PROJECT
Coach assistant via projected and tangible interface
GRANT AGREEMENT
Nr. 769830

D3.5 – Data Broker

SUBMISSION DUE DATE	DELIVERABLE VERSION
Month 14, 31.01.2019	3.0
ACTUAL SUBMISSION DATE	
Month 14, 31.01.2019	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769830

DELIVERABLE TITLE	Deliverable title
DELIVERABLE No.	D.3.5
Deliverable Version	3.0
Deliverable Filename	Captain_Deliverable_3.5_DataBroker_v3.0.docx
Nature Of Deliverable	R = Report
Dissemination Level	Public
Number Of Pages	26
Work Package	WP3. CAPTAIN appliance hardware prototyping and low level software infrastructure development
Partner Responsible	NIVELY
Author(s)	Evdokimos Konstantinidis (NIVELY), Giuseppe Conti (NIVELY) Michalis Timoleon (AUTH), Despoina Petsani (AUTH), Andoni Beristain (VICOM), Francesco Verrini (NIVELY)
Contributor(s)	
Editor	Evdokimos Konstantinidis (NIVELY)
Reviewed by	Alejandro Rivero Rodríguez (SAL), Guillaume Chican (HOL)
Approved by	Panos Bamidis, Project Coordinator

PROJECT FULL TITLE	Coach assistant via projected and tangible interface
Type Of Action	Research & Innovation Action (RIA)
Topic	H2020-SC1-PM-15-2017: Personalised coaching for well-being and care of people as they age
Start Of Project	1 December 2017
Duration	36 months
Project URL	www.captain-eu.org

Table of Contents

LIST of Acronyms

Acronym	Description
CTO	Chief Technology Officer
API	Application programming interface
IoT	Internet of Things
GSMA	Groupe Spéciale Mobile) Association
CEP	Complex Event Processing
Phr	Personal Health Record
GDPR	General Data Protection Regulation

LIST OF FIGURES

1 EXECUTIVE SUMMARY

The goal of this deliverable is to present all the extension for the brokering mechanisms that have been and will be developed by CAPTAIN in order to increase the levels of interoperability with other systems as well as the value proposition towards its target group. The initial plan of this deliverable, as defined during the project proposal document, was to adopt the UNCAP toolkit, which resulted as the outcome of the UNCAP H2020 project, coordinated by Giuseppe Conti (today the CTO of NIVELY). The UNCAP toolkit was comprehensive software framework designed to maximise interoperability between different biosensors (e.g. pulse-oximeters, scales, blood pressure monitor, blood glucose level monitor, etc.). The framework strongly focused on support for data exchange standards (e.g. Open mHealth, etc.). However, the framework did not support any low-level standards, since it relied on a customised publish-subscribe architecture based on the Raptorbox library (https://create-net.fbk.eu/openiot/assets/) developed by FBK, partner of UNCAP. Since the general UNCAP platform is not being maintained any longer, the technical team of CAPTAIN has decided to opt a future-proof standard-based platform as underlying technological platform, namely FIWARE and openHAB. Instead, the consortium has still decided to capitalise on achievements of UNCAP in terms of data exchange protocols and specific technologies developed during the project, most notably the MentorAge system by Nively which was one of the offsprings of UNCAP and which is used for CAPTAIN "satellite" devices.

On the other hand, CAPTAIN has adopted an agile methodology of development and user requirements collection. This choice has been made in order to be able to easier adapt to new changes and technological achievements, by focusing on achieving high user acceptance levels, something that is often regarded as one of the biggest challenges for all the EU projects that deliver technology tailored to older adults. In relation to the two aforementioned factors, and in order to maximise use of resource within the project, the technical team has selected two additional well-known frameworks upon which other high-value services will be developed, namely:

1. FIWARE Orion and related FIWARE data models, which has been adopted as underlying platform to ensure extensibility of the CAPTAIN cloud server
2. OpenHAB, which has been selected to allow local (inside the home of the older adult's home) extensibility.

The deliverable provides an overall picture of the CAPTAIN extendibility by providing a description of its high-level architecture, where by CAPTAIN is regarded as a system, which is extended by the aforementioned frameworks. Finally, this deliverable provides technical information on the integration implementation, both at the server and local side.

2 INTRODUCTION

The aim of this deliverable is to present the data brokers that CAPTAIN bridges with to ensure higher levels of interoperability and more chances for integration with other systems in the future. To do so, CAPTAIN extends its functionality in 2 levels: the local CAPTAIN system and the CAPTAIN Cloud server. OpenHAB was selected as the data broker with which the CAPTAIN Box (local, in home) will be able to get access to a variety of home automations, while FIWARE Orion was selected as the data broker with which the CAPTAIN Cloud will be able to get access to information pertaining not to a specific home or user but to the users' broader environment. Both frameworks are well established on the domain, constantly attracting new researchers and technology providers.

The open standard nature of FIWARE NGSI offers programmers the ability to port their applications across different "Powered by FIWARE" platforms and a stable framework for future development. Additional functionality can be easily added to a Smart Solution simply by using additional FIWARE or third-party components for which the integration with the FIWARE Context Broker component is solved. This integration is simplified since all components comply to the FIWARE NGSI standard interface, which eliminates vendor lock in. The component-based nature of a FIWARE based solution allows for re-architecting as the solution evolves according to business needs. In this line Captain Server cloud provides FIWARE Orion Context broker as a bridge for other networks to provide useful information to Captain Box users. Such information data can be environment data (air quality, water quality etc.), transportation data, weather data, government announces etc.

Moreover, openHAB framework is leading software is the field of home automation that can allow protocol and platform agnostic integration of various devices into the home automation system. Concerning the existing literature about home assistants for older adults, automatized application are frequently a must. In this notion CAPTAIN integrates openHAB in the system providing the opportunity to enable various home automation from open the lights to preventing flooding and fire.

Both FIWARE and openHAB come with instruction about their installation and initial configuration. In the case of CAPTAIN through, specific configuration or event bridging applications were required as CAPTAIN implements its own communication protocols (based on the data models of FIWARE and the MQTT publish subscribe framework which is adopted by openHAB). More specifically, the CAPTAIN Cloud API was developed further to support the communication with FIWARE, supporting also a callback function (RESTful subscription to entities' changes). On the other hand, the openHAB was configured to communicate on top of the CAPTAIN's MWTT framework which is used for the communication on the local level (CAPTAIN Box and Satellites). Similarly to the FIWARE, a couple of functions were developed to ensure the UNCAP system integration, especially on the user management and authentication level.

This remainder of this deliverable is structured as follows: Section 3 presents briefly the 3 frameworks, FIWARE, openHAB and UNCAP, highlighting the main components, data models, interoperability aspects and the currently supported devices for each. Following, Section 4, Section 5 and Section 6 provide information on the development that was carried out for bridging the 3 frameworks with CAPTAIN, pointing out the installation and configuration followed for each system as well as the design approach that was followed for each framework. Finally, the conclusions section (Section 7) puts everything together and presents the concluding remarks of the deliverable.

All the information presented in this document related to UNCAP H2020 project has been retrieved by the publicly open deliverables of the project, available at http://www.uncap.eu/downloads. The majority of the information for UNCAP comes from the deliverables "D.1.3 – System Architecture - v2.0" and "D.2.3 – UNCAP Box, App and CLOUD – final version".

All the information presented in this document related to openHAB has been retrieved by the publicly available official documents and web site of openHAN at https://www.openhab.org.

All the information presented in this document related to FIWARE and FIWARE Orion has been retrieved by the publicly available official documents and web site of FIWARE at https://www.fiware.org and FIWARE Orion at https://fiware-orion.readthedocs.io.

3 DATA BROKERING AND EXTENDABILITY ON THE CLOUD AND LOCALLY

The architecture of CAPTAIN has been designed pursuing the following macro objectives:

- ensure modularity;
- deliver high efficiency;
- maximise interoperability.

These three requirements are key to support maximising horizontal (i.e. across components) communication among sensors and cloud services. The overall CAPTAIN system is composed of three main components which, in turn, encapsulate a set of micro-services and components:

1. The CAPTAIN Cloud,
2. The CAPTAIN Box and
3. The CAPTAIN Satellite components.

The **CAPTAIN Cloud** is a set of server-based solutions on top of which the CAPTAIN Cloud API is hosted. The API exposes all the features needed for the proper functioning of the individual CAPTAIN installations. These may range from User Management and User Profiling to more demanding processing features for analysis of the collected data.

The **CAPTAIN Box** is the heart of CAPTAIN at each the local side, that is the home of the older adult. In a typical setup, at least one box per home is provided which is responsible to communicate and synchronize data with the CAPTAIN Cloud. The box can be thought as a local gateway providing all the services require for communication at local level, including pub/sub channels, allowing local devices to communicate and interact. At the same time, the box acts as a local processing unit (i.e. for image analysis) and it also delivers part of the projection functionality required by CAPTAIN.

Lastly, the CAPTAIN Satellite's functionality concentrates mainly on monitoring the user, through use of 3D sensors, as well as projecting images on the walls of the user's home.

In particular, CAPTAIN has been designed to ensure high level of interoperability with existing and future systems. To do so, CAPTAIN adopts the two well-established frameworks: FIWARE to make its Cloud expandable and openHab to make the local installation (each home) expandable. Figure 1 illustrates how the current CAPTAIN architecture (described in D2.2 First Version of System Specification) is extended on the Cloud and local side. The three differently colored arrows (blue, green and red) represent the three different bridges that were developed and presented in this deliverable.

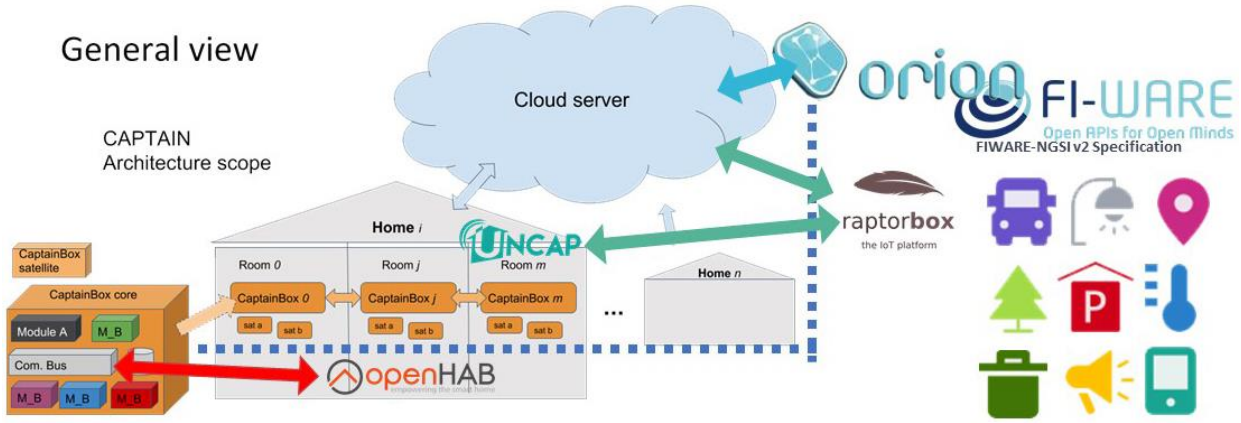


Figure 1 the cloud-level scope of CAPTAIN Architecture, extended with the integration of Fi-Ware, openHab and UNCAP (based on Figure 1 of the D2.2 First Version of System Specification).

Figure 2 zooms in to the local part of the CAPTAIN system: the CAPTAIN Box and the CAPTAIN Satellite. To avoid complexity and to conform as much as possible with the third-party frameworks' recommendations, the openHAB is integrated by being installed on a Raspberry PI 3(https://www.openhab.org/download) and connected to the same local network with the CAPTAIN Box and CAPTAIN Satellites.

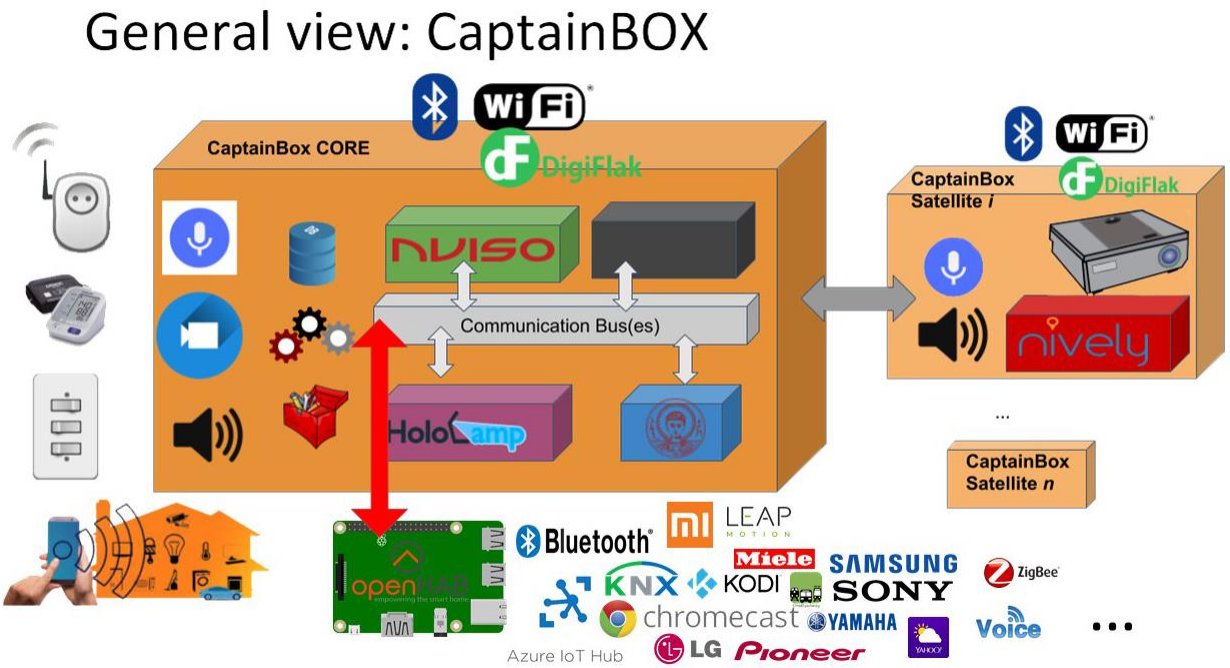


Figure 2 a "local" level view of the "CAPTAIN box" (Core and Satellite) which can be further extended via openHab (based on Figure 3 of the D2.2 First Version of System Specification).

3.1 FIWARE

3.1.1 Presentation of FIWARE and Orion Context Broker's role

FIWARE is a curated framework of open source platform components which can be assembled together with other third-party platform components to accelerate the development of **Smart Solutions**.

According to the official website (https://www.fiware.org/), a Smart Solution is a solution where there is a need to gather and manage context information, to process that information and to inform external actors, enabling them to actuate and therefore alter or enrich the current context.

The FIWARE Foundation is the legal independent body providing shared resources to help achieving the FIWARE mission by promoting, augmenting, protecting, and validating the FIWARE technologies as well as the activities of the FIWARE community, empowering its members including end users, developers and rest of stakeholders in the entire ecosystem. The FIWARE Foundation is open: anybody can join contributing to a transparent governance of FIWARE activities and rising through the ranks, based on merit. Among its main members there are companies such as Atos, Engineering, NEC, Orange or Telefónica.

It also has a strong community behind that gather web entrepreneurs, mentors, investors, students, academia, industry and public-sector innovators to keep progressing as well as a strong online presence in the main relevant sites.

The main and only mandatory component of any "Powered by FIWARE" platform or solution is **the FIWARE Orion Context Broker Generic Enabler**, which brings a cornerstone function in any smart solution: the need to manage context information, enabling to perform updates and bring access to context in a highly decentralized manner.

The Orion Context Broker is an implementation of the Context Broker GE, providing an NGSI interface. Using this interface, clients can do several operations:

- To register context producer applications (e.g. a temperature sensor within a room).
- To update contextual information (e.g. send updates regarding indoor temperature).
- To get notified when changes on key conditions related to contextual information (e.g. the temperature has changed) or with a given frequency (e.g. to retrieve the temperature each minute).
- To query context information and to do so, the Orion Context Broker stores context information updated from applications, so queries are resolved based on that information.

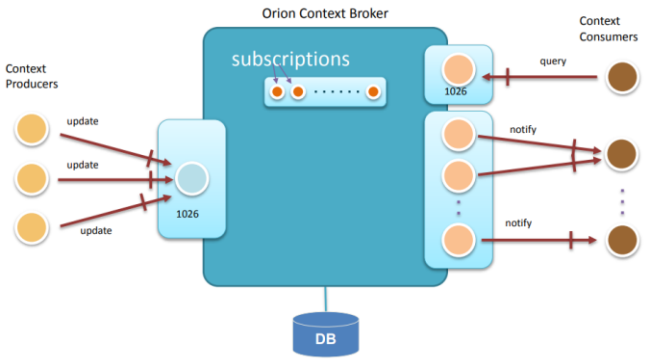


Figure 3 Logical architecture of the Publish/Subscribe Context Broker GE[1]

[1] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.Data.ContextBroker

3.1.2 Main components of Orion context broker

Building around the FIWARE Orion Context Broker, a rich suite of complementary FIWARE components is available, dealing with:

- **Interfacing with the Internet of Things (IoT), Robots and third-party systems**, for capturing updates on context information and translating required actuations.
- **Context Data/API management, publication and monetization**, implementing the expected smart behaviour of applications and/or assisting end users in making smart decisions.
- **Processing, analysis and visualization of context information**, bringing support to usage control and the opportunity to publish and monetize part of managed context data.

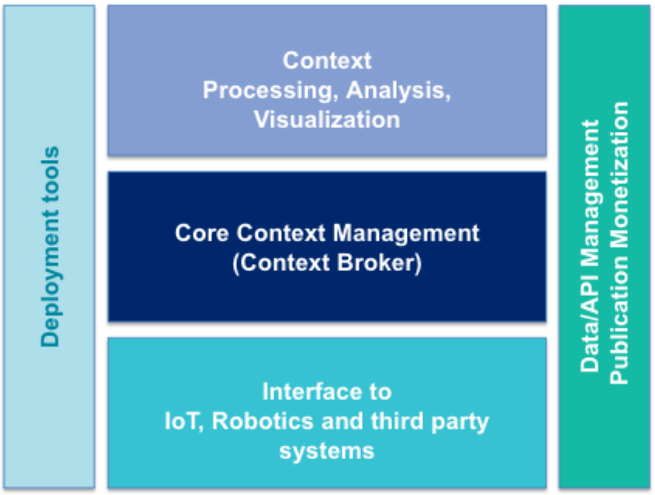


Figure 4 A rich suite of complementary FIWARE components is built around the FIWARE Context Broker[2]

3.1.3 Data models

In CAPTAIN, the key Data Models have been defined in relation to the FIWARE reference data context model (NGSI v2) whose main documentation can be found online at http://fiware.github.io/specifications/ngsiv2/stable/

The general principles guiding their adoption are:

1) To maximise simplicity.
2) To use only the parts of the Data Model strictly required by the application.
3) To use metadata only in case of need, and to follow the assumptions provided in the Data Model.
4) To remain within the logical foundations set by the existing Data Models.
5) While not generally advised, to add additional attributes required by the specific application for the purpose of the application.

In addition, the following conditions should be considered when relying on use of FIWARE:

6) Data Models, in most cases, are NGSI v2 codification of existing data models.

[2] https://www.fiware.org/developers/catalogue

7) JSON Schemas provided cover only the so-called key-value representation of NGSI v2 context data.
8) None of the official FIWARE Core Enablers enforces schema validation.

The main repository that handles the code and specifications to support harmonized data models can be found online at https://github.com/Fiware/dataModels

Currently it hosts 16 Data Model specifications: Alert, Building, Device, Environment, Issue Tracking, Key Performance Indicator, Parking, Parks And Gardens, Point Of Interaction, Point Of Interest, Street Lighting, Transportation, Urban Mobility, User, Waste Management and Weather.

3.1.4 Interoperability

Context representation of Data Models (Section 3.1.3) are meant aid interoperability within the community.

All interactions between applications or platform components of CAPTAIN and the Context Broker will take place by using the FIWARE NGSI RESTful API, a simple yet powerful open standard.

The FIWARE NGSI (Next Generation Service Interface) API defines:

- A data model for context information, based on a simple information model using the notion of context entities.
- A context data interface for exchanging information by means of query, subscription, and update operations.
- A context availability interface for exchanging information on how to obtain context information (whether to separate the two interfaces is currently under discussion).

For context data management, NGSI and the Orion context broker have been accepted as standards or recommendations by a variety of independent standards bodies. For example, GSMA recommends NSGI as a standard for relevant parts of their IoT Big Data architecture and it promotes the Orion Context Broker as the primary example of the standard. Furthermore, NGSI specifications have been selected by the European Commission as a CEF Building Block for the implementation of new smart applications and Public Administration.

3.1.5 Supported devices

FIWARE assembles a large community behind it that comprises, according to its website, more than 1000 startups, 11 Innovative Hubs, 2 acceleration programmes as well as more than 100 cities.

It is also worth mentioning that a FIWARE Marketplace exists with the main purpose of globally disseminating existing commercial offerings around FIWARE. It is a global one-stop shop with the aim of giving visibility to a wide range of Powered by FIWARE solutions/platforms, FIWARE-ready technologies (IoT devices and Software Enablers) as well as FIWARE related training/coaching or consultancy, integration and support services.

3.2 OPENHAB

As described previously, CAPTAIN adopts a participatory, agile approach for the design of the system. To this extend we will include OpenHAB software to our CAPTAIN system in order to achieve local extensibility and enable potential home automation through CAPTAIN system.

3.2.1 Presentation of the OpenHAB Framework

OpenHAB is an open source, technology agnostic automation platform, meaning it can connect to nearly any home automation hardware on the market today, for smart home applications. It is designed to use a pluggable architecture, which means that new devices and protocols can be added easily and it can run almost everywhere (Linux, macOS, Windows, Raspberry Pi, PINE64, Docker etc) providing access with apps for the web, iOS, Android and others. OpenHAB 2 is developed in Java and mainly based on the Eclipse SmartHome framework. It uses Apache Karaf together with Eclipse Equinox to create an Open Services Gateway initiative (OSGi) runtime environment. Jetty is used as an HTTP server. OpenHAB is highly modular software that can be extended through "Add-ons".

Furthermore, OpenHAB does not require any cloud service to work. Its core functionality runs locally, talking directly to local devices whenever possible and keeping data privately. OpenHAB will be installed in the CAPTAIN box in order to enable the communication with external devices based on the demands raised from the CAPTAIN community during the iterative requirement elicitation phase.

OpenHAB supports more than 200 different technologies and systems and more than 1500 supported physical devices, according to their website. Its community is very active with almost 30000 members, and over 353k posts. OpenHAB community has also strong online presence.

3.2.2 Main components of Opehab

OpenHAB requires a Java 8 platform to be installed in the system. OpenHAB will be installed in a Raspberry Pi 3 which will be additional to the CAPTAIN system (CAPTAIN box and satellite) if automation is required by the user. OpenHAB needs a steady power supply in order to run properly while RPI3 needs a micro USB power supply with 5V and at least 2.5 Amps which is included on the KIT. Also, it requires an SD card of 16GB or more and Ethernet connection is preferred.

OpenHAB provides bindings with various market leading devices for home automation which can be easily configured. If a user decides to extend CAPTAIN for home automation any capability can be added by integrating the appropriate device. Through CAPTAIN-OpenHAB configuration, CAPTAIN system can control these added devices.

3.2.3 Concepts

In OpenHAB a collection of Things and Items represent physical or logical objects in the user's home automation setup. Things are the entities that are physically added to a home automation system, either device or web services and other source of information. They can provide more than one function, for example a sensor may do motor detection and measure temperature at the same time. OpenHAB provides the capability of automatically discovering most of the Things connected to the local network. Each thing connected has a status which can be UNINITIALIZED, INITIALIZING, UNKNOWN, ONLINE, OFFLINE, REMOVING, REMOVED.

On the other hand, Items represent capabilities of controlling every device, sensor, or information element of the user’s home automation. Items are basic data types and have a state which can be read from or written to. There are various available Item types such as Color, Contact, DateTime, Dimmer, Location, Switch to name but a few. Items can be linked to a Binding channel for interaction with the outside world. Bindings act as software adapters that link Items to Things. They are provided as add-ons specially developed for each device. In this way openHAB provides the capability of adding several Things (devices or source of information) to the system and control their functionalities using Items.

To do so, Things expose their capabilities through Channels. Each Channel controls one feature of the Items. In the previous example, we are having a channel for controlling the motion detection and another for the temperature.

To control these concepts, OpenHAB provides both graphical UI and programmable capability for configuring the home automation setup. It is possible also to configure a sitemap, which is a UI that enables the user to control the home automation with simple buttons. Furthermore, the developer can write Rules which are scripts where you can define schedules or conditions for an action to happen. Rules define the real automation process.

3.2.4 Interoperability

OpenHAB's main purpose is to embrace interoperability among home automation community by

1. Supporting the integration of different technologies, standards and APIs in a seamless manner and controlled by flexible and easy to define rules, triggers, script and actions.
2. Enabling to control home automation from an easily configured environment almost everywhere (Linux, macOS, Windows, Raspberry Pi, PINE64,Docker, Synology etc.).

OpenHAB provides a binding for the MQTT (Message Queuing Telemetry Transport) protocol, which is used in CAPTAIN architecture for the communication between different components. Any Thing that is connected with the OpenHAB can also be controlled by the CAPTAIN system through MQTT. The main complication with the MQTT protocol is that it does not enforce any topic layout or topic value format and any new thing added to the system can publish to different topics or follow different conventions for value format. This is why the Homie (https://homieiot.github.io) convention will be adopted for topic and format definition in order to assure interoperability.

3.2.5 Supported devices

Currently, OpenHAB supports 323 add-ons and 1547 things. Some of the widely recognised device bindings that are supported include BENQ, DAIKIN, LG, Miele, Xiaomi, Pioneer, Samsung, Sony etc. OpenHAB also supports system integrations in order to expose it to external systems such as Google Calendar integration that allows OpenHAB to control Items at scheduled times in the future. Moreover, Data Persistence services such as MySQL support the storage of time series data for further analysis. In general, OpenHAB is fully customizable and supports multiple ways to achieve your goal.

3.3 UNCAP

3.3.1 Presentation of the UNCAP framework

The main objective of the UNCAP H2020 IA project (No. 643555) (http://www.uncap.eu) was to develop and open, scalable and privacy-savvy ICT infrastructure based on solutions and technologies developed in previous research projects, to help aging people live independently while increasing the quality of life. UNCAP main goals were to:

- Improve effectiveness of the health care processes during the hospital---hospice recovery.
- Enhance home care treatment and prevention towards delaying cognitive decline of elderly.
- Support more independent living and improve quality of life of cognitively impaired older adults.

From a technical point of view, the five pillars of UNCAP were:

- Interoperability, by building on top of open standards, allowing for future extensions in terms of hardware and software.

- Openness, through release of open specifications and open software components.
- Scalability, through use of cloud-centric services.
- User friendliness.
- Privacy and security, through attention to all related privacy and security aspects.

While the overall UNCAP solution is not any more maintained, some of its components have been turned into commercial products and therefore have been further extended. This is the case of MentorAge by Nively, a spin-off of the UNCAP project. The following sub-sections shed a light on the variety of the components, highlight the technical information relevant to CAPTAIN.

3.3.2 Main components of UNCAP

The UNCAP system is composed of two main sub-systems: the UNCAP device and the UNCAP services. The UNCAP device enables the users (older adults and caregivers/nursing stuff) to interact “at home” and “on the go” or through the UNCAP Web Application (management Graphical User Interface accessible through web browser). The UNCAP services accommodates the backend modules as well as the data brokers that ensure interoperability and scalability.

3.3.2.1 UNCAP device and Web Applications

The UNCAP Box architecture was based on a driver management module which managed the installation of new drivers for third party devices (i.e., glucometer), checked for drivers within the UNCAP driver catalogue on the UNCAP Cloud server, and scanned for locally installed drivers. The reminders module was responsible for triggering and delivering reminders across the different apps (e.g., measurement or medicine reminders). The communication Module provided video communication functionality between patients and doctors. The location module exploited information coming either from the GPS receiver or from other location technologies. The red button module generated alarm messages and dispatched them to the message bus. The user and authentication module handled all the user’s authentication and authorization requests. The serious games module provided a range serious games. The message bus service bus managed all communication between internal components. The settings module served all the commonly needed operations, such as driver configuration, user settings, etc. The data receiver service listened for incoming communication from third-party drivers, which was then routed to the message bus and received by the data sync module. The internal database module buffered the data produced by the various medical devices (i.e., measurements, location data, alarms) locally when operating offline. The real-time data display module displayed real-time data of the ongoing measurements from the services while the historical data display module fetched historical data from the cloud and displayed it in the UNCAP device. Finally, the entertainment module provided easy access to any installed Android application (launcher screen).

3.3.2.2 UNCAP Cloud

The UNCAP Cloud consisted of a collection of modules as depicted in Figure 5 which presents an abstract architecture.

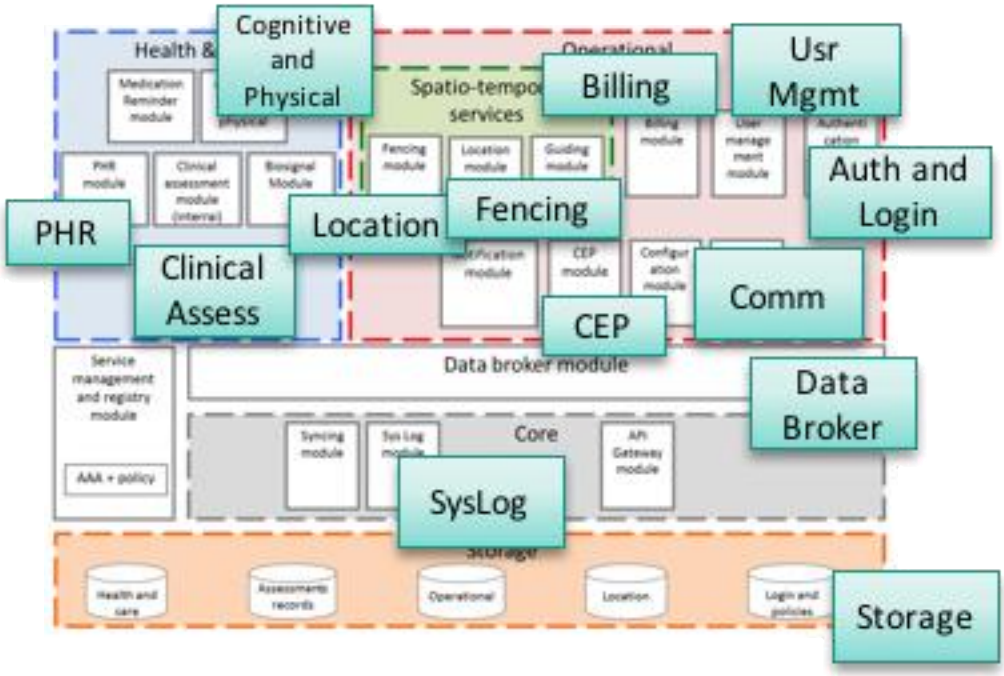


Figure 5 Deliverable D.2.3 - UNCAP Box device and service[3], (Figure 4: UNCAP set of modules.)

The DataBroker Module (based on Raptorbox, available at https://github.com/raptorbox) was the main module of the UNCAP services providing support for the communication between the UNCAP Box and all the modules that require collection data for processing, visualisation, or other purposes. The Clinical Assessment Module bridged UNCAP Cloud with the Atl@nte software a proprietary solution for clinical assessment. The Location Module manages users' positions making them available (authorized and authenticated) for use through the UNCAP platform. The fencing module generated alerts every time a given patient or device entered or left a specified district/location/room. The UserManagement Module enabled the management of patients (add, remove, associate with a given doctor) and users within the UNCAP Platform. Complementary, the Authentication And Login Module handled the authentication and login of the users onto the UNCAP Platform and services. The video/audio communication was served by the Communication Module. The storage of sensitive data on the UNCAP Cloud was delegated to a third-party technology, CHINO (https://www.chino.io), which had been integrated with UNCAP. CHINO, provided a database compliant with EU regulations for storage of sensitive health data. CHINO (exposing its functionality through a restful API) had been integrated through the UNCAP DataBroker. The Cognitive And Physical Module integrated the serious games applications. The CEP (Complex Event Processing) Module enabled the filtering of information from all the data collected via the Data Broker. Finally, the Phr Module allowed for communication with external healthcare providers and services.

[3] http://www.uncap.eu/downloads

3.3.3 Data models

One of the most relevant legacy of UNCAP, that will be beneficial to CAPTAIN, is the knowledge on data model based on Open mHealth[4]. The standard specification were adopted across the complete lifecycle of biosignals measurements, from the data acquisition and transfer, to data storage and analysis[5]. Open mHealth provides a framework for data standardization to facilitate companies, organizations, and individuals write reusable code and exchange data, making at the same time the data easier to understand, regardless of where the data comes from. For this reason, Open mHealth was among the initial data models that were suggested for the CAPTAIN system (see Deliverable D2.2 – First Version of system specification, §5.2 Main Core Components, ComponentM01 Communication Bus). However, the official data models for CAPTAIN follows the FIWARE approach.

For compliance with the main communication bus UNCAP, Raptorbox, a custom data model definition was necessary which is found at https://docs.raptorbox.eu/pages/documentation/api-docs/v5/http.html. Raptor useed a well-defined data model to define a "device" data model and all of its possible interactions, including Streams, Channels, and Actuations. Even a simple application should have follow the "device" model in order to comply with the Raptorbox.

3.3.4 Data broker

The UNCAP IoT DataBroker, which was based on Raptorbox, served as the data/messages exchanging mechanism between data sources and the UNCAP various services and applications, applying security checks on the data flow. Raptorbox was composed of a set of Restful HTTP APIs and an MQTT broker (publish / subscribe) to create a reactive stream of data and action triggers for devices and applications.

To address security and privacy in UNCAP, DataBroker (Raptor) worked in tight integration with the User Management module since both, queried for data updates and for subscription to channels which was authenticated by the User Management module for security purposes. Already registered users could start publishing data pushing it to an HTTP API of the DataBroker or requesting access to data streams produced by the DataBroker.

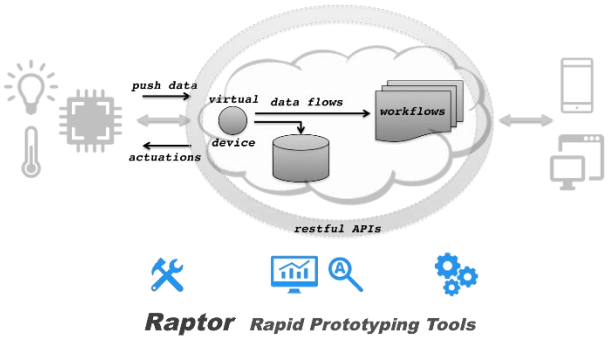


Figure 6 Raptorbox architecture[6]

[4] http://www.openmhealth.org/documentation/#/schema-docs/schema-library

[5] UNCAP Deliverable D.1.3 – System Architecture - v2.0, Section §9.9 Storage, available at (http://www.uncap.eu/downloads)

[6] https://docs.raptorbox.eu/pages/overview/architecture.html

Each device or application was represented in Raptor as a "virtual device". The new data, produced by a device, was pushed into the services through the Raptor restful APIs. There, the data was automatically associated to such a virtual device, stored into a scalable data store and made available for future usage and retrieval. Any business logic associated with the specific device in the cloud was therefore automatically triggered and executed (https://docs.raptorbox.eu/pages/overview/architecture.html).

4 FIWARE ORION DATA BROKER

4.1 INSTALLATION AND CONFIGURATION

Orion data broker officially recommends an installation procedure based on CentOS 7.x using RPM packages. According to Orion documents available from the official website[7], the typical requirements are 2 CPU cores and 4GB RAM, CentOS/RedHat as the operating system, MongoDB as the NoSQL database and finally the contextBroker package (mandatory) which has dependency on the following packages: libstdc++, boost-thread, boost-filesystem, gnutls, libgcrypt, libcurl, openssl, logrotate and libuuid. The installation of the package carried on done with yum (*yum install contextBroker*) or by downloading the package directly from the FIWARE Yum repository using command (as root) *rpm -i contextBroker-X.Y.Z-1.x86_64.rpm*.

After successful installation of Orion, a context broker service (contextBroker) should be installed on the system, inside /etc/init.d/. To check contextBroker service, it is necessary to issue the following commands from command line:

- /etc/init.d/contextBroker status : To check the status of the service
- /etc/init.d/contextBroker start : To start the contextBroker service
- /etc/init.d/contextBroker stop : To stop the contextBroker service

4.2 COMPILING AND INSTALLING ORION CONTEXT BROKER FOR UBUNTU LINUX 16.04

Captain Cloud server will run on an Ubuntu Linux 16.04. Since Orion does not have installation procedure for this operating system, the Orion context broker source files were compiled for Ubuntu Linux. The following sequence of commands was used for compiling and installing source files of Orion context broker on a Ubuntu Linux machine, following from the official guide[8]. The commands install the needed building tools, the required libraries, the MongoDB driver, the rapidjson, the libmicrohttpd, the Google Test/Mock and building and installing the fiware-orion.

```
sudo apt-get install make cmake gcc scons

sudo apt-get install libboost-all-dev libcurl4-gnutls-dev \ libgnutls-dev libgcrypt20-dev libssl-dev uuid-dev libsasl2-dev \
libcurl4-openssl-dev

wget https://github.com/mongodb/mongo-cxx-driver/archive/ \
legacy- 1.1.2.tar.gz
tar xfvz legacy-1.1.2.tar.gz
cd mongo-cxx-driver-legacy-1.1.2
```

[7] https://fiware-orion.readthedocs.io/en/latest/admin/install/index.html

[8] https://fiware-orion.readthedocs.io/en/latest/admin/build_source/index.html

```
sudo scons install --prefix=/usr/local --disable-warnings-as-errors

wget https://github.com/miloyip/rapidjson/archive/v1.0.2.tar.gz
tar xfvz v1.0.2.tar.gz
sudo mv rapidjson-1.0.2/include/rapidjson/ /usr/local/include

wget http://ftp.gnu.org/gnu/libmicrohttpd/ \
libmicrohttpd-0.9.48.tar.gz
tar xvf libmicrohttpd-0.9.48.tar.gz
cd libmicrohttpd-0.9.48
./configure --disable-messages --disable-postprocessor \
--disable-dauth
make
sudo make install
sudo ldconfig

wget https://nexus.lab.fiware.org/repository/raw/public/storage/ \
gmock-1.5.0.tar.bz2
tar xfvj gmock-1.5.0.tar.bz2
cd gmock-1.5.0
./configure
make
sudo make install
sudo ldconfig

sudo apt-get install git
git clone https://github.com/telefonicaid/fiware-orion

cd fiware-orion
sudo make di

sudo make && make install INSTALL_DIR=/usr
```

To check if everything was installed properly, the broker version message can be invoked by calling: *contextBroker –version*, while the expected outcome should be:

```
2.0.0-next (git version: 0540b6346cb69ea0faf6a390441f9c596501a01c)
Copyright 2013-2018 Telefonica Investigacion y Desarrollo, S.A.U
Orion Context Broker is free software: you can redistribute it and/or
modify it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.

Orion Context Broker is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero
General Public License for more details.

Telefonica I+D
```

Finally, if everything went well, the GET request to the broker, http://localhost:1026/version/, should give a result similar to the following:

```
{
   "orion": {
      "version": "2.0.0-next",
```

```
    "uptime": "2 d, 15 h, 43 m, 22 s",
    "git_hash": "0540b6346cb69ea0faf6a390441f9c596501a01c",
    "compile_time": "Tue Dec 11 16:59:10 EET 2018",
    "compiled_by": "root",
    "compiled_in": "captain-desktop",
    "release_date": "Tue Dec 11 16:59:10 EET 2018",
    "doc": "https://fiware-orion.rtfd.io/"
  }
}
```


The installation has been on the CAPTAIN testing server, located at AUTH's premises, under the domain name **api.captain-eu.org**. This domain name will be assigned to the CAPTAIN cloud server, a couple of months after the delivery of this deliverable. Thus, the version GET request can be tested at http://api.captain-eu.org:1026/version.

4.3 CAPTAIN-ORION BRIDGE

Orion accepts subscriptions[9] (method for passing Orion entities' activity to the CAPTAIN Cloud by notifying CAPTAIN servers for any change) to existing models. Thus, each new FIWARE entity must be created through the CAPTAIN Cloud API, which then creates the corresponding entity on the Orion (communication between CAPTAIN Cloud API and the Orion Broker). To do so, the FIWARE entity will perform a POST call to the following CAPTAIN Cloud API function the first time:

http://{{mentorage_endpoint}}/FiwareCreateEntity

With the following data structure as the body of the POST call:

```
{
  "entity":
  {
    "id":"1",
    "type":"sensor",
    "attributes":
    {
      "temperature":
      {
        "value": "23",
        "type": "Float"
      },
      "humidity":
      {
        "value": "720",
         "type": "Integer"
      }
    }
  }
}
```

[9] https://fiware-orion.readthedocs.io/en/1.8.0/user/walkthrough_apiv2/#subscriptions

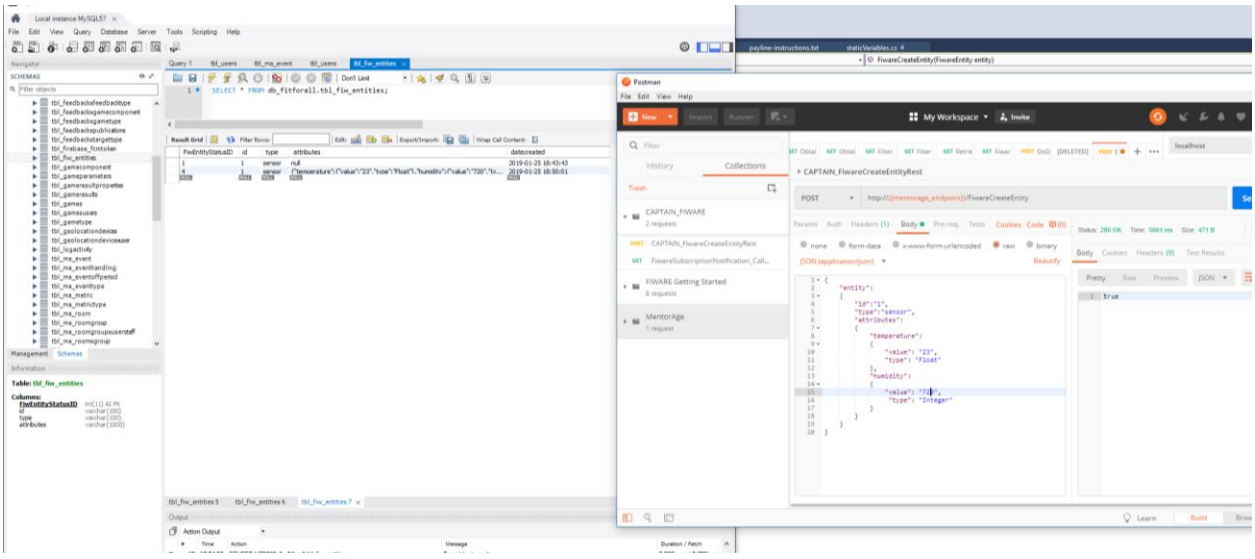


Figure 7 POST calls (through Postman) to the CAPTAIN Cloud API FiwareCreateEntity function

At the same time, the FiwareCreateEntity of the server creates a subscription for the specific device in order to get updated on each entity's attributes as soon as they are published on the Orion broker. This is by the following POST /v2/subscriptions operation is used:

```
{
  "description": "A subscription to get info about Room1",
  "subject": {
    "entities": [
      {
        "id": "Room1",
        "type": "Room"
      }
    ],
    "condition": {
      "attrs": [
        "pressure"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://localhost:1028/accumulate"
    },
    "attrs": [
      "temperature"
    ]
  },
  "expires": "2040-01-01T14:00:00.00Z",
  "throttling": 5
}
```

Each time there is an update on any entity on the Orion broker, a notification is sent to the CAPTAIN Cloud API which then saves this data to the CAPTAIN Database to further serve CAPTAIN requests (CAPTAIN Box, Satellite, users, etc). The callback function listens to:

http://{{mentorage_endpoint}}/FiwareSubscriptionNotification_Callback

The bidirectional communication which demands CAPTAIN information to be sent over the Orion Broker, is currently under consideration. Once the CAPTAIN Cloud API is ready (after M24) and by consulting the Data Management Plan and the GDPR issue, the technical partners will decide what can be published and it is meaningful (doesn't make sense to publish status of a CAPTAIN Box to a smart city environment). When and if this is decided, the CAPTAIN Cloud API will be invoking the corresponding Orion Broker Post call to publish the data properly, updating the attributes of the CAPTAIN entities:

```
curl localhost:1026/v2/entities/{id}/attrs
{
  "temperature": {
    "value": 26.5,
    "type": "Float"
  },
  "pressure": {
    "value": 763,
    "type": "Float"
  }
}
```

4.4 SECURITY

According to the official documentation of FIWARE[10], Orion doesn't provide "native" authentication nor any authorization mechanisms to enforce access control. However, authentication/authorization can be achieved by adopting the access control framework provided by FIWARE GEs, which will be further investigate by DIG partner once the security of the CAPTAIN system is developed. Based on FIWARE, Orion is integrated in this framework using the FIWARE PEP Proxy GE. At the present moment, there are two GE implementations (GEis) that can work with Orion Context Broker: Wilma[11] and Steelskin[12].

5 OPENHAB DATA BROKER

5.1 INSTALLATION AND CONFIGURATION

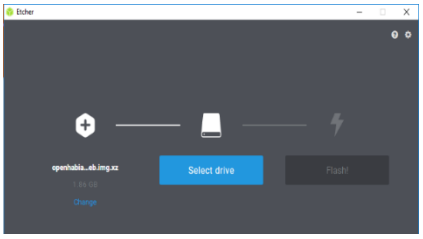


Figure 8 Flashing openhabian image using Etcher

OpenHAB provides the OpenHABian OS for Raspberry Pi installation which is an SD card image with pre-configured Linux system setup and OpenHAB software along with the OpenHABian Configuration Tool. OpenHABian provides also installation of the Zulu Embedded OpenJDK Java 8, the Samba file sharing and ogin information screen, powered by FireMotD. OpenHABian image can be downloaded and easily flashed into the SD card using Etcher (https://www.balena.io/etcher/). Etcher provides a very easy graphical interface allowing you to choose the image that you want

[10] https://fiware-orion.readthedocs.io/en/master/user/security/index.html

[11] https://catalogue-server.fiware.org/enablers/pep-proxy-wilma

[12] https://github.com/telefonicaid/fiware-pep-steelskin

to flash and the drive (Figure 9). When the writing has finished, the SD card is extracted from the PC. First of all, Raspberry PI (RPI) needs to be connected to the internet using a network cable and afterwards the SD card is placed into the slot. RPI is connected to the power supply and the installation starts automatically and lasts about 15-45 minutes. OpenHAB provides a Web UI which can be accessed from http://openhabianpi:8080. Also, SSH access is active by default in OpenHABian and can be used to remotely control Raspberry Pi.

Typing "*sudo openhabian-config*" and using the default password (openhabian) can open the configuration tool which provides optional settings and components such as easy installation of optional components, setup a backup of the system, Move the system partition to an external USB stick or drive and many more.

In order to further use the OpenHAB software we need to set up also the MQTT binding as the CAPTAIN system's architecture is based on the MQTT protocol. To do that we have to change the addons.cfg file by using the command "*sudo nano /etc/openhab2/services/addons.cfg*" and uncommenting the bindings and add the word mqtt "*binding = mqtt*".

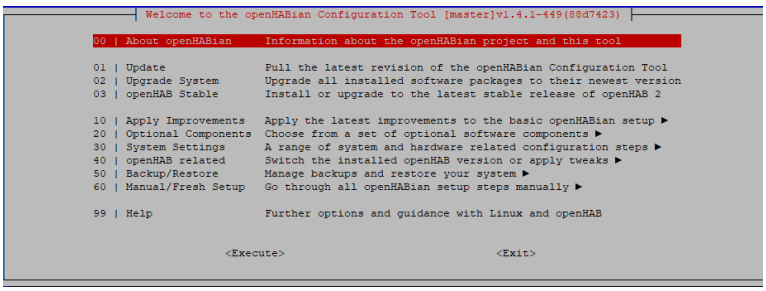


Figure 9 OpenHAB configuration tool

After we need also to configure the mqtt.cfg file, using the command "*sudo nano /etc/openhab2/services/mqtt.cfg*" and adding the following lines:

```
mqbroker.url=tcp://155.XXX.XX.XXX:XX
mqbroker.user= XXXX
mqbroker.pwd= XXXXX
```

where the url is the ip of the CAPTAIN's broker and user, pwd are the username an password. To check if the installation and the broker configuration were successful, we created a simple rule that posts a "Hello world" message every second and that was received by the CAPTAIN configure MQTT broker.

5.2 CAPTAIN-OPENHAB BRIDGE

In order to provide a home automation extension using CAPTAIN, we will develop a set of rules and processes. When a new device is added in the OpenHAB system, it will inform CAPTAIN system publishing a MQTT message under the topic openhab/devices when the status of the Thing is ONLINE announcing the device ID. The message will also contain the available Items for each device/Thing and the Item type. The available item types with the commands that handle their functionality can be found in https://docs.openhab.org/v2.1/concepts/items.html.

In order to handle each device, CAPTAIN will adopt the Homie 3+ convention. Homie adopts a topology for the topic declaration for MQTT home automation applications. The Homie topology includes:

- Devices: the physical piece of hardware e.g. a coffee machine
- Nodes: logically separable parts of a device, the Items that are defined in the openHAB framework
- Properties: represent basic characteristics of the node/device, the Items types defined in openHAB.

The base topic for all devices will be homie/ in order to be eligible for automatic discovery by third party controllers. The basic topic of a device will be homie/device ID which is a unique ID that will be announced to all the CAPTAIN components under openhab/devices topic. The topic homie / device ID / node ID will be the basic topic for each node where the node ID is the unique Item name. homie / device ID / node ID / property ID is the base topic of a property where property ID is the Item type of the given Item. A property value is directly published to the property topic based on the command type format of each item type[13].

On the openHAB side, each Item's inbound and outbound messages can be configured accordingly using the following format

```
Item myItem {mqtt="<direction>[<broker>:<topic>:<type>:<transformer>], ..."}
```

Where:

- Item is the Item type,
- myItem is replaced with the node ID,
- direction property is set to "<" for inbound messages and ">" for outbound
- broked, is the broker alias as it is defined in the openHAB configuration
- topic is the MQTT topic to subscribe to as defined by the homie convention
- type can be either 'state' or 'command', describing what the message contains

5.3 SECURITY

According to the OpenHAB official documentation, OpenHAB does not (yet) support restricting access through HTTP(S) for certain users - there is no authentication in place, nor is there a limitation of functionality or information that different users can access. They recommend that the OpenHAB instance MUST NOT be directly exposed to the Internet and the environment variable OPENHAB_HTTP_ADDRESS to be set accordingly. In CAPTAIN, the OpenHAB instance is connected to the internal network (in-home) as the communication of the CAPTAIN in-home devices with the CAPTAIN Cloud and the internet takes place through the CAPTAIN Box.

6 UNCAP DATA BROKER

6.1 INSTALLATION AND CONFIGURATION

The UNCAP Data Broker (Raptor) cloud service was hosted at https://raptor.srv.uncap.eu until the end of the UNCAP project. The UNCAP web Dashboard is available at the following URL: https://care.srv.uncap.eu. The UNCAP Single Sign On (SSO) service (User management and authentication) is located at https://authentication.srv.uncap.eu/UncapAuth/v2. An existing token can be verified as follows: https://authentication.srv.uncap.eu/UncapAuth/v2/verifytoken/[token]/[doctoruserid]/[patientid]. It is clear that no extra configuration is needed (or can be done) on the Cloud part of UNCAP.

The UNCAP BOX application is available online from GooglePlay store at http://www.uncap.eu/Ver2.1/Uncap-Box-v8-2.1.apk. Drivers for four devices are also available through

[13] https://docs.openhab.org/v2.1/concepts/items.html

Google Play (four devices already supported are a Glucometer, a Scale, a Blood Pressure monitor and an Oximeter.

Testing credentials for the UNCAP platform (available in the public deliverable D.2.3-–-UNCAP-Box-App-and-CLOUD-–-final-version) are:

Doctors (username: drlinda, password: 123)

Patients (username: david, password:123)

6.2 PUSH DATA FROM SENSOR TO THE UNCAP DATABROKER (HTTP API)

Authorised clients for pushing data through the data broker are those who have already registered with the User Management module (therefore able to provide a valid SSO token to the DataBroker). Any information that is sent to the IoT DataBroker, is first authenticated by the user management module (SSO token) and then it is published to the corresponding MQTT pub/sub channel/topic (Figure 10).

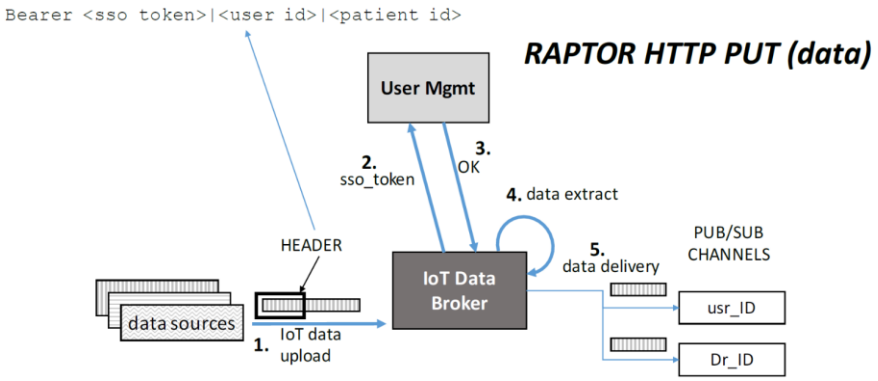
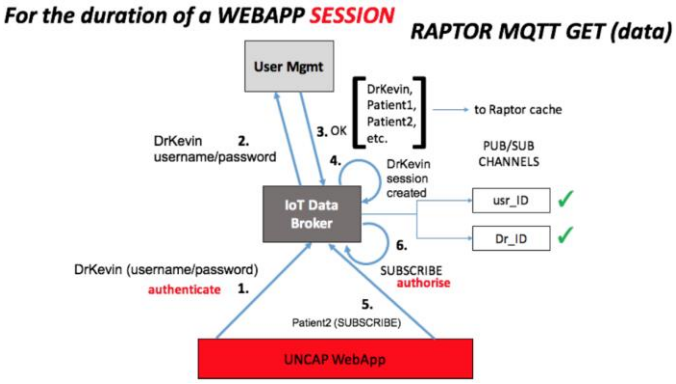


Figure 10 Deliverable D.2.3 - UNCAP Box App and CLOUD[14], (Figure 24: updating data on the DataBroker)

6.3 GET (LIVE) DATASTREAMS FROM THE DATABROKER (VIA MQTT)

Any authorized client can subscribe to a live data stream, uploaded on the DataBroker. Access to this streams is controlled by the DataBroker as in the following picture.



[14] http://www.uncap.eu/downloads

Figure 11 Deliverable D.2.3 - UNCAP Box App and CLOUD[15], (Figure 25: getting data from the DataBroker with Access Control)

6.4 CAPTAIN-UNCAP BRIDGE

On the CAPTAIN Cloud API side, a couple of functions were developed to make it ready for the integration of the basic functionality of UNCAP to the CAPTAIN. The main function that bridges CAPTAIN with UNCAP is the bridging of the two user management systems, so that both systems function independently but the collected data concerns the same user in both systems. This function is the following:

LogInTokenByUNCAPv2(string uncaptoken, string useruncap_id, string patientuncap_id);

taking as arguments the uncaptoken (created by the UNCAP user management module), the useruncap_id (the id of the user requesting the information) and the patientuncap_id (the id of the user for whom information is requested). The CAPTAIN Cloud API communicates with the UNCAP token verification mechanism to make sure that the token is valid. After a successful verification, CAPTAIN checks if the useruncap_id and the patientuncap_id have already been correlated with any existing CAPTAIN user (usage of external UNCAP key as attribute of the CAPTAIN user component). In case the users do not exist, they are automatically created in the CAPTAIN Cloud.

6.5 SECURITY

UNCAP is built around a good and robust user (and also users relation) authentication and authorization mechanism. Having everything under SSL encryption (https), the CAPTAIN Cloud API complies with the authentication and authorization mechanisms of UNCAP.

7 CONCLUSIONS

This deliverable has presented the data brokers that CAPTAIN bridges with, towards a more interoperable system, adopting standard communication protocols and frameworks. The different data brokers were adopted as complementary approaches and not as alternatives, meaning that one of the frameworks (FIWARE) extends CAPTAIN's communication on the cloud while the other one (openHAB) does the same on the local lever (CAPTAIN Box). Consequently, the CAPTAIN Cloud is enriched with information pertaining to the outdoor environment of the older adult while the local network, composed of the CAPTAIN Box and Satellites, is enriched with various home automation mechanisms.

However, CAPTAIN will focus on the main communication bus and protocol, namely MQTT and FIWARE data models that will serve the its main functionality. The work presented in this deliverable is expected to be further exploited by the latest stages of CAPTAIN where liaison with other projects and integration of third-party technologies will be sought.

[15] http://www.uncap.eu/downloads