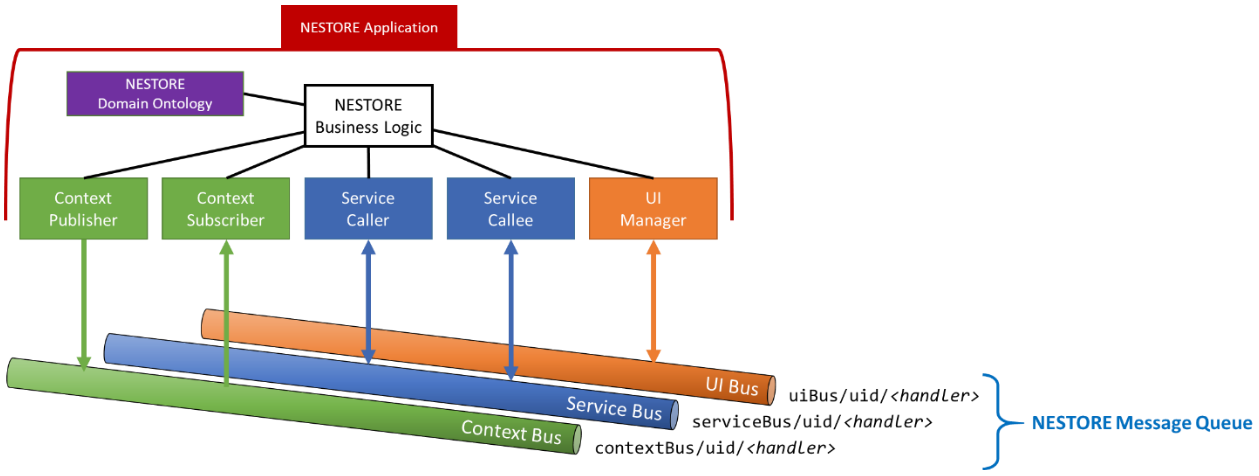
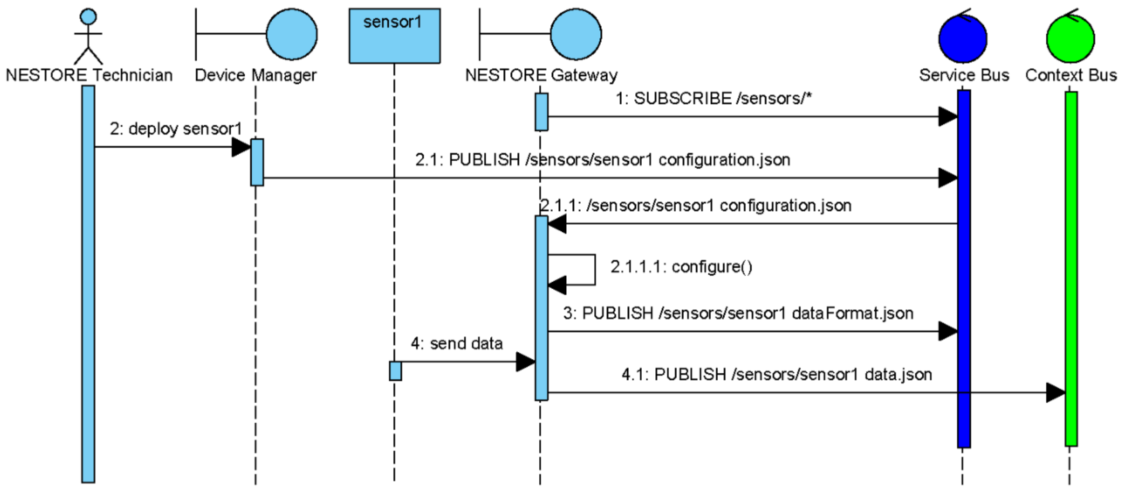


Figure 17 Separation of concerns in different MQTT topics and their relationships with uAAL buses

When a new sensor/service is deployed in the NESTORE/uAAL ecosystem, it is announced publishing a JSON message on the serviceBus containing its description in terms of capabilities and configurations. In this way, another service previously subscribed to the serviceBus is notified, i.e. it "discover" the new sensor/service, and can start to collect data from the relative subtopic in the contextBus. Figure 18 shows the sequence diagram of the announce phase of a newly deployed sensor. It should be noted that messages published on the serviceBus are *retained*, helping newly subscribed clients to get a status update immediately after subscribing to a topic and without waiting until the publishing clients send the next update.


NESTORE Technician
Device Manager
sensor1
NESTORE Gateway
Service Bus
Context Bus
1: SUBSCRIBE /sensors/*
2: deploy sensor1
2.1: PUBLISH /sensors/sensor1 configuration.json
2.1.1: /sensors/sensor1 configuration.json
2.1.1.1: configure()
3: PUBLISH /sensors/sensor1 dataFormat.json
4: send data
4.1: PUBLISH /sensors/sensor1 data.json

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

Figure 18 Sequence diagram of the announce phase for discovering a newly deployed sensor

Figure 19 shows a more comprehensive example of the interaction of NESTORE modules with the uAAL buses. A generic NESTORE analytic tool (e.g. calculating real time statistics on data sensed by a generic sensor1) first publishes its api to the serviceBus in order to be properly called by other modules, then it subscribes to the serviceBus topic related to the sensor in which it is interested waiting for its availability and its data format. As soon as it receives the notification of the presence of the requested sensor (sensor1 in the figure), it subscribes to the corresponding contextBus topic and starts collecting data. The same process applies to the NESTORE Decision Support System (DSS) module that is in charge of generating reccomendation for the Virtual Coach using the results coming from the analytic tool. The DSS will publish the computed reccomendation to the uiBus that will dispatch the message to the Virtual Coach application, previously subscribed to the uiBus, that will finally diplay the coaching message to the user.

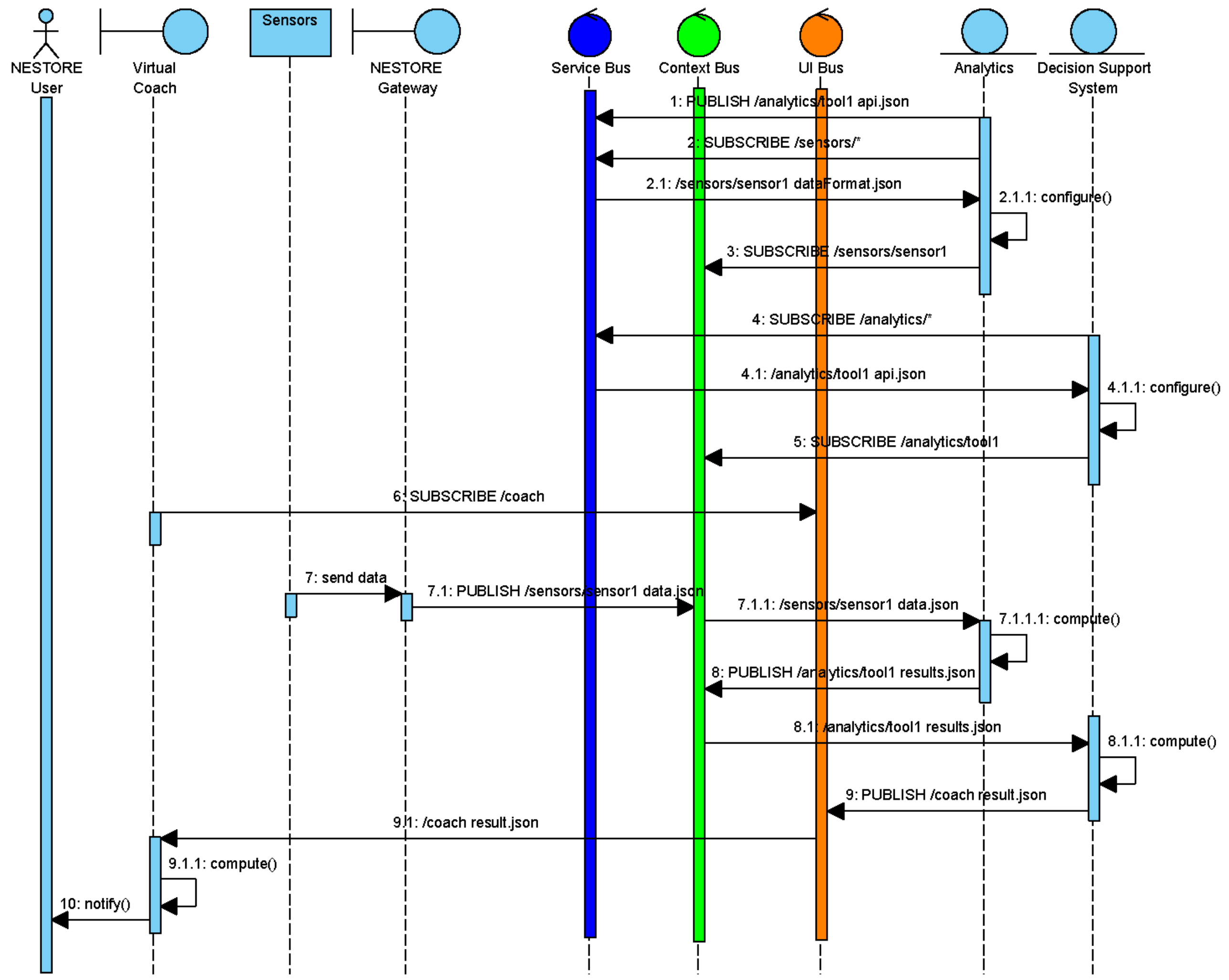

NESTORE User
Virtual Coach
Sensors
NESTORE Gateway
Service Bus
Context Bus
UI Bus
Analytics
Decision Support System
1: PUBLISH /analytics/tool1 api.json
2: SUBSCRIBE /sensors/*
2.1: /sensors/sensor1 dataFormat.json
2.1.1: configure()
3: SUBSCRIBE /sensors/sensor1
4: SUBSCRIBE /analytics/*
4.1: /analytics/tool1 api.json
4.1.1: configure()
5: SUBSCRIBE /analytics/tool1
6: SUBSCRIBE /coach
7: send data
7.1: PUBLISH /sensors/sensor1 data.json
7.1.1: /sensors/sensor1 data.json
7.1.1.1: compute()
8: PUBLISH /analytics/tool1 results.json
8.1: /analytics/tool1 results.json
8.1.1: compute()
9: PUBLISH /coach result.json
9.1: /coach result.json
9.1.1: compute()
10: notify()

Figure 19 Sequence diagram of the interaction of the NESTORE modules with the uAAL buses

In the proposed approach, all the modules connected to the NESTORE/uAAL buses can be reached by a dedicated handler acting as entry point for communication allowing the hot deploy of new modules at runtime. This approach has proven to be reliable and functional to different scenarios like social robotics [4][5] and exergaming [6][7] in other EU funded projects [web13][web14] dealing with the interoperability with the universAAL platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

5.4. Application Integration

For NESTORE inter-application communication / integration the RESTFull API over HTTPS where selected as preferred technology. Based on the NESTORE requirements application integration will take place at levels based on classification proposed by Linthicum [2].

- Data level: integrate applications that basically consists on moving data between different data sources: extracting data from its source, processing it conveniently and moving it to another data source. The main characteristics of this level are that it does not require many changes on applications and exchanging and transforming data is relatively less expensive that the other three levels. However, business logic is still enclosed in the primary application (the one that holds the initial data source) thus reducing real-time transactions

- Application interface level: This integration level focuses on application interoperability through sharing of common business logic which is exposed by means of a predefined programming interface. It is usually implemented to expose the interface from packaged, or custom, applications to consume their services or retrieve their data.

- Method level: Also called business integration level [3], this level comprises the business logic within the organization that applications may share, such as: services to access shared data, security, and underlying rules.

- User interface level: This level is dedicated to creating standardized interfaces (usually browser-based) to provide a single interface for a set of applications (legacy). Despite integrations at this level supposes highly coupling with applications, which also means higher maintenance costs, it is also the easiest integration to implement and has significant importance, since it ensures a consistent and effective user experience, especially with legacy applications

On the NESTORE we identify next level of integration that correspond to the four levels defined:

- User interface level: The integration objective for this level is that all NESTORE applications follow the same user interface design rules. This ensures that the end-users experience the same “look and feel” across all NESTORE applications, and can easily identify these applications as a result of the NESTORE project;

- Data level: Common data formats to store NESTORE objects (Knowledge structures is defined on the T2.4 and implemented on the T6.4) are identified, agreed and implemented in the integration platform.

- Method level: This level requires next implementation aspects:

  - the integration platform providing common functions for accessing a central repository via web services

  - NESTORE applications are implemented using these Web services.

  - Web Services open the interoperability with other software components and projects

  - NESTORE – uAAL bridge will open the interoperability with other uAAL solutions

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 769643

- Application Level Integration: This level encloses sharing of business and logic between integrated tools, as well as a user interface integration that serves as a single-entry point to the system
    - to integrate at application level, all NESTORE applications expose integration APIs as Web services; Web services follows the guidelines provided by the integration platform. In future can be implemented a service discovery mechanism enabling it to extend homogenously its services as new tools register their APIs.
    - the User Interface Integration ensures that tools have common UI front-end.

The acronym API comes from Application Programming Interface. An API is a set of functions and procedures that fulfill one or many tasks for being used by other software. It allows to implement the functions and procedures that conform the API in another program without the need of programming them back. As RESTful systems usually communicate with the Hypertext Transfer Protocol (HTTP), a REST API is a library based completely on the HTTP(S) standard. It is used to add functionality to a software somebody already owns safely. The functionality of an API is usually limited by the developer so no more functionality can be added.

The Hypertext Transfer Protocol (HTTP) is a stateless application-level request/response protocol that uses extensible semantics and self-descriptive message payloads for flexible interaction with network-based hypertext information systems (RFC 7230.2014).

In a RESTful system, clients and servers negotiate the representations of resources via HTTP. RESTful systems apply the four basic functions of persistent storage, CRUD (Create, Read, Update, Delete), to a set of resources. In terms of the HTTP standard, those actions can be translated to the HTTP methods (also known as verbs): POST, GET, PUT, and DELETE. Other HTTP methods that are also used but not as often as the formers are OPTIONS, HEAD, TRACE, PATCH and CONNECT.

The communication between NESTORE components is made through representations of resources. For NESTORE API, the format of those representations where selected as JSON. JavaScript Object Notation (JSON) is a lightweight data-interchange format. It was derived from the ECMAScript Programming Language Standard (ECMA-404 2013; RFC 7159.2014). JSON structure can be defined as a ordered list of objects (array of objects). These objects are a collection of name/value pairs separated by colons (:). The utilization of JSON instead of other standard format like XML is due to its simplicity. Both XML and JSON are human readable, but JSON does not need closing tags and is easier to read and is less dense.

On the T6.3 and T6.5 a guide line for creating new Web Services API will be released for having a common description of the resources.

Using a REST-based approach to implement a Web-service integration platform can acquire multiple advantages:

- being lightweight: REST-based platforms built over HTTPS have almost all that is needed to process messaging between applications; no additional protocols nor toolkits are needed, thus improving efficiency and time spent developing application interfaces;
- being useful for fast deployments or first prototypes: creating and deploying REST-based Web services can be quickly and ready to be consumed; shared resources are available almost immediately and they can be easily consumed by any application
- interoperability: if applications are able to access a URL and understands the resource semantics, it does not matter in which platform or system they are deployed

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

- natural use of the Web: REST is completely embedded in the World Wide Web, thus, a REST-based platform over HTTPS provides the benefits of scalability, which means supporting many applications accessing services at the same time;
- interoperation and functionalities sharing of intelligent applications: more complex intelligent applications can be created, while keeping them decoupled, thus allowing them to evolve without negative effects on other applications.

6. Conclusions

We outline here conclusions categorized on:

A. Non-technical

B. Technical and

C. NESTORE & uAAL.

6.1.A) Non-Technical

At the time that NESTORE proposal where written the uAAL platform promise to benefit from a wide user's support, at that time different EU projects where supporting the uAAL and persons as well organizations behind uAAL promise to invest more on the platform.

The uAAL solution is based on (very) big code base, complex architecture (potential for bugs, misconfigurations). The community on GitHub is small (4 users at the date of our investigation) and the dynamic in the code support is low; latest release become old – Release 3.4.0 from 09.2015. We also search for different reference of usage of the uAAL on different research as well commercial projects, therefore we found past reference but no actual ones.

License: The uAAL is released under Apache2 open source license, unfortunately there are code sections that are licensed under some proprietary license such as below [web 6]:

```
/**
 * This code was edited or generated using CloudGarden's Jigloo SWT/Swing GUI
 * Builder, which is free for non-commercial use. If Jigloo is being used
 * commercially (ie, by a corporation, company or business for any purpose
 * whatever) then you should purchase a license for each developer using Jigloo.
 * Please visit www.cloudgarden.com for details. Use of Jigloo implies
 * acceptance of these licensing terms. A COMMERCIAL LICENSE HAS NOT BEEN

 * PURCHASED FOR THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED LEGALLY FOR

 * ANY CORPORATE OR COMMERCIAL PURPOSE.

 */
```

While this is in sample code (the reason why it was observed!) there are no guarantees other part of the code (more important) are not similarly impacted.

6.2.B) Technical

uAAL define themselves as: "In the core of universAAL IoT is a middleware that understands data and functionality only through the definition of ontologies." They are not offering anymore a platform but a middleware from where each project can start building their own technical platform blended with ontologies objects.

As presented before, nowadays, uAAL is not supported as promised by large number of projects and users. The technical investment to the middleware is significant, some technologies are obsolete taken in consideration the fact that NESTORE solution will be released on 2019 (we need up to date components that have support for +3 years). We outline few of the issues:

- Migration to Java 10 (actual support of java is obsolete[web9]), fall 2018 will see Java 11 LTS so current solution should work at least with the current Java 10 [web7]
- Migration to actual Karaf server [web10] – Karaf version 2 is not supported anymore by the Apache[web8]; even current self-updated Karaf 3 is not active anymore; the target should be the current stable 4.2 which supports Java 10 [web11]
- Karaf 4.2 supports OSGi 6 while 2.x and 3.x are on OSGi 5; update (and usage of the new features) to OSGi6 should be done in order to consider it an up-to-date software
- Communication between uAAL nodes is constrained due to the implemented architecture using multicast network. Multicast network involves complex setup, might not work for all scenarios (is disabled in most clouds, AWS does not support it natively https://aws.amazon.com/vpc/faqs/ it requires additional solutions and configurations); It also add dependency on infrastructure hardware (capabilities) and setup (enabled/disabled). This reduce dramatically the usage on the NESTORE use cases, where are involved aged users with limited technical skills.

For usage on an IoT scenario we find out that are other open source IoT platforms that are offering basic features that with uAAL must be implemented from scratch. Such features are:

- Device management – register device, manage, and push acquired date to the cloud
- Support of basic drivers – to connect end user devices to the cloud, drivers are needed to support the low-level communication – i.e. Bluetooth protocol, and these drivers are consuming huge effort to create and maintain. Selection of a IoT platform that provide up to date drivers is a must for NESTORE.
- We did not find any gateway producer to provide hardware gateway that support uAAL. For other platforms like home-automation.io are such providers.

uAAL application must be all written as uAAL bundle and this create a huge impact on NESTORE. Due to this limitation we cannot integrate, or is extremely costly, existing open source modules i.e. chatbot's or other AI modules/algorithms.

6.3.C) NESTORE & uAAL

NESTORE objective is to deliver the Coach activities; to do this the uAAL platform was selected as candidate to be used as support platform. It is not the NESTORE project scope to create / define a new IoT platform but to use an existing one.

Using uAAL to deliver NESTORE Coach(es) requires investing a considerable effort into uAAL platform (to update to latest java version as well latest Karaf and used libraries). It was found challenging to simply build the uAAL from the GitHub sources, due to the complexity of the environment and choosing of default configuration (unit test crash, additional small changes needed). Even for the samples published on the GitHub we found out that because of the source code consistency with available binaries from uAAL maven repository it's not possible to just run them. You need to adapt, change, build from sources this make even harder to adopt the uAAL without substantial effort unaccounted in the project proposal. Some parts of online documentation were not updated and synchronized with the actual source code.

All the NESTORE coaching applications are involving different subsystems that partners developed with different programming languages that are not compatible with uAAL requirements (only Java or something compiling to JVM bytecode). For AI modules the Python or Go is used, for device management is used PHP / NodeJS, etc.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

7. References

[web 1] https://ercim-news.ercim.eu/en87/special/universaal-an-open-platform-and-reference-specification-for-building-aal-systems

[web2] http://www.cip-reaal.eu/about/the-platform/

[web3] https://www.osgi.org/

[web4] mqtt.org

[web6] Jigloo reference:

https://github.com/universAAL/samples/blob/master/lighting/client/src/main/java/org/universAAL/samples/lighting/client/LightClient.java#L38

[web7] Java SE Support Roadmap: http://www.oracle.com/technetwork/java/javase/eol-135779.html

[web8] Karaf versions: http://karaf.apache.org/download.html with release notes

https://github.com/apache/karaf/blob/master/RELEASE-NOTES

[web9] Java 1.5 as target in middleware

https://github.com/universAAL/middleware/blob/5d65c438bcd8d20697a1982b13c2ecba74cf3c7b/pom/pom.xml#L77

[web10] Karaf 2.2.9 used in latest release

https://github.com/universAAL/distro.karaf/releases/tag/3.3.0

[web11] Various universAAL on karaf info https://github.com/universAAL/distro.karaf/wiki/Running-universAAL-in-Karaf

[web12] https://github.com/universAAL/platform/wiki/About-universAAL

[web13] https://cordis.europa.eu/project/rcn/101840_it.html

[web14] https://cordis.europa.eu/project/rcn/110829_en.html

[1] M.-R, Tazari & F, Furfari & Valero Á, Fides & Hanke, Sten & Höftberger, Oliver & Kehagias, Dionisis & M, Mosmondor & Wichert, Reiner & P, Wolf. (2012). The universal Reference Model for AAL. Handbook f Ambient Assisted Living. 11. 610 - 625. 10.3233/978-1-60750-837-3-610.

[2] Linthicum, D.S. Enterprise Application Integration. Essex: Addison-Wesley Longman Ltd., 2000. 0-201-61583-5.

[3] A Model Driven Architecture for Enterprise Application Integration. Al Mosawi, A., Zhao, L. and Macaulay, L. s.l. : IEEE Computer Society, 2006. Proceedings of the 39th Hawaii International Conference on System Sciences. p. 181.3. 0-7695-2507-5.

[4] Palumbo, Filippo, et al. "Sensor network infrastructure for a home care monitoring system." Sensors 14.3 (2014): 3833-3860.

[5] Coradeschi, Silvia, et al. "Giraffplus: Combining social interaction and long term monitoring for promoting independent living." Human system interaction (HSI), 2013 the 6th international conference on. IEEE, 2013.

[6] Bacciu, Davide, et al. "Detecting socialization events in ageing people: the experience of the DOREMI project." Intelligent Environments (IE), 2016 12th International Conference on. IEEE, 2016.

[7] Bacciu, Davide, et al. "Smart environments and context-awareness for lifestyle management in a healthy active ageing framework." Portuguese Conference on Artificial Intelligence. Springer, Cham, 2015.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

8. Annexes

8.1. Annex 1

Proof of concept support diagrams

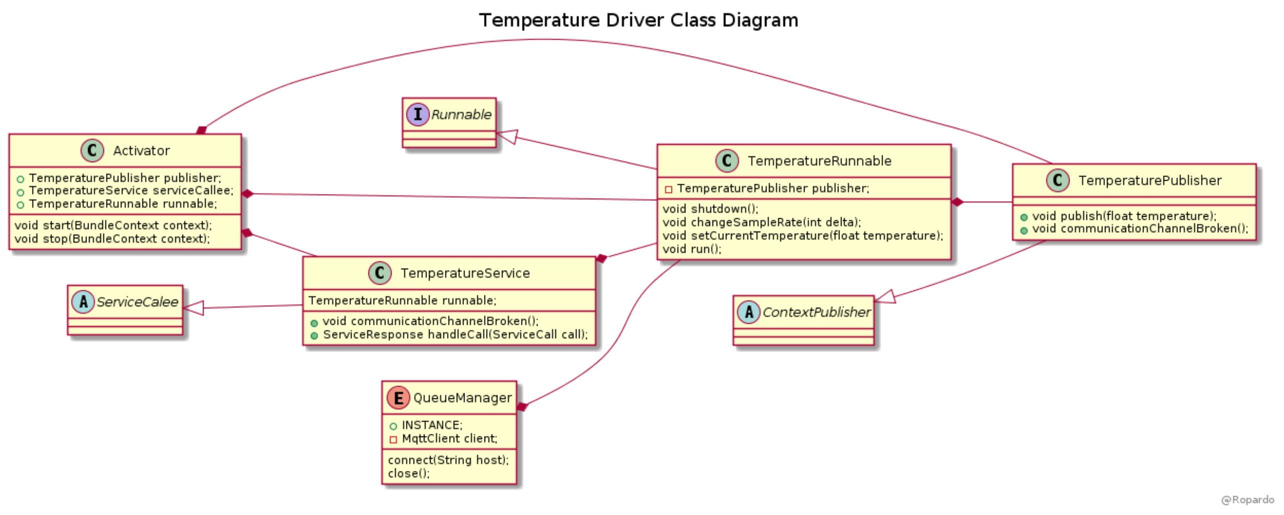

Temperature Driver Class Diagram
Runnable
Activator
TemperaturePublisher publisher;
TemperatureService serviceCallee;
TemperatureRunnable runnable;
void start(BundleContext context);
void stop(BundleContext context);
TemperatureRunnable
TemperaturePublisher publisher;
void shutdown();
void changeSampleRate(int delta);
void setCurrentTemperature(float temperature);
void run();
TemperaturePublisher
void publish(float temperature);
void communicationChannelBroken();
ServiceCalee
TemperatureService
TemperatureRunnable runnable;
void communicationChannelBroken();
ServiceResponse handleCall(ServiceCall call);
ContextPublisher
QueueManager
INSTANCE;
MqttClient client;
connect(String host);
close();
@Ropardo

Figure 20 Driver offer interface with temperature sensor

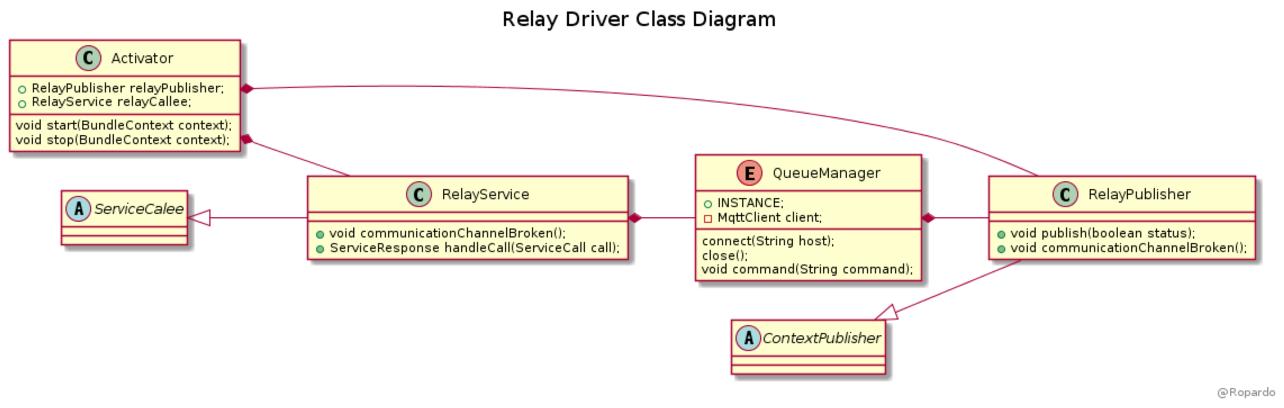

Relay Driver Class Diagram
Activator
RelayPublisher relayPublisher;
RelayService relayCallee;
void start(BundleContext context);
void stop(BundleContext context);
ServiceCalee
RelayService
void communicationChannelBroken();
ServiceResponse handleCall(ServiceCall call);
QueueManager
INSTANCE;
MqttClient client;
connect(String host);
close();
void command(String command);
RelayPublisher
void publish(boolean status);
void communicationChannelBroken();
ContextPublisher
@Ropardo

Figure 21 Driver offer interface with relay (actuator)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

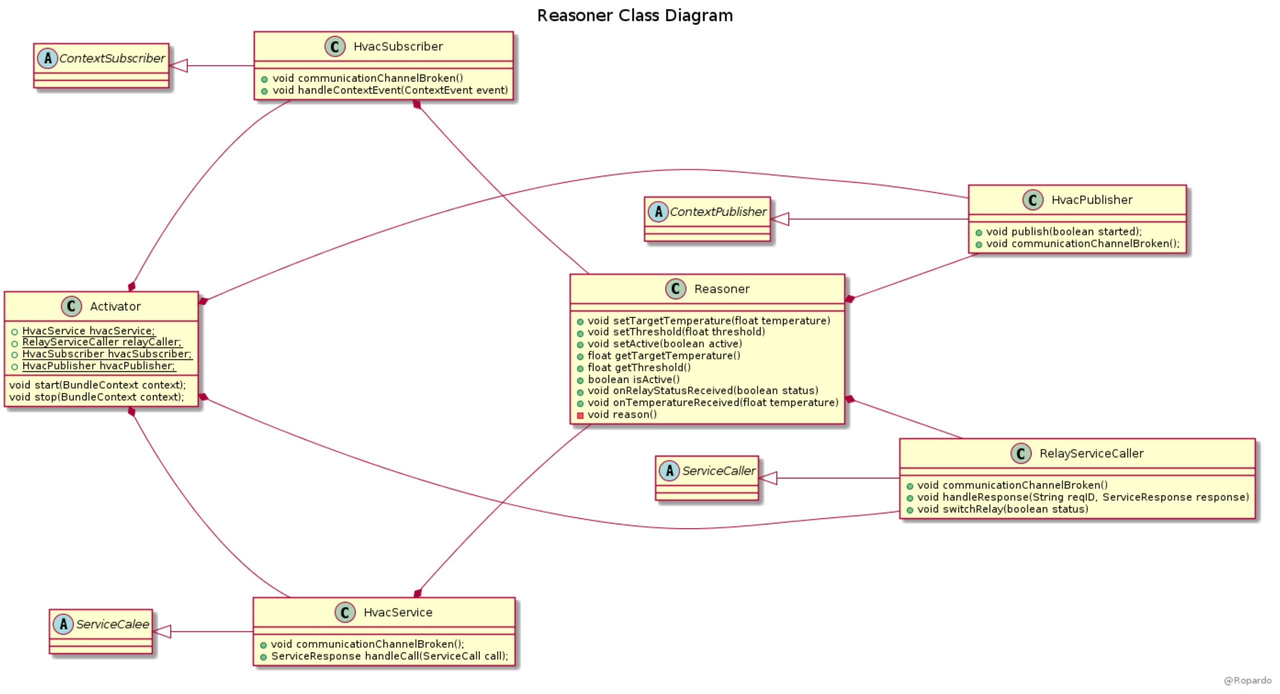

Reasoner Class Diagram
ContextSubscriber
HvacSubscriber
void communicationChannelBroken()
void handleContextEvent(ContextEvent event)
ContextPublisher
HvacPublisher
void publish(boolean started);
void communicationChannelBroken();
Activator
HvacService hvacService;
RelayServiceCaller relayCaller;
HvacSubscriber hvacSubscriber;
HvacPublisher hvacPublisher;
void start(BundleContext context);
void stop(BundleContext context);
Reasoner
void setTargetTemperature(float temperature)
void setThreshold(float threshold)
void setActive(boolean active)
float getTargetTemperature()
float getThreshold()
boolean isActive()
void onRelayStatusReceived(boolean status)
void onTemperatureReceived(float temperature)
void reason()
ServiceCaller
RelayServiceCaller
void communicationChannelBroken()
void handleResponse(String reqID, ServiceResponse response)
void switchRelay(boolean status)
ServiceCallee
HvacService
void communicationChannelBroken();
ServiceResponse handleCall(ServiceCall call);
@Ropardo

Figure 22 Reasoner observes the "world" and can take various decisions based on its rules programming

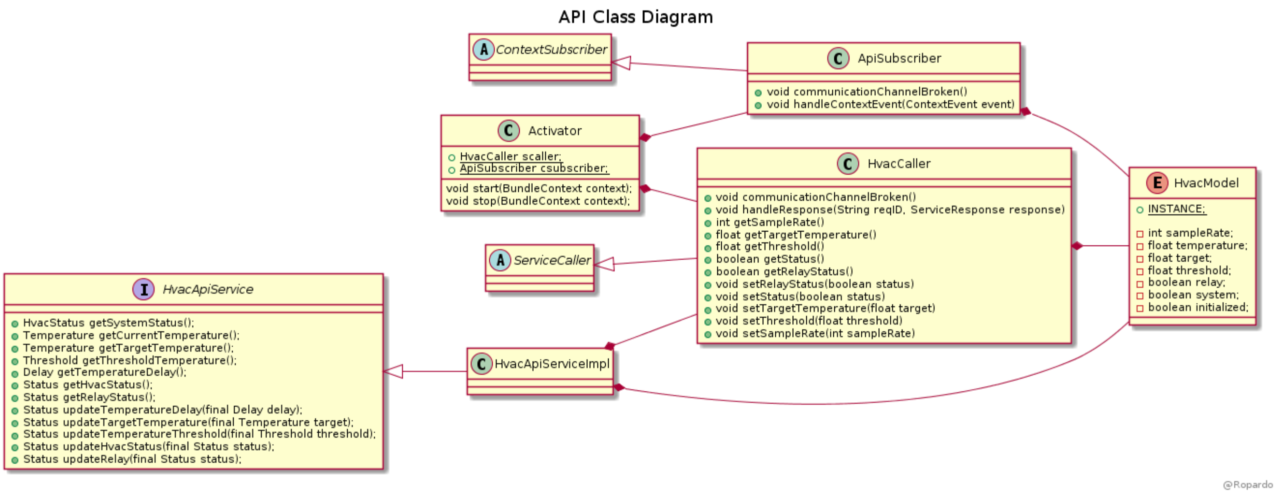

API Class Diagram
ContextSubscriber
ApiSubscriber
void communicationChannelBroken()
void handleContextEvent(ContextEvent event)
Activator
HvacCaller scaller;
ApiSubscriber csubscriber;
void start(BundleContext context);
void stop(BundleContext context);
HvacCaller
void communicationChannelBroken()
void handleResponse(String reqID, ServiceResponse response)
int getSampleRate()
float getTargetTemperature()
float getThreshold()
boolean getStatus()
boolean getRelayStatus()
void setRelayStatus(boolean status)
void setStatus(boolean status)
void setTargetTemperature(float target)
void setThreshold(float threshold)
void setSampleRate(int sampleRate)
HvacModel
INSTANCE;
int sampleRate;
float temperature;
float target;
float threshold;
boolean relay;
boolean system;
boolean initialized;
ServiceCaller
HvacApiService
HvacStatus getSystemStatus();
Temperature getCurrentTemperature();
Temperature getTargetTemperature();
Threshold getThresholdTemperature();
Delay getTemperatureDelay();
Status getHvacStatus();
Status getRelayStatus();
Status updateTemperatureDelay(final Delay delay);
Status updateTargetTemperature(final Temperature target);
Status updateTemperatureThreshold(final Threshold threshold);
Status updateHvacStatus(final Status status);
Status updateRelay(final Status status);
HvacApiServiceImpl
@Ropardo

Figure 23 It offers a REST/JSON standard interface over HTTP to integrate with external users

8.2.Annex 2

Partner' feedback about UniversAAL adoption within NESTORE Project.

8.2.1. NEOS

WP6 enlisted a task named 6.1 to learn and understand UniversAAL platform and the way it can support NESTORE architecture development. After the deep analysis conducted by Ropardo team, some key points have been found relevant to understand the involvement of such technology.

First of all there are some strong concerns about UniversAAL documentation and project roadmap, since there hasn't been a wide adoption among community. Moreover, it is not clear whether any integration task could be developed by UniversAAL development group, supporting NESTORE partners within the strict time frame planned by the project.

In addition, some strong costraint from an architectural point of view raised more concerns:

- when installed on devices, has been found that there is no support for a wide range of nodes, so a communication mechanism has to be put in place such as an MQTT channel between home IoT nodes and a central hub running UniversAAL
- authentication of home central hub has to be managed outside UniversAAL because there is no device provisioning provided by the framework
- UniversAAL relies on outdated libraries (provided by Apache foundation) and it is not up to date with latest releases, so a full recompile of project from source code is needed to include bug fixes and increase support
- there is no support for communication errors or retrieval: developer has to assure the node is in a consistent state and handle any error to fail back whenever an error occurs

In conclusion we suggest splitting NESTORE architecture from UniversAAL message bus and adopt a different platform to handle nodes and support the widest adoption. Then, an extension to NESTORE can be obtained leveraging UniversAAL as an integration layer within NESTORE platform, assigning this task to UniversAAL group, under strict timing since they have the knowledge about their system and can effectively translate ontology and data from NESTORE data mode to UniversAAL model.

8.2.2. TUD

TUD is responsible for the creation of (serious) game(s) that accompany the NESTORE coach. The goal of the game(s) is to foster engagement with the coach and to stimulate well-being in the various pathways where possible. For development, we will use the game engine Unity3D. Data exchange will take place between the coaching app and the game through the use of a database (e.g. MongoDB or MySQL, depending on decisions of partners). For example, activities performed by the user might be recognized by the game and generate a reward. Similarly, in-game activities might be noticed by the coach and considered part of the user's pathway progress. While it is possible to explore integration of the eventual wearable or other (sensor) technology in the game, data exchange is mainly planned to happen between the game and the coaching application. For this reason, TUD has no clear preference regarding the use of uAAL. Being a framework for easier communication between devices (IoT), TUD will most likely not be using the functionality it provides. We therefore defer to the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

other partners regarding the decision to use the platform. Should uAAL be adopted in NESTORE, TUD will adapt where needed to accommodate this decision

8.2.3. HES-SO

HES-SO has evaluated the opportunities offered by the uAAL framework concerning the development of the interfaces for the Virtual Coach, as described in the grant agreement (WP5). The User Interface bus of the uAAL framework has been discontinued by the uAAL coalition, probably due to the obsolescence of the proposed protocols and supported UIs. As such, the NESTORE interface implementation has started without considering the uAAL UI bus as an integral part of the system for any message that is related to the interaction between the user and the NESTORE coach. Supporting the uAAL UIs bus and conforming to the desired user experience requirements would require at present a complete change of the interface architecture and of the first prototype implemented so far (a cross-platform app and chatbot), and an extra development effort that is not considered in HES-SO budget. Moreover, according to the tests performed by Ropardo, it seems unlikely the possibility to run the OSGi stack on the user personal devices, because of: the strict requirements and the complexity of the software stack. Similar constraints may apply for NESTORE devices that require direct interaction with the users (e.g., the tangible coach).

However, HES-SO recognizes the interest in making the NESTORE architecture compliant with uAAL, ensuring data exchanges between projects (and, in particular, services and devices) inside the uAAL coalition, provided that the required architectural adaptations would not affect the performance and the user experience of the NESTORE system. Therefore, HES-SO encourages the adoption of a lightweight integration of the uAAL platform, bridging data between the NESTORE system and other services and devices supporting the uAAL framework.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643