BE in CPPS

Innovation Action Project

HORIZON 2020 - EU.2.1.5. - Ref. 680633

D2.4 - IoT Platform Federation

Lead Author: Giacomo Inches [FINCONS]

With contributions from:

Mauro Isaja [ENG], Antonio J. Jara [HOPU], Rafael Ruiz [HOPU] , Germán M. Carrasco [HOPU]

Deliverable nature	Other
Dissemination level	PU
Contractual delivery date	30 October 2017
Actual delivery date	22 December 2017
Version	1.1
Keywords	Internet of Things (IoT), Wrappers, Agents, Device Management Interoperability, Deployment, Device Management

Version History

Nr.	Date	Notes and Comments
0.1	18.09	First draft of document structure
0.2	06.10	FINCONS initial contributions
0.3	27.10	ENGINEERING contributions
0.4	10.11	HOPU contributions, document structure revision, first draft
0.5	20.11	HOPU and FINCONS contributions revision
0.6	23.11	Final draft ready for internal review
0.7	30.11	Internal review
0.8	04.12	HOPU revision
1.0	06.12	Final release
1.1	22.12	Final release with formatting enhancement

Deliverable Peer Review Summary

Id.	Addressed	Comments

List of Figures

Table of Contents

Executive Summary

Deliverable D2.4 updates and finalizes the work introduced in deliverable D2.3 with respect to the IoT platform federation.

The information reported in this deliverable takes into account the significant restructuring and rationalization of the BEInCPPS architecture reported in D2.2.

Therefore, as compared to the previous D2.3 version, this document takes into account the outcomes reported in D2.2 and describes the new, or revised, components as compared to the ones reported in D2.3.

For unchanged components the reader is still referred to D2.3, which are still part of the BEinCPPS platform even if their placement within the overall BEinCPPS must be read in the light of the D2.2 rationalised architecture.

In particular, what is framed in D2.3 within the context of the *Smart Lane* (see D2.3 §3.1.1.2) are now background components of the *Factory Level* and support the D2.2 *OpenIoT Middleware* and *CPPS Messaging Services*, while the D2.3 *Fast Lane* (see D2.3 §3.1.1.3) support the *FIWARE Orion Context Broker* and *CPPS Publishing Services*. The *Homard* background components (D2.3 §3.1.1.4) are now framed as *Value Added Services and Applications* in the revised BEinCPPS architectural framework (see Figure 26 in D2.2). The Homard components are being maintained within the scope of D2.4 being components supporting the integration between the *Factory* and *Field* levels.

In fact, all components, both the ones reported in this document as the ones in D2.3, belongs to the *Factory Layer*, (see Figure 11 in D2.2) or deals with the integration of this layer with the *Field* or *Cloud* ones.

The new or enhanced/revised *Factory Level* foreground components belonging to this D2.4 are:

- the CPPS Publishing Services, connecting shopfloor machines and field devices to FIWARE-based applications through the Platform's Information Bus,
- the CEP Input and Output Adapters offering the possibility to use the OpenIoT Middleware to connect the *Field* to the CEP in the *Cloud*,
- the SeDEM component which enhance a CPPS Messaging Services module (*OpenIoT Client Library*) provided in the first release of the IoT Platform supporting end-to-end data confidentiality,
- the Task Orchestration for CPPS which is an Activiti BPM Engine based component for the semantically-enhanced design, execution and monitoring of Business Processes,
- the Device Url Manager Component to manage devices' broadcasted information,
- the Homard IoT Management for CPPS to manage devices using the OMA LwM2M protocol.

For each component the following sections is provided within this document: a short description and a summary of the functionalities. Technical and operation details are illustrated in the specific annexes.

1. Introduction

1.1. Objective of the Deliverable

As for the previous D2.3 deliverable, the objective of D2.4 is to provide information about the new functional components added to the BEinCPPS *Factory Layer* or components enhanced as compared to their description in D2.3.

The information reported in this deliverable takes into account the significant restructuring and rationalization of the BEInCPPS architecture reported in D2.2, as well as lessons learned from on-the-field experiments in pilot sites – see deliverable D3.7.

Anyway, all components, both the ones reported in this document as the ones in D2.3, belongs to the *Factory Layer*, (see Figure 11 in D2.2) or deals with the integration of this layer with the *Field* or *Cloud* ones.

As for the first release (D2.3) this document includes background and foreground components.

For unchanged components the reader is still referred to D2.3, which are still part of the BEinCPPS platform even if their placement within the overall BEinCPPS must be read in the light of the D2.2 rationalised architecture.

1.2. Structure of the Deliverable

This document in §2 shortly frames the BEinCPPS IoT Platform within the revised BEinCPPS architecture as detailed in D2.2; following §3 describes the enhanced or new components framing them within the overall BEinCPPS architecture; while §4 reports some conclusions and future work. Finally the appendix section reports the specific factsheets of the enhanced or new components.

1.1. Applicable Documents

The components described in this document, as well as the ones in D2.3, are an instantiation of the "implementation perspective" described in D2.2 §4.4 and, specifically, belong to the BEinCPPS *Factory Level*.

Relevant documents for what is present in this deliverable are therefore:

D2.3 IoT Platform Federation: for components not revised after D2.3 delivery,

D2.2 BEinCPPS Architecture and Business Processes: which details the final BEinCPPS architecture which is the reference framework for this deliverable,

D3.7 Technical evaluation, lessons learned, recommendations: which provides the rationale behind the architectural revision and technical choices.

2. IoT Platform Federation

2.1. Introduction and positioning in the updated BEinCPPS architecture

The IoT Platform which titles this deliverable is tied to the original BEinCPPS view of including IoT related solutions in its architecture and exploit components of IoT platforms, like the OpenIoT one [4], side by side with FIWARE/FITMAN components to support the integration of the shop floor with the cloud services. The above view has to be revised in light of the BEinCPPS architectural view revision detailed in D2.2.

The components in D2.4, and the unchanged ones in D2.3, cover the BEinCPPS *Runtime* domain *Factory Level* (see Figure 1).

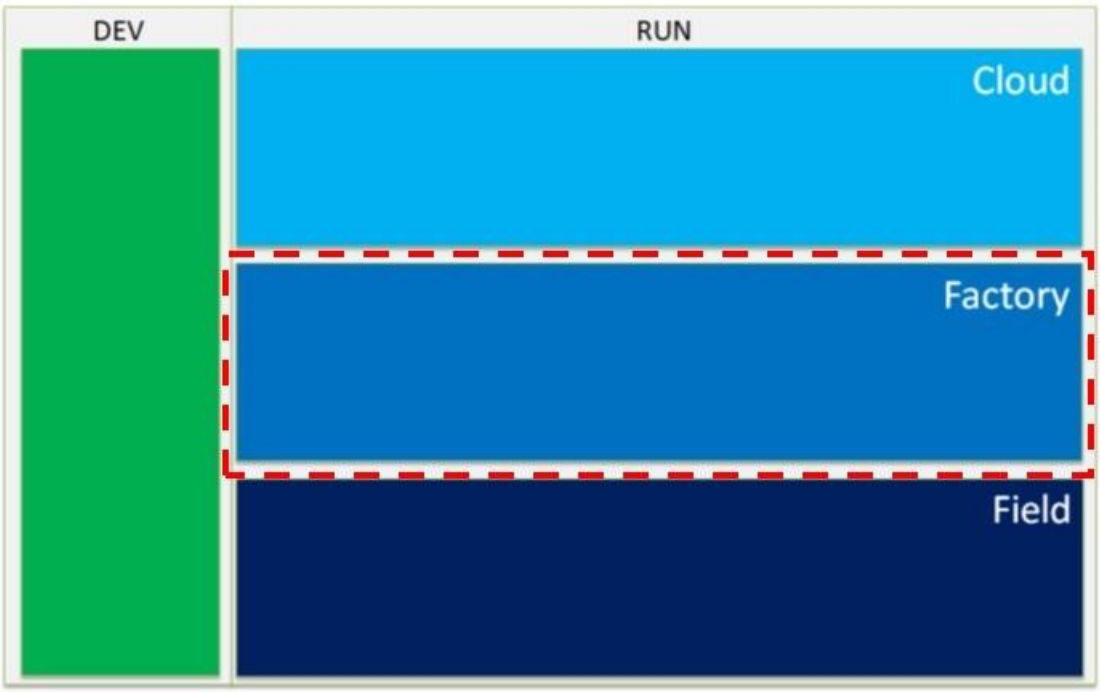

Figure 1 - BEinCPPS Architectural Structural Perspective

Therefore D2.4 still focuses BEinCPPS features to connect the *Field* to the *Cloud* and support production plant local processing providing a multi-layered, federated system for connecting and controlling Cyber-Physical Production Systems.

Figure 2, instead, frames D2.4 within the BEinCPPS *Implementation Perspective* and provides a summary of the functional components in the *Factory Level*.

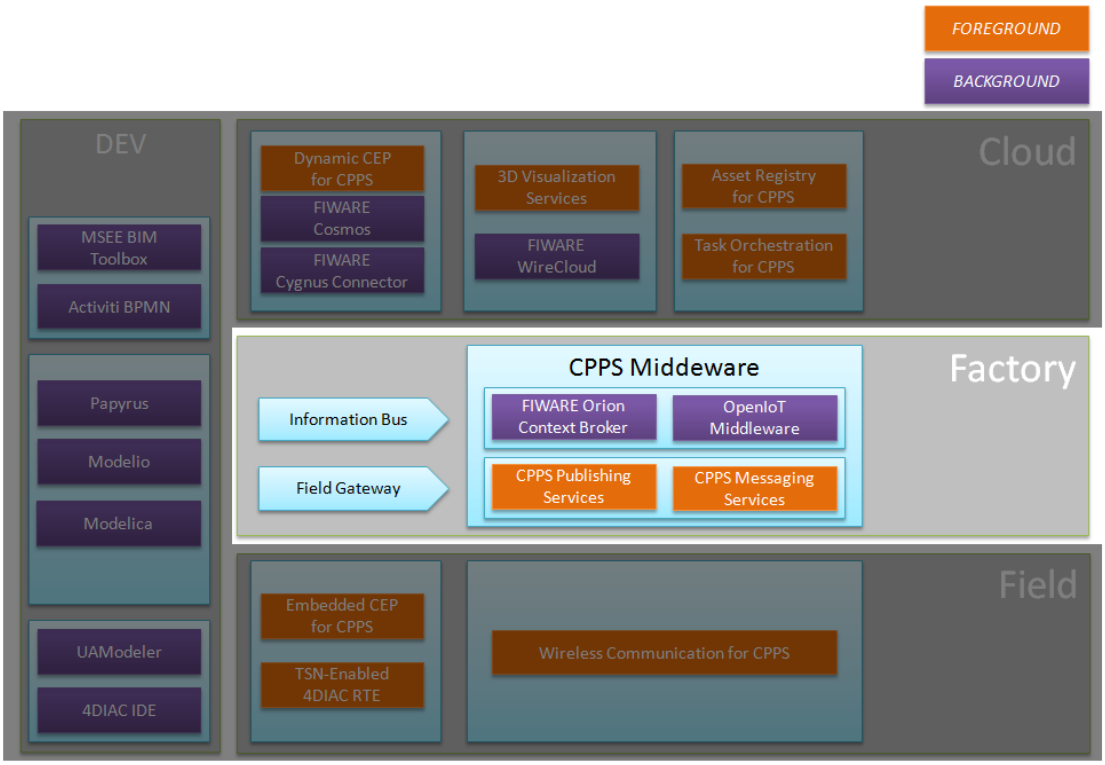

Figure 2 - Highlight of the IoT Platform within the BEinCPPS architecture

As clearly indicated in Figure 2 the BEinCPPS IoT Platform foreground components are the *CPPS Publishing Services* and *CPPS Messaging Services* which provide an abstraction layer over the shop floor exposing shop floor components data and services in a uniform and common way.

The specific components described in §3 are:

- the CPPS Publishing Services is a component of the Field Gateway provided by the IoT Platform, connecting shopfloor machines and field devices to FIWARE-based applications through the Platform's Information Bus,
- the CEP Input and Output Adapters which enhance the CPPS Messaging Services offering the possibility to use the OpenIoT Middleware to connect the *Field* to the CEP in the *Cloud*,
- the SeDEM component which enhance a CPPS Messaging Services module (*OpenIoT Client Library*) provided in the first release of the IoT Platform supporting end-to-end data confidentiality,
- the Task Orchestration for CPPS which is an Activiti BPM Engine based component for the semantically-enhanced design, execution and monitoring of Business Processes,
- the Device Url Manager Component to manage devices' broadcasted information,
- the Homard IoT Management for CPPS to manage devices using the OMA LwM2M protocol.

3. Components

As for the previous deliverable, the D2.4 components presentation is split according to the *nature* of the component and, specifically, at a first level according to the classification as background or foreground, and at a second level according to their license (i.e., open source or proprietary).

3.1. Background components

For the background components the ones reported in D2.3 are confirmed as components of the BEinCPPS *Factory Level*, except for the *Design Tools* (see §3.1.1.1 in D2.3) which have been moved to D3.6 as a consequence of the rationalization of the BEinCPPS architecture.

The remaining D2.3 background components are therefore still to be considered as building blocks on which most of the BEinCPPS foreground components are built or operate.

It must anyway be highlighted that, due to the rationalization of the BEinCPPS architecture, some of the *wording* in D2.3 are no more valid. Therefore, what is framed in D2.3 within the context of the *Smart Lane* (see D2.3 §3.1.1.2) are now background components of the *Factory Level* and support the D2.2 *OpenIoT Middleware* and *CPPS Messaging Services*, while the D2.3 *Fast Lane* (see D2.3 §3.1.1.3) support the *FIWARE Orion Context Broker* and *CPPS Publishing Services*.

The *Homard* background components (D2.3 §3.1.1.4) are now framed as *Value Added Services and Applications* in the revised BEinCPPS architectural framework (see Figure 26 in D2.2). Even if not directly positioned within the D2.2 *Factory Level* the Homard components are being maintained within the scope of D2.4 being components supporting the integration between the *Factory* and *Field* levels.

Being the D2.3 background components, both the open source and the proprietary ones, still valid even if reframed as specified above they will not be described in the following. The reader is referred to the mentioned sections in D2.3 for details.

3.2. Foreground components

In this section the new or enhanced/revised *Factory Level* foreground components are described.

For each component the following sections provide:

- a short description
- a summary of the functionalities

while the following technical and operational details:

- BEinCPPS specific developments
- HW/SW Prerequisite
- Installation Instructions

- User Manual
- Developers' Guide
- Examples
- Licensing

in specific appendixes.

3.2.1. Open Source Components

3.2.1.1. CPPS Publishing Services

CPPS Publishing Services (CPPS-PS) is a component of the Field Gateway provided by the IoT Platform, connecting shopfloor machines and field devices to the Platform's Information Bus and, from there, to FIWARE-based applications. CPPS-PS is actually a collection of software modules, called *Agents*, each implementing the adaptation/translation between a specific field communication protocol (e.g., MQTT, LWM2M, OPC UA) and the NGSI standard, based on the publish/subscribe pattern, that is used in FIWARE for the exchange of *context information*.

While some of these Agents are provided off-the-shelf by the FIWARE core platform (and as such are included in the BEinCPPS IoT Platform as background technology), the BEinCPPS project filled an existing gap: an Agent supporting OPC UA devices, which allows system integrators to configure mappings between OPC UA *address spaces* and NGSI *contexts*, without the need to write any code and hiding the nitty-gritty of the underlying OPC UA binary communication protocol. In the following paragraphs, and in the technical factsheet that is provided as an annex to this document, we describe the OPC UA Agent in detail.

The main functionalities of the component are:

- **Connect physical devices to FIWARE-based systems**
  Supports devices that use the OPC UA standard API, and connects them to an instance of FIWARE Orion Context Broker (OCB).

- **Manage NGSI Context Entities**
  Automatically creates one Context Entity per physical connected device.

Each device will be mapped as a Context Entity following some rules. Basically, the user will provide an Entity Name and an Entity Type for a given device. The Entity Type will define some named Entity Attributes. Individual fields of an OPC-UA device's address space will be mapped to specific Entity Attributes of the matching Entity Type.

Fields can have two different behaviours:

- **Active attributes**
  Fields that are pushed from the device to the Agent. This field's changes will be sent to the Context Broker as `updateContext` requests over the device entity.

- **Lazy attributes**
  Some sensors will be passive, and will wait for the Agent to request for data. For those fields, the Agent will register itself in the Context Broker as a Context Provider (for all the lazy fields of that device), so if any component asks the Context Broker for the value of that sensor, its request will be redirected to the Agent. Updates over this field will be transformed into commands over the device by the Agent.

These are the features that the OPC-UA Agent exposes:

- **Device registration**
  Multiple devices will be connected to the Agent, each one of those mapped to a Context Broker entity. The OPC UA Agent will register itself as a Context Provider for each device when it provides commands or lazy attributes.
- **Device information update**
  Whenever a device has new fields to publish, it should send the information to the Agent with OPC UA protocol. This message should, in turn, be sent as an `updateContext` request to the Context Broker, where the fields will be updated in the device entity.
- **Device command execution and value updates**
  As a Context Provider, the Agent should receive update operations from the Context Broker, and relay them to the corresponding device (decoding it using its ID and Type, and other possible metadata). These commands will arrive as `updateContext` operations from the Context.

The most important functionality introduced in this second release of the software is *automatic mapping*. The Agent can be instructed to *watch* a particular Folder of the target Address Space and to automatically create one NGSI Context Entity for each and every Object found there. In the Agent configuration file it is possible to define the Folder name and the structures to look for. Every time a new Object appears on the OPC UA Server, the Agent immediately creates a new Context Entity on the OCB, establishing a live connection between the two.

3.2.1.2. CPPS Messaging Services

The enhancements in the *CPPS Messaging Services* are finalized to both complete the BEinCPPS architecture vertical connectivity between shopfloor devices or applications and some services (i.e., the CEP) in the cloud level, as well as to take into account ongoing standardization activities related to OPC UA.

Indeed, the OPC Foundation is currently revising the OPC UA specifications [1] to both revise some features, as well as to add new ones. In particular the coming OPC UA specifications will include a full **OPC UA PubSub communication schema** [2] supporting the well-known publish subscribe communication pattern.

The current OPC UA standard envisages a kind of publish-subscribe service (in the new draft specifications called *Client Server Subscription model*) that provides reliable delivery, acknowledgements, and retransmissions. Unfortunately, this kind of

subscription service is a point-to-point service and therefore requires resources in the OPC UA Server for each connected OPC UA Client.

The new **OPC UA PubSub** schema instead is designed to support a uniform model for distributing data and events from an OPC UA information source to interested observers. OPC UA applications do not directly exchange requests and responses, but published information is sent to a *Message Oriented Middleware* that takes care of distributing it to all interested subscribers.

The new schema, as currently drafted in [2], envisages two different *Message Oriented Middleware* variants to cover a large number of use cases:

- a ***broker-less form*** in which the *Message Oriented Middleware* is the network infrastructure and Subscribers and Publishers use datagram protocols like UDP multicast;
- a ***broker-based form*** in which the *Message Oriented Middleware* is a message Broker and Subscribers and Publishers use a standard messaging protocol like AMQP [3].

In the context depicted above the new *CPPS Messaging Services* components are:

- CEP for CPPS - Input Event Adapter
- CEP for CPPS - Output Event Adapter

While to take into account the OPC UA specifications revision process an enhancement of the *OpenIoT Client Java Library* (see D2.3 §3.2.1 and D2.3 - Appendix 3 – Factsheet) is being provided named *Secure Data Exchange Middleware* (SeDEM).

3.2.1.2.1. CEP Input Event Adapter Component

3.2.1.2.1.1. Short Description of the component

This section describes a component providing the capability of connecting the BEinCPPS Factory Level (see D2.2) to the BEinCPPS Cloud Level and, specifically, the CPPS Messaging Services to the Dynamic CEP for CPPS so that events managed by the Factory Level Messaging Services can feed CEP workflows, as depicted in Figure 3, and therefore be processed within the CEP as specified in the fed workflows.

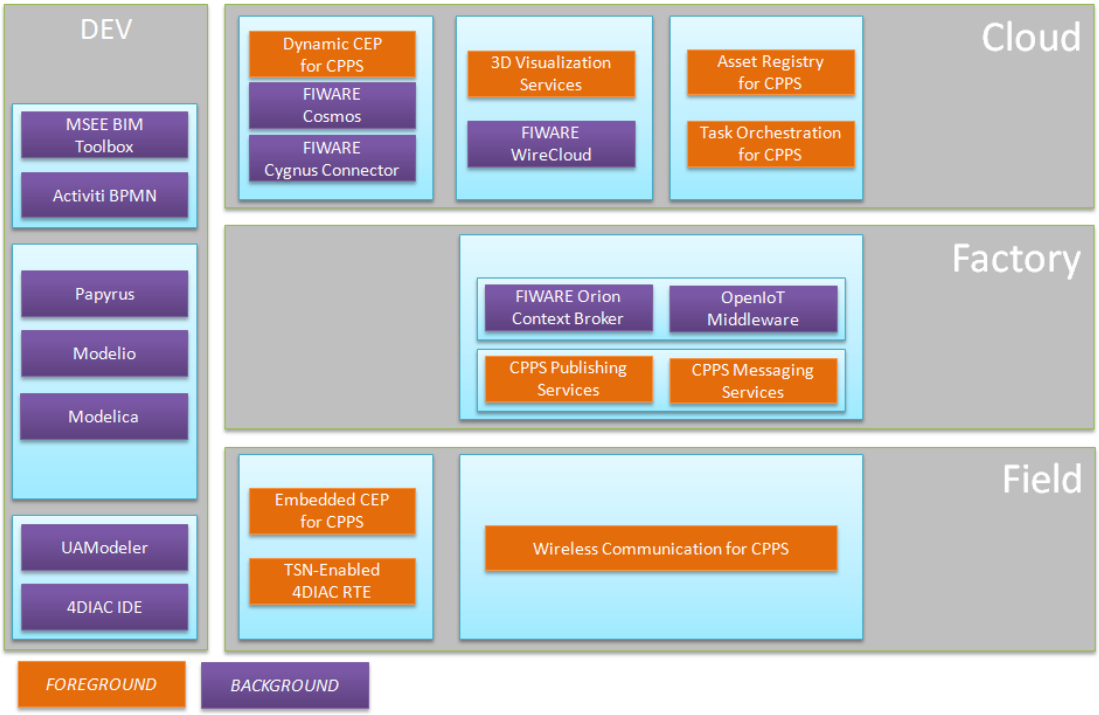

Figure 3 - BEinCPPS Implementation Perspective

This component actually supports the user in two different contexts:

- while designing the CEP workflows thanks to the provision of specific HTML and code to be imported within the interactive CEP management web application;
- at run time thanks to the provision of the Java code that will actually connect the Messaging Services to the CEP run-time.

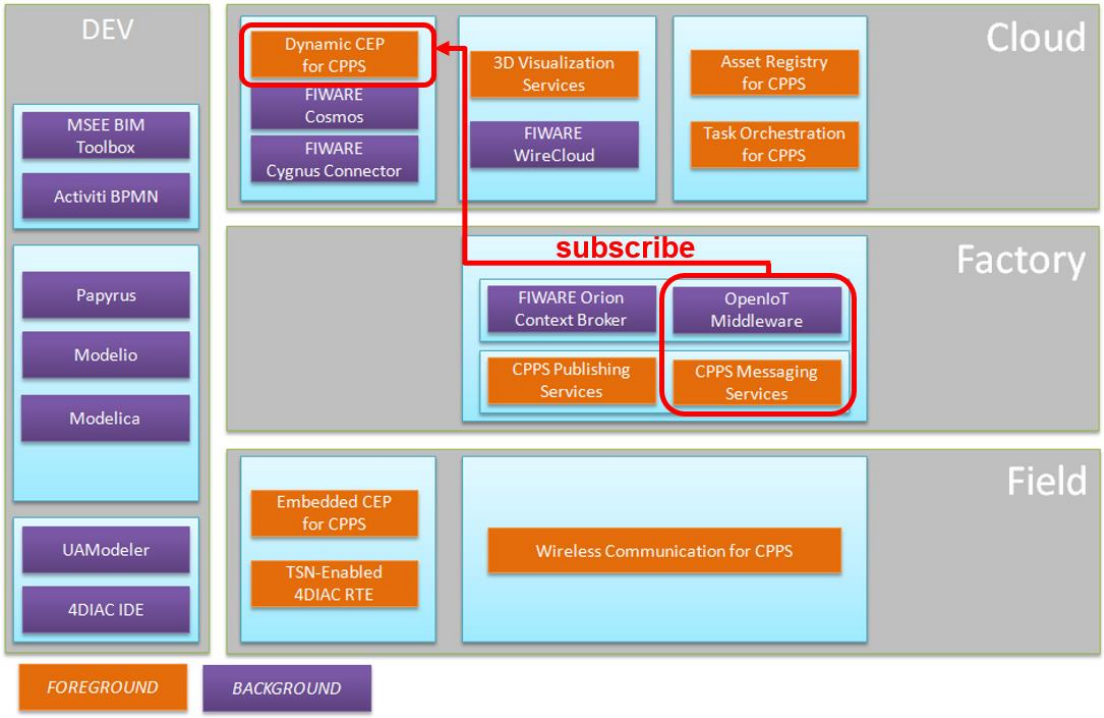

Figure 4 - BEinCPPS developed components in the architecture

Being the Dynamic CEP for CPPS based on the WSO2 CEP platform, for the design and testing of CEP workflows the component described in the following adopts the WSO2 CEP Management platform and, therefore, provides components to enrich this platform with a specific event input adapter able to manage the events' feeding from the BEinCPPS Messaging Services, which is based on RabbitMQ.

In order to provide to users hints consistent and uniform with the WSO2 CEP Management platform terminology and UI, in the following we will directly use the terms WSO2 CEP and RabbitMQ to refer to the components the adapter connects.

3.2.1.2.1.2. Summary of main functionalities

A CEP provides a specialized and optimized environment for data stream analysis, and specific features (e.g., a data analysis language, data processing workflows) for stream data, supporting the data stream processing and patterns detection.

CEP processing is usually organized in workflows, where a workflow deals with specific data stream sources and perform specific data processing and pattern search. As depicted Figure 5 a workflow has one or more Event Receivers that connect the workflow to the external data sources, directives on the processing to be performed and one or more Event Publishers that pushes specific events to external systems or services as patterns are identified in the input streams.

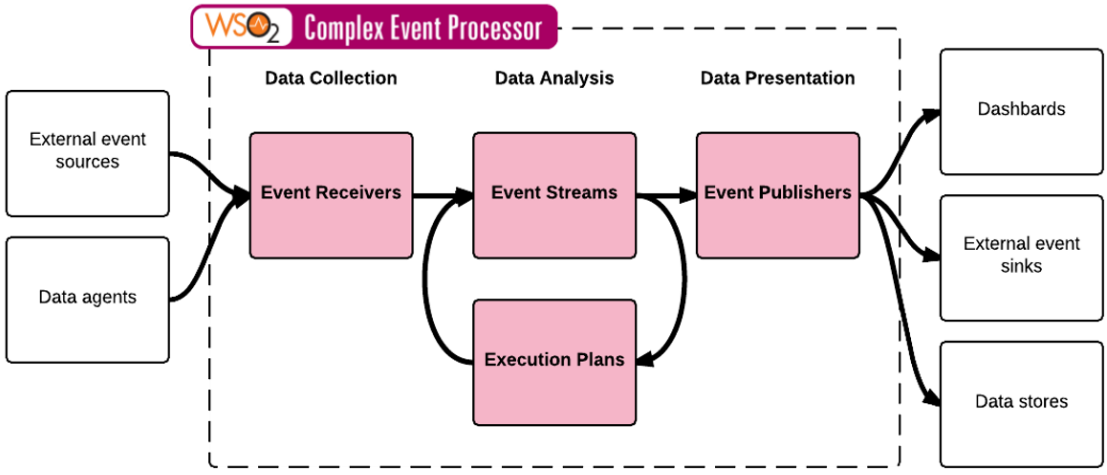

Figure 5 - Diagram of event processing in a CEP event flow.

The WSO2 CEP does not support acquiring events from a RabbitMQ event broker, that is why the component described in this document has been developed.

The developed component, therefore, adds an additional stream data source to the ones supported by the CEP therefore making possible to feed CEP workflows via RabbitMQ.

3.2.1.2.2. CEP Output Event Adapter

3.2.1.2.2.1. Short Description of the component

This factsheet describes a component providing the capability of connecting the BEinCPPS Cloud Level (see D2.2) to the BEinCPPS Factory Level (see in Figure 3) and, specifically, the Dynamic CEP for CPPS to the CPPS Messaging Services so that events generated by CEP workflows can fed the OpenIoT Middleware in the Factory Level, as depicted in Figure 3. In this way events coming out from CEP processing can be, horizontally or vertically, made available to other components, in the Cloud or Field BEinCPPS levels, without overloading the CEP workflows.

This component actually supports the user in two different contexts:

- while designing the CEP workflows thanks to the provision of specific HTML and code to be imported within the interactive CEP management web application;
- at run time thanks to the provision of the Java code that will actually connect the CEP run-time Messaging Services to the Messaging Services.

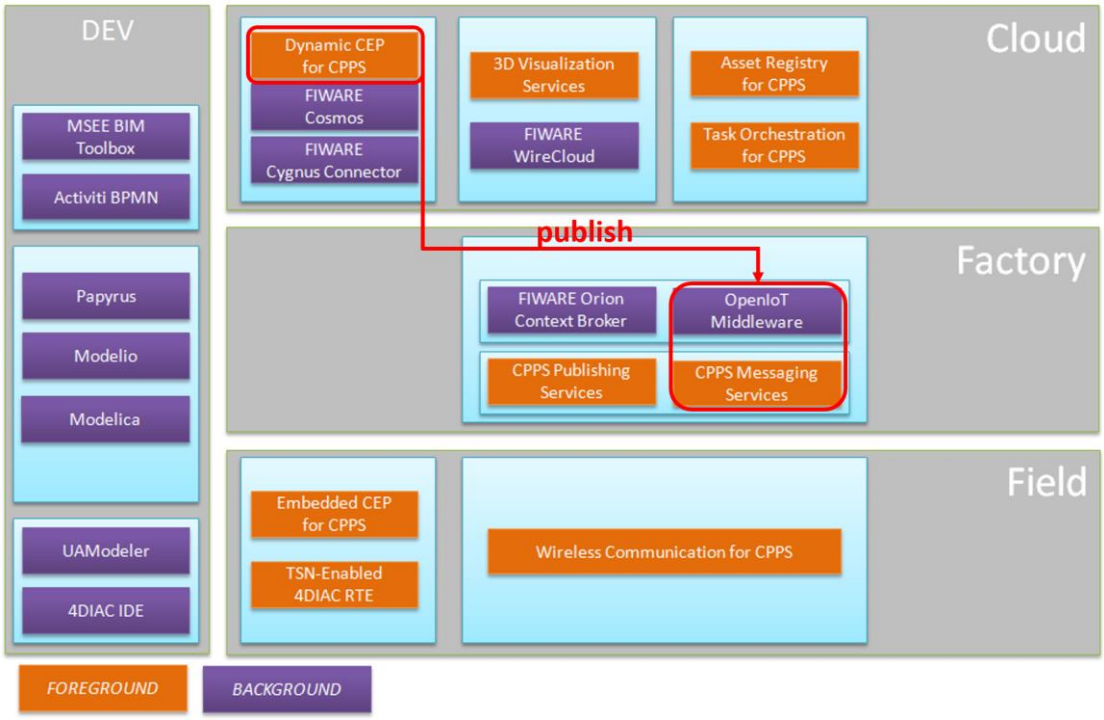

Figure 6 - BEinCPPS developed components in the architecture

Being the Dynamic CEP for CPPS based on the WSO2 CEP platform, for the design and testing of CEP workflows the component described in the following adopts the WSO2 CEP Management platform and, therefore, provides components to enrich this platform with a specific event output adapter able to manage the events generated by the Dynamic CEP to the BEinCPPS Messaging Services, which is based on RabbitMQ, therefore enriching the output adapters of the CEP platform.

In order to provide to users hints consistent and uniform with the WSO2 CEP Management platform terminology and UI, in the following we will directly use the terms WSO2 CEP and RabbitMQ to refer to the components the adapter connects.

3.2.1.2.2.2. Summary of main functionalities

As described in §3.2.1.2.1.2 a CEP provides a specialized and optimized environment for data stream analysis normally via the definition of workflows with one or more data sources, one or more data processing and pattern search elements and one or more Event Publisher (see Figure 5) that pushes specific events to external systems or services as patterns are identified.

The WSO2 CEP does not support publishing events to an event broker like RabbitMQ, that is why the component described in this document has been developed.

The developed component, therefore, adds an additional output data stream to the ones supported by the CEP. The output data stream is actually made up of the events the WSO2 CEP generates as outcome of pattern search and push out through the Output Event Adapter to an AMQP *Exchange* so that the events' consumers will receive them.

3.2.1.2.3. Secure Data Exchange Middleware

3.2.1.2.3.1. Short Description of the component

The Secure Data Exchange Middleware (SeDEM) is an enrichment of the OpenIoT Client Java Library described in the BEinCPPS D2.3 - Appendix 3 – Factsheet that provides features to support end-to-end data confidentiality using a new approach to ensure data sovereignty and flexible access control to data owners.

As for the previous Client Java Library the new one provides Java classes to ease publish and subscribe operations using the AMQP protocol [3]. Typically, the new library is aiming at supporting end-to-end data confidentiality between smart devices in the BEinCPPS Factory Level and services in the BEinCPPS Cloud Level via the CPPS Messaging Services.

The new library supports events' publishers, providing specific methods for events creation, encryption and publishing, and events' subscribers, providing methods for new event's notification and decryption (see Figure 7).

In the context of the new OPC UA draft specifications the SeDEM library supports the implementation of the *OPC UA Broker DataSet Writer* and OPC UA *Broker DataSet Reader* (see §7.3.4 and §7.3.5, respectively, in [2]) when using RabbitMQ as AMQP Message Broker.

The current OPC UA draft specifications envisage the possibility to encrypt the exchanged messages using shared keys managed via a *Security Key Server*. This security approach still presents issues tied to the need to share secret keys and to scalability. That is why the SeDEM library provides a more secure and scalable message encryption approach as is described in the following.

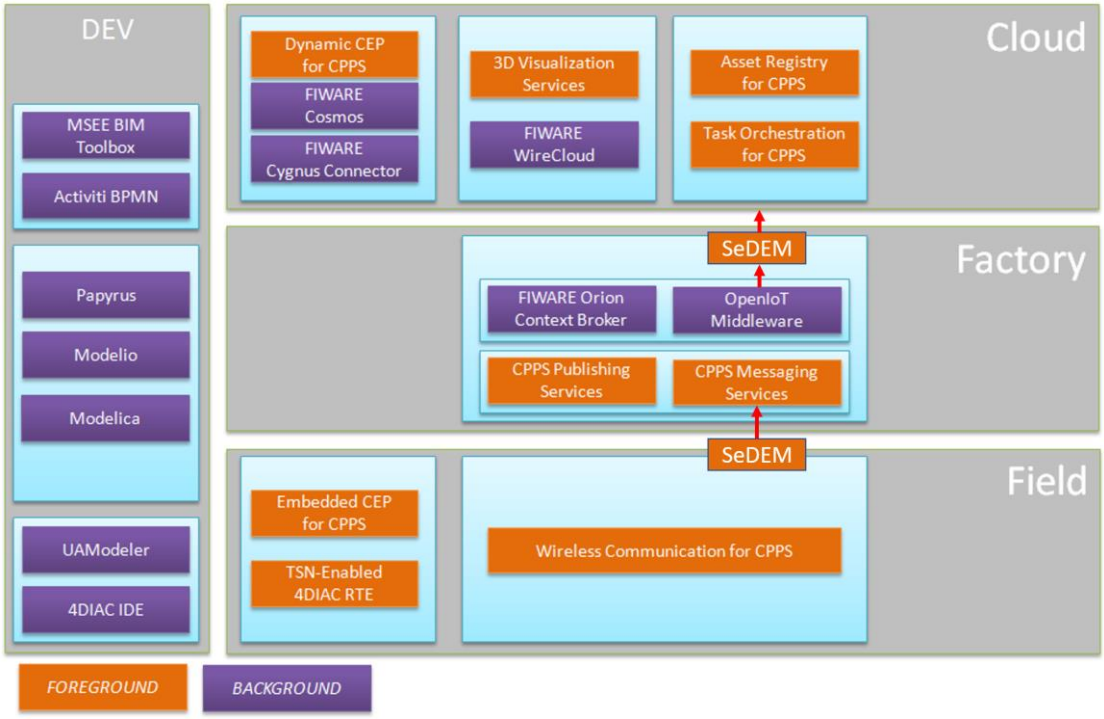

Figure 7 - BEinCPPS developed components in the architecture

The data confidentiality is assured using a combination of the CP-ABE asymmetric encryption technique and the symmetric AES one, therefore combining the efficiency of symmetric cryptography with the flexibility and fine-granularity provided by Attribute-Based cryptography.

CP-ABE (Ciphertext-Policy Attribute-Based Encryption) is a new asymmetric cryptographic technology for guaranteeing data confidentiality but also fine-grained access control. The encryption process is based on an access policy (see Figure 8) that specifies the conditions a private key must match to successfully decrypt the data; the same data encrypted with a different access policy will produce a different encrypted data. Each subject (e.g., person, device, etc.) potentially interested to decrypt data has a personal private key that is generated according to a set of attributes that characterize the subject (e.g., a user's profile); if the subject's attributes are compatible with the access policy with which a given data has been encrypted the private key will succeed in decrypting the data. CP-ABE, therefore, does not require to share keys, to know in advance the subjects that need to have access to the confidential data and provides flexible and fine-grade access policy.

```
(
    (
        Department = Marketing
        AND
        (
            Location = Rome
            OR
            Location = Paris
        )
    )
    OR
    (Department = Production)
    OR
    (Role = Administrator)
)
```

Figure 8 - Example of a CP-ABE access policy

The processing resources of the CP-ABE encryption technique depend on the complexity of the access policy (i.e., number of attributes and of logical operators, as well as the policy complexity), therefore a pure CP-ABE approach is not suitable in contexts with resource-constrained devices or with high data rates (see Figure 9).

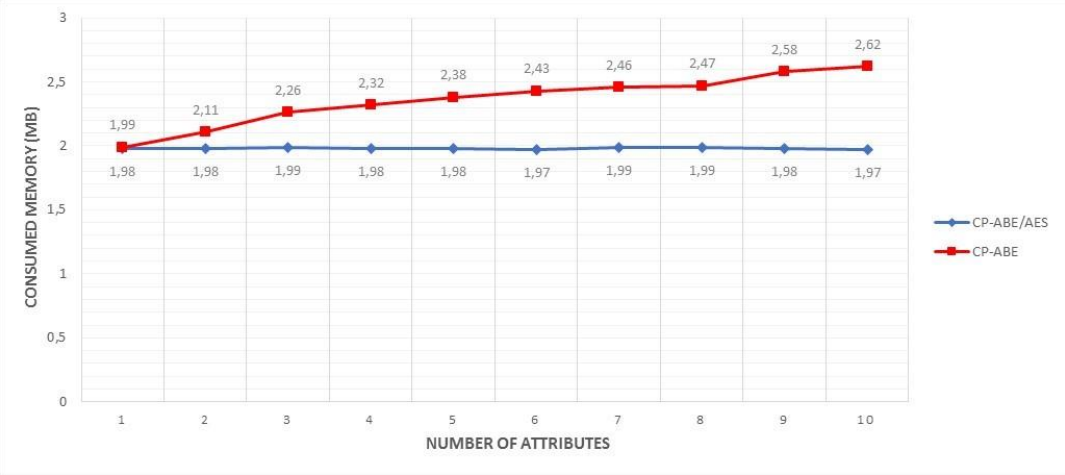

Figure 9 - RAM usage vs Policy Complexity[1]

[1] actual measures on a Raspberry Pi 3 board

To overcome this problem SeDEM combines the flexibility of CP-ABE with the efficiency of the AES symmetric encryption algorithm. A SeDEM data source cooperates with a SeDEM service (the ABE Proxy Service, see **Figure 22**) to securely (using ECDH) generate ephemeral AES keys, these keys are encrypted using CP-ABE while the data is encrypted via AES with the ephemeral keys. The ephemeral keys can be renewed as necessary (e.g., every *t* sec or every *n* events) so to increase the security of the encrypted data, while the CP-ABE encrypted ephemeral keys can be made available to data consumers using any suitable mean (e.g., publishing them via a message broker, sharing them via a database, etc.) being them secured thanks to the CP-ABE encryption. A data consumer has to first acquire the AES ephemeral key used to encrypt the data at hand, decrypt the AES key suing its CP-ABE key and then decrypt the data using AES. If the data consumer CP-ABE key meets the access policy the data consumer will acquire the right AES and therefore succeed in decrypting the data.

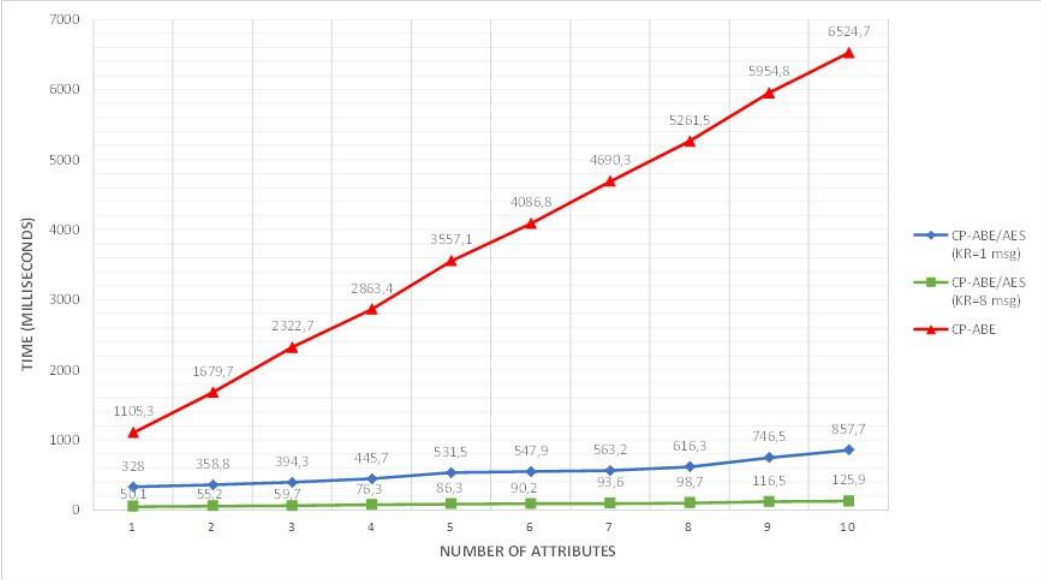

Figure 10 - Encryption Time vs Policy Complexity[2]

As evident from the measures in Figure 9 and Figure 10 the SeDEM approach of combining CP-ABE and AES is quite effective.

3.2.1.2.3.2. Summary of main functionalities

The SeDEM component has been conceived as a wrapper library intended to ease the publish and subscribe operations of AMQP clients of a RabbitMQ-based event

[2] actual measures on a Raspberry Pi 3 board

broker, hiding underlying details of the mechanisms transparently provided by the library to grant confidentiality in the execution of these operations.

The SeDEM library provides the following functionalities:

- *Event Creation*: the library enables the creation of the Event dispatched to the broker. An Event is made up by the following parts:
  - headers: a set of (key, value) pairs the publisher can use to indicate some data useful to enable subscribers to correctly understand the payload of the event and/or to put the informative content of the event;
  - payload: an opaque list of bytes that represents the informative content of the event. NOTE: the informative content carried by an event may be put in the headers hence the message payload may be empty.
- *Publishing*: provides the capability to send the generated Event to the event broker in a secure way.
- *Subscribing*: enables the creation of a consumer for a specific set of events dispatched through the event broker. The actual processing of the events is performed by registering an event listener that is invoked every time a new event has been received by the subscriber.

3.2.1.3. Task Orchestration for CPPS

3.2.1.3.1.1. Short Description of the component

It is a component at the Cloud level for the support of Field devices managed by "Homard IoT Devices Management platform". In details this component offers a complementary tool for the definition of logic via BPMs and subscription to events (observations) that can be linked to the mentioned BPMs but also to other REST-based API services. In particular, it is using Activiti BPM Engine core in conjunction with the new agents and connectors within Homard, to prepare it for IoT and CPPS systems to become deployed in the cloud infrastructure. The following section explains the main functions and components involved to make feasible the task orchestration for CPPS.

3.2.1.3.1.2. Summary of main functionalities

The main functions are:

- **Interconnection of IoT devices based on OMA LwM2M with BPMs**: In details, the BPMs supported are the Open Source Activiti project, which is presented in the architecture as a background component.

An adapter is provided for the Activiti BPM Engine module (Annex 5 explains in detail how Activiti has been extended as part of the Task Orchestration for CPPS) with the aim of adapting its functionality to the processes, measurements and values of the IoT and the industry. To facilitate its use, it is provided a graphical interface for the creation and management of rules and alerts within the Homard management platform.

Additionally, other proprietary solutions that enable the interconnection with over 2000 connectors for a wide task orchestration integrating ERPs such as SAP, and several online tools are also supported as Fujitsu RunMyProcess.

- **IoT devices management platform (Homard):** Task Orchestration for CPPS is directly built on top of Homard IoT devices management platform, therefore it offers all the benefits from Homard in order to interconnect IoT sensors based on OMA LwM2M from the field level to the Cloud platform. As a devices management platform it offers added value services to other middleware defined in the Factory level such as remote firmware upgrade, connectivity diagnostics, management support for sensors configurations such as calibration, data pooling frequency at the physical level (ADCs, GPIO interfaces etc.), security bootstrap (authentication of the different services that can access to the devices) and finally the configuration of the connectivity parameters for GPRS, WiFi and other wireless mediums (More details in the next subsection and in the Annex 7.)

- **Bootstrap and deployment of the IoT services over the Cloud infrastructure:** The logic set-up such as the definition of historical data storage, subscription to events, data logging for audits etc. require the deployment of micro-services built on containers such as OpenStack images and Dockers swarm. This Task Orchestration for CPPS component is also integrated with this Bootstrap service that facilitates the interconnection of the IoT devices with the different services required to deploy and run the defined logic. For example, the deployment of the container where the data will be periodically stored or the container that will contain an instance of Activiti to run the BPM logic for a specific use-case.

- **Complementary components for specific hardware:** Additionally, extra components have been defined for specific IoT devices, which offer new

capabilities such as Human-Machine interaction based on the emerging IoT protocol OMA GotAPI and Google Eddystone (commercially known as Physical Web).

- **Device URL Manager Component:** this component extends via Homard the capabilities of an IoT device to support the configuration of the Industrial Physical Web capabilities from a specific family of devices called beacons built on top of Bluetooth Low Energy. These specific components have been designed to facilitate the configuration of specific parameters such as the Web page announced in an area of the factory to offer the instructions / guidance and user interfaces to manage digitized machines. An example of this component is being used in Euskadi champion to simplify the use of 3D Scans from Trimek via a simplified mobile-driven user interface that the staff can access when they are nearby the machine. Thereby, offering an interface driven by proximity and context that simplifies the usual CAD tools for the visualization of the 3D models and control of the operations. The motivation of these particular components is to offer a better interface for connected works; since the particular nature and particularities of these devices; we have required to include plug-ins (complementary tools) as part of the Homard IoT device management component (More details in Annex 6.)
- **FIWARE IoT Agent OMA LwM2M for FIWARE Orion Context Broker:** As part of the bootstrap services mentioned in the previous component, an IoT Agent for OMA LwM2M has been also developed and integrated. IoT Agent for OMA is an enhanced version from the original IoT Agent develop from FIWARE in terms of reliability, solving some issues related with logging. Thereby, when a device requires transmitting an event to FIWARE, the Task Orchestration for CPPS in conjunction with the interconnected components will deploy a micro-container that includes the IoT Agent for OMA LwM2M in order to carry out the mapping and translation of the data into the correspondent OMA NGSI APIs and ETSI ISG CIM data models. Thereby, this component is also fully interconnected with the Factory level components based on FIWARE Orion Context Broker. An example of these devices based on OMA LwM2M interconnected

with FIWARE is the Industrial Smart Spot developed by HOP Ubiquitous that offers multiple sensing capabilities such as distance sensors, noise, temperature, humidity and industrial air quality; other examples of industrial OMA LwM2M devices are being produced by companies such as Bosch, Farine (http://farinehtech.com) etc.

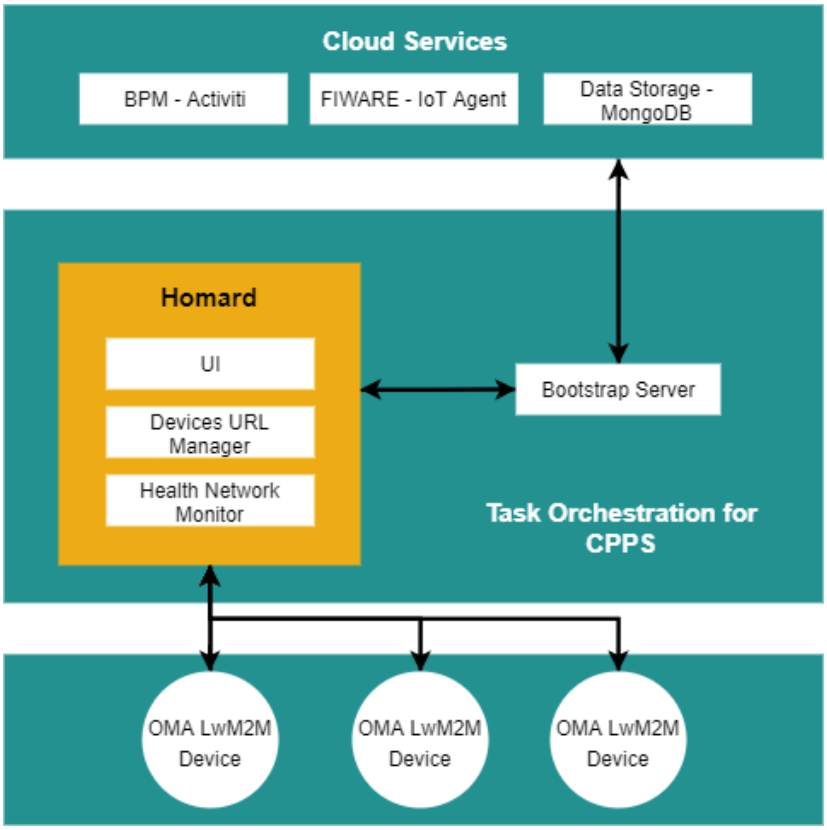

Figure 11 - Task Orchestration for CPPS Architecture

In definitive, the Task Orchestration for CPPS offers a cloud-level component to enable final services based on CPPS including IoT devices, for the particular case of OMA LwM2M protocol. This component takes benefit of the existing background components such as Activiti for the BPM-based logic and it is also integrated with other background components such as FIWARE Orion Context Broker to enable the OMA LwM2M devices support for the rest of components from BEinCPPS architecture.

The benefits of this component are the capacity to close the loop from sensing devices (IoT sensors) to action (BPM processes, events management and human interactions via industrial physical web).

This component is being used in Open Call 2 scenarios where HOP Ubiquitous's OMA LwM2M devices have been integrated via Task Orchestration for CPPS component, addressing mainly the parts defined by the OMA LwM2M IoT Agent and Homard IoT devices management platform. In the same way, Euskadi Champion takes benefit of Industrial Physical Web in order to interact with workers in an innovative and intuitive way, making use of the Device URL manager tool.

3.2.2. Proprietary components

3.2.2.1. Device URL Manager Component

3.2.2.1.1. Short Description of the component

Devices URL Manager Component as described is a complementary tool for managing Beacons devices to enable the Industrial Physical Web functionalities via the Homard IoT Devices Management platform as part of the Tasks Orchestration for CPPS.

Industrial Physical Web defines a new paradigm to make accessible Web Apps at mobile devices (smart phones, tablets etc.) via context and proximity. Thereby, Web-based interfaces are pushed and made accessible to people when they are nearby a machine or a specific zone to receive notifications, information, instructions and even actionable interfaces to interact with the physical environment via intuitive digital interfaces.

The benefits of Industrial Physical Web with respect to classical Mobile Apps or static Web pages is the capacity to adapt dynamically to the context, it means the capacity to offer a content based on the physical position of the worker, status of the machines and step in the production process. This adaptability simplifies the use of these interfaces by blue collar operators, that does not require to open explicitly a Web or App, and navigate through a wide number of options to find the machine that they have in front of them; this context and proximity-driven approach offers this direct access; in order to make their use more intuitive and direct.

For this purpose, every machine is integrated with a beacon (small and low-cost devices; this web page since is dynamic and content evolves with the status of the process, requires to be fully interconnected with the Task Orchestration for CPPS. For that reason, this component enables the capacity to define which URL (Web page locator) is announced at every moment by the beacons.

In definitive, Device URL Manager is used for administrating the URL broadcasted by the devices. This URL can be broadcasted by BLE or WiFi Direct and must be coded with the Eddystone URL protocol.

Eddystone URL is an open source protocol which google has taken the helm. Google is integrating it natively in some of its key applications such as Chrome, although in recent days other browsers (i.e., Opera) make use of this protocol.

Nowadays, there are more than 3 million of apps hosted on Google Play and most of them are used a few times for their temporal or location context and later they are forgotten, wasting resources on mobile devices or at the best case uninstalled. Google wants to solve this problem through Physical Web. This technology will allow service providers to interact with users depending on their location, temporality context or directly with the objects surrounding the user without the need of installing any application on their devices. These applications will be developed as progressive webs and they will allow the user to feel that they are interacting with the real world through native applications. These applications can directly interact with mobile device hardware or even receiving notifications.

3.2.2.1.2. Summary of main functionalities

Thanks to the Device URL Manager and technologies such as Physical Web, industry machinery will be able to broadcast a URL with temporal and spatial context that will send to the employee's device a control panel that will adjust to their own needs. This control panel will be capable of automating tasks and simplifying the tasks the operators have to perform.

3.2.2.2. Homard IoT Management for CPPS

HOMARD is a device management platform for the OMA LwM2M protocol. The platform offers functionalities for device management, i.e., remote maintenance, firmware upgrade and open/standard APIs for information reporting.

OMA LwM2M is the evolution of OMA Device Management (OMA DM) in order to attend the new requirements and constraints of the Internet of Things, OMA DM is the most extended protocol for remote device management worldwide being used in over 1,4 billions of devices, making OMA LwM2M a very relevant standard based on the experience and knowledge from the most validated and extended protocol for device management (firmware upgrade over the air, remote monitoring, remote reboot, maintenance etc.). In details, the operations offered by the device management platform are:

- Software Management: enabling the installation, removal of applications, and retrieval of the inventory of software components already installed on the device and the most relevant firmware upgrade over the air.
- Diagnostics and Monitoring: enabling remote diagnostic and standardized object for the collection of the memory status, battery status, radio measures, QoS parameters, peripheral status and other relevant parameters for remote monitoring.
- Connectivity and security: allowing the configuration of bearers (WiFi, Bluetooth, cellular connectivity), proxies, list of authorized servers for remote firmware upgrade and also all the relevant parameters for enabling secure communication.

- Device Capabilities: allowing the Management Authority to remotely enable and disable device peripherals like cameras, Bluetooth, USB, sensors (ultrasound, temperature, humidity, etc.) and other relevant peripherals from the nodes.
- Lock and Wipe: allowing to remotely lock and/or wipe the device, for instance when the device is lost (relevant for devices in open ocean, air etc.), or when the devices are stolen or sold. It enables the remote erase of personal / enterprise data when they are compromised.
- Management Policy: allowing the deployment on the device of policies which the client (node, device, sensor) can execute and enforce independently under some specific conditions, i.e., if some events happen, then perform some operations.

In addition to the functionalities, OMA LwM2M defines the semantics for the management objects. These objects have been defined with other standards organizations such as oneM2M and IPSO Alliance, which cooperate with OMA to avoid fragmentation and duplication that enables the semantic integration with the Management Objects. OMA LWM2M provides service providers with a secure, scalable, application-independent IoT control platform that provides control and security across multiple industries.

Thereby, this extension will also enable the integration into other initiatives such as oneM2M, which is the major initiative being led by ETSI and all the members from 3GPP to enable a worldwide architecture for Internet of Things. It has a special focus on Semantic Web and interoperability.

4. Conclusions

The previous sections have summarized the new or enhanced components within the BEinCPPS Factory Layer as compared to the ones described in the previous document (D2.3). Details about each component are provided in the dedicated appendixes.

This document has also framed the components according to the BEinCPPS architectural view detailed in D2.2.

Most of the D2.3 and D2.4 components have been actually validated in the different BEinCPPS champions and OC1 experimentations, and are being further validated in the current OC2 activities.

Even if there will be no additional deliverable focused on BEinCPPS Factory Level components after this D2.4, anyway they will be still maintained and supported taking into account the hints and feedbacks coming from validation activities both inside and outside BEinCPPS with the final objective to provide, at the end of the project, a set of functional components suitable to be exploited in many contexts.

5. References

[1] OPC Foundation, "*OPC Unified Architecture Specification - Part 1: Overview and Concepts, Release Candidate 1.04*", July 2017
[2] OPC Foundation, "*OPC Unified Architecture Specification - Part 14: PubSub, Release Candidate 1.04.24*", February 2017
[3] *"ISO/IEC 19464:2014 - Information technology -- Advanced Message Queuing Protocol (AMQP) v1.0 specification*", 2014
[4] http://www.openiot.eu/

Appendix 1 – CPPS Publishing Services factsheet

HW/SW Requirements

- The OPC UA Agent is a Node.js application, so installation can be performed either on **Windows**, **Linux** or **macOS** systems.
- **Node.js v4.4.3** or greater plus **npm** (Node Package Modules)**,** which is usually included with the Node.js package.
- Access to a **FIWARE OCB** instance, which can run either remotely on the clous or locally on the same hosting machine.
- In real world scenarios, a running OPC UA device; however, the software can be configured to run an internal device simulator, for NGSI integration testing purposes.

Installation Instructions

The following steps need to be performed to get the OPC UA Agent up and running:

1. Download the NodeOPCUA SDK [3] distribution from https://github.com/node-opcua/node-opcua/archive/v0.1.0-9.zip and extract the archive.
   OR
   Get the latest version of the SW directly from the GitHub repository (assuming the `git` client is installed on your development machine):

```
git clone https://github.com/node-opcua/node-opcua.git
```

2. Download the software from https://github.com/BEinCPPS/idas-opcua-agent/archive/v1.0.zip and extract the archive.
   OR
   Get the latest version of the SW directly from the GitHub repository (assuming the `git` client is installed on your development machine):

```
git clone https://github.com/BEinCPPS/idas-opcua-agent.git
```

[3] http://node-opcua.github.io/

3. Configure the installation. Your project structure should be as follows:

```
<your_project_dir>
|__ node-opcua (NodeOPCUA SDK distribution)
|__ idas-opcua-agent (OPC UA Agent)
```

4. Apply one of the two supported mapping methods:
   - Manual mapping

     Edit the *config-idas.json* configuration file; example:

```
{
    "id": "MyDevice1",
    "type": "teststation",
    "mappings": [{
      "ocb_id": "attrib1",
      "opcua_id": "ns=1;s=PumpSpeed"
    },
    {
      "ocb_id": "attrib2",
      "opcua_id": "ns=1;s=Temperature"
    }
    ]
  }
```

   - Automated mapping: browsing the Address Space of the OPC UA Server

     Edit the *config.js* configuration file. In this section of the file you have to define the Address Space Folder to browse on the OPC UA Server and the rules to follow for identify mappable data structures. Example:

```
browseServerOptions: {
      mainFolderToBrowse: 'TestStationFolder',
      eventNotifier: 'TestStationEventNotifier',
      mainObjectStructure: {
        namePrefix: 'TestStation', // devices
        variableType1: {
          name: 'variableType1',
          namePrefix: 'measure',
          type: 'integer',
          properties: []
        },
        variableType2: {
```

```
            name: 'variableType2',
            namePrefix: 'state',
            type: 'string',
            properties: [{
              name: 'statePayload',
              type: 'string'
            }]
          },
      },
    method: {
      namePrefix: 'Method',
      type: 'method'
    }
},
```

5. (*optional, only for testing*) Launch the simulated OPC UA Server
   - Open a terminal session
   - Navigate to the node-opcua directory:
     `cd <your_project_dir>/node-opcua`
   - Launch the simulated Server as a Node.js process:
     `node bin/simple_server.js`
   - Look at the on-screen log for any problems; take note of the endpoint in order to connect an OPC UA Agent instance (see next point).
   - Leave the terminal session running: you can shut down the Server anytime by pressing CTRL+C.

6. Launch the OPC UA Agent instance

   For a manual launch, follow the instructions below. To automate the process, possibly launching multiple processes for multiple OPC UA devices, you will have to use the scripting tools provided by your operating system of choice.
   - Identify the endpoint of the target OPC UA device; this will be in `opc.tcp://<server_name>:<port>` format.
   - Open a terminal session
   - Navigate to the opcua-agent directory:
     `cd <your_project_dir>/opcua-agent`
   - Launch the Agent as a Node.js process, passing the device's endpoint as a command line option:

```
node index.js -e "opc.tcp://localhost:26543"
```

Use the `-b` option if you want to "browse the address space"

- When done, terminate the process manually (`CTRL-C`).

Communication Security

The communication between the Server and the Agent may be optionally protected by the use of SSL connections. We recommend the use of a certificate provisioned by a trusted Certification Authority.

It is possible to reproduce this scenario in the enclosed simulated Server using a certificate. For demo purpose you can use a self-signed certificate. In the simulated Server edit the file `bin/simple_server.js` and add the right file path in the variables:

```
var server_certificate_file = path.join(__dirname,
"../certificates/server_selfsigned_cert_2048.pem"
```

```
var server_certificate_privatekey_file =
path.join(__dirname,"../certificates/server_key_2048.pem"
);
```

Appendix 2 – CEP Input Event Adapter Component

BEinCPPS specific developments

The module has been developed in Java and implements the WSO2 CEP InputEventAdapter interface as detailed in the Listing 1

```
public class RabbitMQEventAdapter implements InputEventAdapter
{
        private final InputEventAdapterConfiguration eventAdapterConfiguration;
        private final String id = UUID.randomUUID().toString();
        private RabbitMQAdapterListener rabbitmqAdapterListener;

        public RabbitMQEventAdapter(
                InputEventAdapterConfiguration eventAdapterConfiguration,
                Map<String, String> globalProperties) {
                        this.eventAdapterConfiguration = eventAdapterConfiguration;
                }

    public void init(InputEventAdapterListener eventAdapterListener)
    {
                RabbitMQBrokerConnectionConfiguration
            rabbitmqBrokerConnectionConfiguration = new
            RabbitMQBrokerConnectionConfiguration(eventAdapterConfiguration);
            rabbitmqAdapterListener = new
            RabbitMQAdapterListener(rabbitmqBrokerConnectionConfiguration,
            eventAdapterConfiguration, eventAdapterListener);
    }

    public void testConnect() throws TestConnectionNotSupportedException
    {
            throw new TestConnectionNotSupportedException("not-supported");
    }

    public void connect()
```

Listing 1 - Extract of InputEventAdapter implementation

```
            rabbitmqAdapterListener.createConnection();
    }

    public void disconnect()
    {
            if (rabbitmqAdapterListener != null) {
                    rabbitmqAdapterListener.stopListener(eventAdapterConfiguration.getName
            ());
            }
    }
}
```

The implementation of this adapter completes the connecting structure from the Factory Level to the Cloud Level.

HW/SW Prerequisite

The main prerequisite needed for successfully running the released component is the presence of WSO2 CEP. Prior to installing any WSO2 Carbon based product, it is necessary to have the appropriate prerequisite software installed on your system. Verify that the computer has the supported operating system and development platforms before starting the installation.

Hardware requirements

Memory	~ 2 GB minimum ~ 512 MB heap size. This is generally sufficient to process typical messages but the requirements vary with larger message sizes and the number of messages processed concurrently.
Disk	~ 1 GB minimum, excluding space allocated for log files and databases

Software requirements

Java	All WSO2 Carbon-based products are Java applications can be run on any platform that is Oracle JDK 1.8.x compliant. We do not recommend OpenJDK as we do not support it or test our products with it.
Databases	All WSO2 Carbon-based products are generally compatible with most common DBMSs. It is not recommended to use Apache DS in a production environment due to issues with scalability. Instead, it is recommended to use an LDAP like OpenLDAP for user management.
Operating Systems	WSO2 CEP does not support SunOS 5.10 and IBM AIX. For environments that WSO2 products are tested with, see Compatibility of WSO2 Products on the official website.
Web Browser	

Message Broker **WSO2 Products**	To access WSO2 product's Management Console. The Web Browser must be JavaScript enabled to take full advantage of the Management console. It is compatible with Firefox 39.0 or later, Chrome 43 or later, Safari 8.0.7 or later. **N.B**: WSO2 CEP does not support Internet Explorer browser. RabbitMQ 3.6.1 or later version. A RabbitMQ broker requires the installation of the Erlang/OTP run time (see the RabbitMQ documentation for details). WSO2 Complex Event Processor v4.2.0 or later downloadable at http://wso2.com/products/complex-event-processor/. Take into account that all the basic configurations and settings not concerned with the installation of the Input Stream Adapter (e.g. the Execution Plan and Event Stream definition) will not be detailed in this document. It is essential that the user be aware of how to create and execute a WSO2 CEP workflow. Details on how to create and manage these workflows are provided within the WSO2 CEP documentation to which the reader is referred.

Installation Instructions

The released component and resources are available in the provided installation package shared (available at https://github.com/BEinCPPS/). To successfully run the WSO2 CEP the released JAR and the related dependences libraries must be copied in a specific location.

In particular copy the content of the folder *libsWSO2\dropins*, in the provided package, to the *<WSO2 CEP HOME>\repository\components\dropins* folder. At the end of the copy procedure, this folder will contain the following jars:

- *amqp_client_3.6.1_1.0.0.jar*: the RabbitMQ Java client library necessary to manage the connection with RabbitMQ[4];

[4] It is important to use a RabbitMQ library in line with the version of the installed RabbitMQ Broker. Therefore, in case of use of a more recent version of RabbitMQ Broker download the related Java client library from the RabbitMQ web site

- *org.wso2.carbon.extension.analytics.receiver.rabbitmq.jar*: the developed component described in this report;
- *org.wso2.carbon.logging.propfile_1.0.0.jar*: provide the log4j logging capabilities, which generates administrative activities and server-side logs.

Moreover, to successfully execute WSO2 CEP workflows that use the provided Input Event Adapter an already running instance of the RabbitMQ Broker Server is required and the AMQP elements (i.e., the AMQP Queue) relevant for the Input Event Adapter must already exist and properly configured. To complete the configuration of the CEP Input Event Adapter the following parameters must be acquired from the RabbitMQ server:

- Virtual host address or symbolic link (e.g. 127.0.0.1 or localhost);
- the AMQP Queue it has to connect to in order to receive the events. Of course, the Queue must already exist, as well as the required AMQP bindings between the Queue and the AMQP Exchanges;
- the subscriber specific credentials (e.g., username/password) to be able to successfully connect to the RabbitMQ Broker.

Please refer to RabbitMQ website tutorial (https://www.rabbitmq.com/tutorials/amqp-concepts.html) for more details about the involved AMQP concepts.

User Manual

After following the steps described in the previous section, it is possible to start the WSO2 CEP through the following command in a Command Prompt window:

<WSO2 CEP HOME>\bin>wso2server.bat

When the WSO2 CEP is up and running, the Command Prompt window will display the Management Console URL to access the CEP Management console. The URL in question could be different for each run of WSO2 CEP, then it is important to wait the CEP startup and use the provided URL. Here is an example:

[2017-11-02 11:19:08,153]INFO {org.wso2.carbon.ui.internal.CarbonUIServiceComponent} - Mgt Console URL : https://172.25.13.206:9443/carbon/

After logging in, the Management Console Home will be appear as depicted in the following figure:

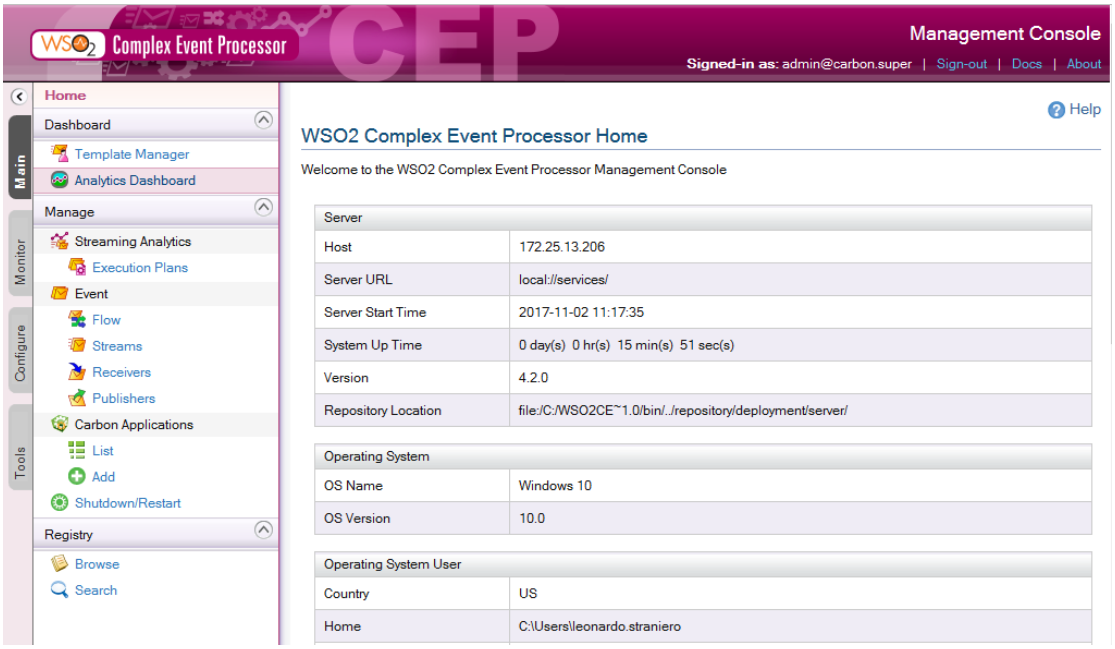

Figure 12 - The WSO2 CEP Management Console

Creating a WSO2 CEP workflow, as indicated in its documentation, has to start from the creation of a new, or the use of an existing, *Stream*. In order to use a data stream coming from RabbitMQ is therefore required to select or create a *Stream* and then define the events you intend to acquire (i.e., event's attributes, attribute's types, etc.). The following figure provides an example of an Event Stream.

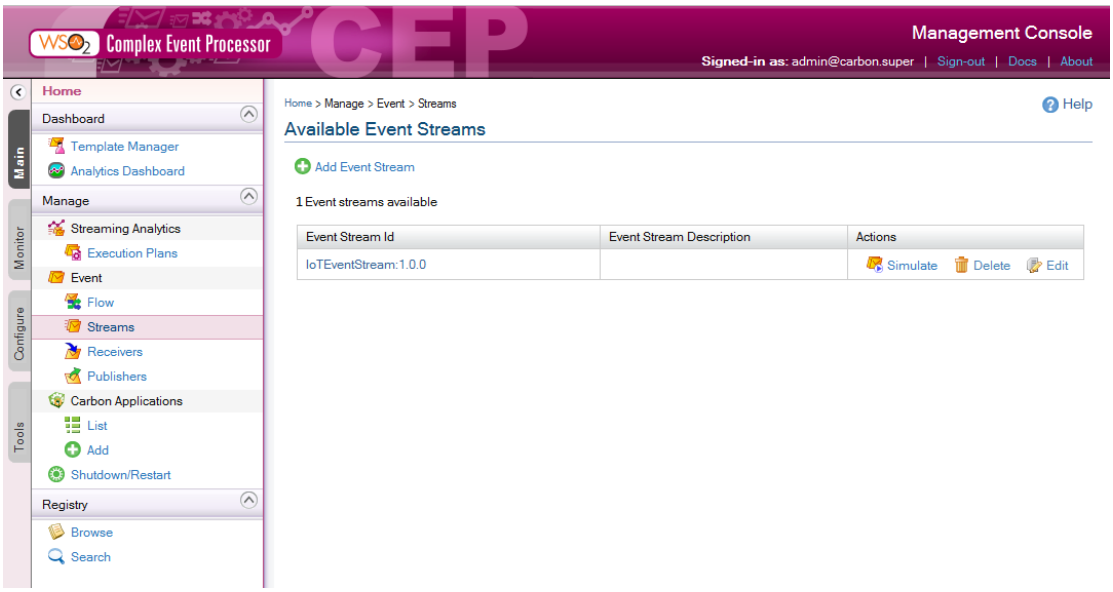

Figure 13 - WSO2 CEP Event Stream Definition

After having specified the kind of events to be acquired, select the *Receivers* section and the *Add Event Receiver* button.

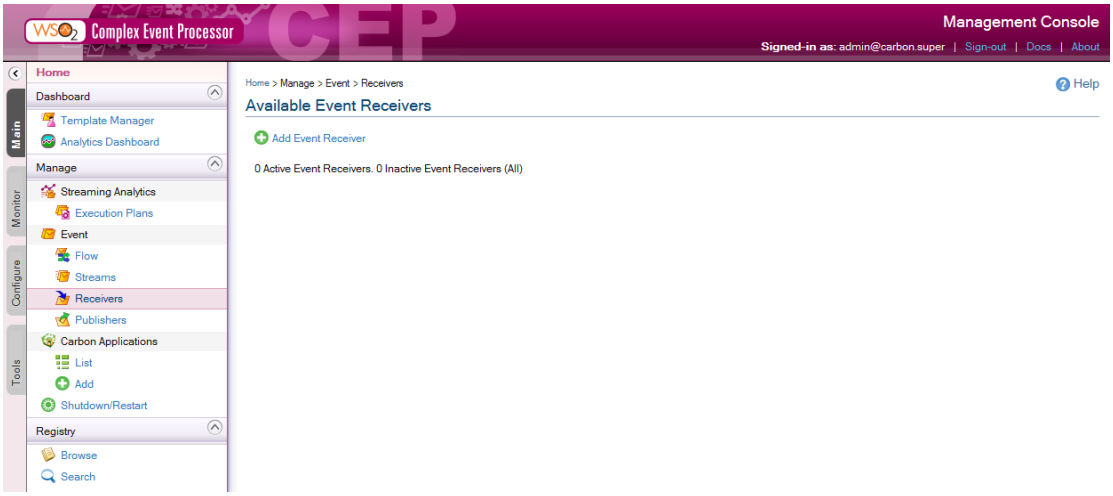

Figure 14 - WSO2 CEP Add Event Receiver

The Figure 15 contains the entire list of fields available to model a new Event Receiver. Not all the fields are required and all the fields provide a related help text that provides hints on how to fill up each field. The following list provides more details about the most significant fields:

- *Event Receiver Name*: a custom name to identify the new Event Receiver;
- *Input Event Adapter Type*: the type of the adapter. The installation of our component adds a new type of adapter (**wso2_rabbitmq**) that has to be selected to connect the Cep workflow to a RabbitMQ data stream;
- *Host Port*, *Host Name*, *Virtual Host*: the TCP port, host name or IP address and AMQP virtual host the CEP workflow has to connect to get the input data stream;
- *Username and Password*: credentials to be used to connect to the RabbitMQ instance;
- *Queue Name*: the name of the AMQP Queue through which input events will be received;
- *Event Stream*: select the Event Stream that will use these input data stream;
- *Message Format*: contains the format of the Event Message, in our case it is a JSON message.

All other fields can be left blank or with their default values.

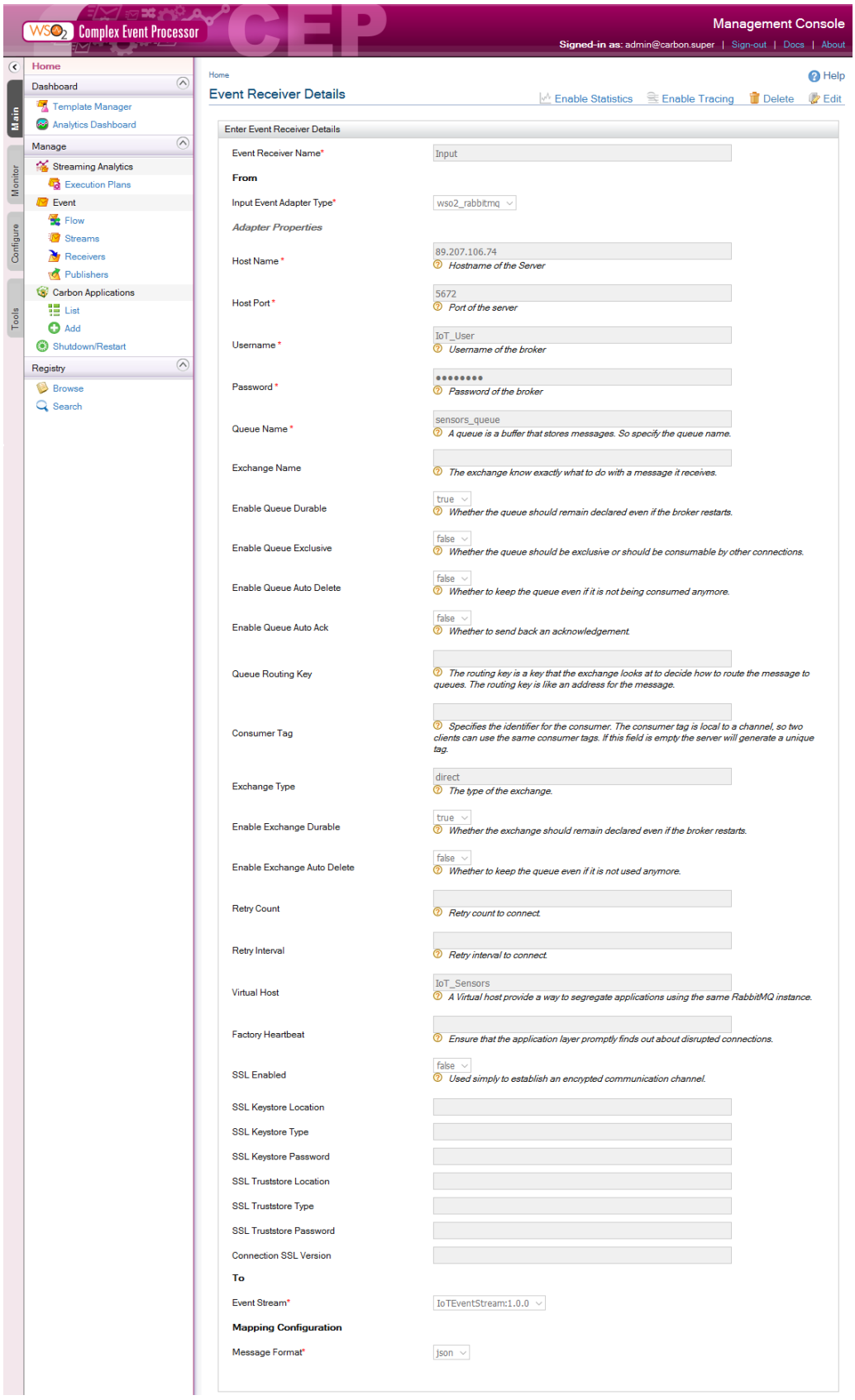
WSO2 Complex Event Processor
CEP
Management Console
Signed-in as: admin@carbon.super | Sign-out | Docs | About
Main
Monitor
Configure
Tools
Home
Dashboard
Template Manager
Analytics Dashboard
Manage
Streaming Analytics
Execution Plans
Event
Flow
Streams
Receivers
Publishers
Carbon Applications
List
Add
Shutdown/Restart
Registry
Browse
Search
Home
Help
Event Receiver Details
Enable Statistics
Enable Tracing
Delete
Edit
Enter Event Receiver Details
Event Receiver Name*
Input
From
Input Event Adapter Type*
wso2_rabbitmq
Adapter Properties
Host Name *
89.207.106.74
Hostname of the Server
Host Port *
5672
Port of the server
Username *
IoT_User
Username of the broker
Password *
Password of the broker
Queue Name *
sensors_queue
A queue is a buffer that stores messages. So specify the queue name.
Exchange Name
The exchange know exactly what to do with a message it receives.
Enable Queue Durable
true
Whether the queue should remain declared even if the broker restarts.
Enable Queue Exclusive
false
Whether the queue should be exclusive or should be consumable by other connections.
Enable Queue Auto Delete
false
Whether to keep the queue even if it is not being consumed anymore.
Enable Queue Auto Ack
false
Whether to send back an acknowledgement.
Queue Routing Key
The routing key is a key that the exchange looks at to decide how to route the message to queues. The routing key is like an address for the message.
Consumer Tag
Specifies the identifier for the consumer. The consumer tag is local to a channel, so two clients can use the same consumer tags. If this field is empty the server will generate a unique tag.
Exchange Type
direct
The type of the exchange.
Enable Exchange Durable
true
Whether the exchange should remain declared even if the broker restarts.
Enable Exchange Auto Delete
false
Whether to keep the queue even if it is not used anymore.
Retry Count
Retry count to connect.
Retry Interval
Retry interval to connect.
Virtual Host
IoT_Sensors
A Virtual host provide a way to segregate applications using the same RabbitMQ instance.
Factory Heartbeat
Ensure that the application layer promptly finds out about disrupted connections.
SSL Enabled
false
Used simply to establish an encrypted communication channel.
SSL Keystore Location
SSL Keystore Type
SSL Keystore Password
SSL Truststore Location
SSL Truststore Type
SSL Truststore Password
Connection SSL Version
To
Event Stream*
IoTEventStream:1.0.0
Mapping Configuration
Message Format*
json

Figure 15 - WSO2 CEP Create new Event Receiver

At the end of the new Event Receiver definition, click on the Add Event Receiver button and check if the just created receiver is active, looking if the number of active event receivers depicted in Figure 14 is increased.

Examples

The following image captured from a real WSO2 CEP workflow provides an example of usage of the released component and its definition. The released component works under the definition of the *Input* node as Event Receiver and represents the connection between a RabbitMQ Queue and the CEP Event Stream entry point making the events available to two classic WSO2 CEP Execution Plans.

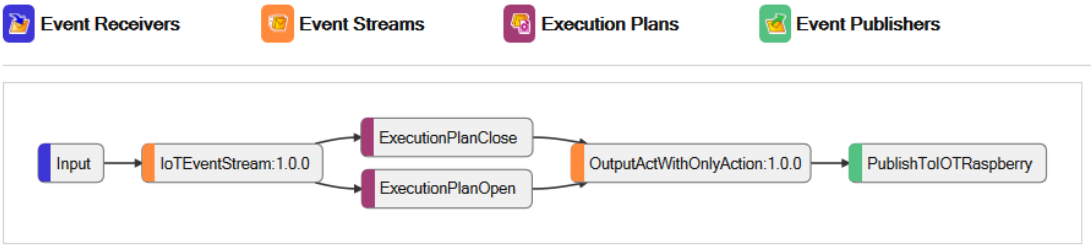

Figure 16 - Input Event Adapter example of usage

Licensing

The component released is licensed under the Apache License 2.0 (http://www.apache.org/licenses/), a permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Appendix 3 – CEP Output Event Adapter

BEinCPPS specific developments

The module has been developed in Java and implements the WSO2 CEP OutputEventAdapter interface as detailed in the Listing 1

```
public class RabbitMQEventAdapter implements OutputEventAdapter {
                private final OutputEventAdapterConfiguration eventAdapterConfiguration;
                private Map<String, String> globalProperties;
                private RabbitMQAdapterPublisher rabbitmqAdapterPublisher;
                private int connectionKeepAliveInterval;
                private String qos;
                private static ThreadPoolExecutor threadPoolExecutor;
                private static final Log log = LogFactory.getLog(RabbitMQEventAdapter.class);
                private int tenantId;

    public RabbitMQEventAdapter(OutputEventAdapterConfiguration eventAdapterConfiguration,
                    Map<String, String> globalProperties) {
this.eventAdapterConfiguration = eventAdapterConfiguration;
this.globalProperties = globalProperties;

Object keeAliveInternal = globalProperties.get(RabbitMQEventAdapterConstants.CONNECTION_KEEP_ALIVE_INTERVAL);

if (keeAliveInternal != null) {
    try {
            connectionKeepAliveInterval = Integer.parseInt(keeAliveInternal.toString());
    } catch (NumberFormatException e) {
    log.error("Error when configuring user specified connection keep alive time, using default value", e);
    connectionKeepAliveInterval = RabbitMQEventAdapterConstants.DEFAULT_CONNECTION_KEEP_ALIVE_INTERVAL;
}
} else {
   connectionKeepAliveInterval = RabbitMQEventAdapterConstants.DEFAULT_CONNECTION_KEEP_ALIVE_INTERVAL;
}
   }


 public void init()  throws OutputEventAdapterException {

   tenantId = PrivilegedCarbonContext.getThreadLocalCarbonContext(). getTenantId();

   //ThreadPoolExecutor will be assigned  if it is null
   if (threadPoolExecutor == null) {
```

```
            int minThread;
            int maxThread;
            int jobQueSize;
            long defaultKeepAliveTime;

            //If global properties are available those will be assigned else constant values will be assigned
            if (globalProperties.get(RabbitMQEventAdapterConstants.RABBITMQ_MIN_THREAD_POOL_SIZE_NAME) != null) {
                minThread = Integer.parseInt(globalProperties.get(RabbitMQEventAdapterConstants.RABBITMQ_MIN_THREAD_POOL_SIZE_NAME));
            } else {
                minThread = RabbitMQEventAdapterConstants.DEFAULT_MIN_THREAD_POOL_SIZE;
            }

            if (globalProperties.get(RabbitMQEventAdapterConstants.RABBITMQ_MAX_THREAD_POOL_SIZE_NAME) != null) {
                maxThread = Integer.parseInt(globalProperties.get(RabbitMQEventAdapterConstants.RABBITMQ_MAX_THREAD_POOL_SIZE_NAME));
            } else {
                maxThread = RabbitMQEventAdapterConstants.DEFAULT_MAX_THREAD_POOL_SIZE;
            }

            if (globalProperties.get(RabbitMQEventAdapterConstants.RABBITMQ_KEEP_ALIVE_TIME_NAME) != null) {
                defaultKeepAliveTime = Integer.parseInt(globalProperties.get(
                RabbitMQEventAdapterConstants.RABBITMQ_KEEP_ALIVE_TIME_NAME));
            } else {
                 defaultKeepAliveTime = RabbitMQEventAdapterConstants.DEFAULT_KEEP_ALIVE_TIME_IN_MILLIS;
            }

            if (globalProperties.get(RabbitMQEventAdapterConstants.RABBITMQ_EXECUTOR_JOB_QUEUE_SIZE_NAME) != null) {
                jobQueSize = Integer.parseInt(globalProperties.get(
                RabbitMQEventAdapterConstants.RABBITMQ_EXECUTOR_JOB_QUEUE_SIZE_NAME));
            } else {
                jobQueSize = RabbitMQEventAdapterConstants.DEFAULT_EXECUTOR_JOB_QUEUE_SIZE;
            }

            threadPoolExecutor = new ThreadPoolExecutor(minThread, maxThread, defaultKeepAliveTime,
            TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>(jobQueSize));
        }
    }


        @Override
        public void testConnect() throws TestConnectionNotSupportedException {
            throw new TestConnectionNotSupportedException("Test connection is not available");
        }
```

```
        public void connect() {
          //rabbitmqAdapterListener.createConnection();
          RabbitMQBrokerConnectionConfiguration rabbitmqBrokerConnectionConfiguration =
          new RabbitMQBrokerConnectionConfiguration(
           eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_SERVER_HOST_NAME),
           eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_SERVER_PORT),
           eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_SERVER_VIRTUAL_HOST),
           eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_SERVER_USERNAME),
           eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_SERVER_PASSWORD),
           eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_EXCHANGE_DURABLE),
           connectionKeepAliveInterval,
           eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_CONF_CLEAN_SESSION)
           );

           qos = eventAdapterConfiguration.getStaticProperties().get(RabbitMQEventAdapterConstants.RABBITMQ_MESSAGE_QOS);
           rabbitmqAdapterPublisher = new RabbitMQAdapterPublisher(rabbitmqBrokerConnectionConfiguration,
eventAdapterConfiguration);
        }

        @Override
        public void publish(Object message, Map<String, String> dynamicProperties) {
          try {
                threadPoolExecutor.submit(new RabbitMQSender(/*topic, */message));
              } catch (RejectedExecutionException e) {
                EventAdapterUtil.logAndDrop(eventAdapterConfiguration.getName(), message, "Job queue is full", e, log, tenantId);
          }
        }

        @Override
        public void disconnect() {
          try {
            if (rabbitmqAdapterPublisher != null) {
              rabbitmqAdapterPublisher.close(eventAdapterConfiguration.getName());
              rabbitmqAdapterPublisher = null;
            }
          } catch (OutputEventAdapterException e) {
            log.error("Exception when closing the rabbitmq publisher connection on Output RabbitMQ Adapter '" +
              eventAdapterConfiguration.getName() + "'", e);
          }
        }

        class RabbitMQSender implements Runnable {

          Object message;
```

```
        RabbitMQSender(/*String topic, */Object message) {
            this.message = message;
        }

        @Override
        public void run() {
            try {
                if (qos == null) {
                    rabbitmqAdapterPublisher.publish(message.toString()/*, topic*/);
                } else {
                    rabbitmqAdapterPublisher.publish(Integer.parseInt(qos), message.toString()/*, topic*/);
                }
            } catch (Throwable t) {
                EventAdapterUtil.logAndDrop(eventAdapterConfiguration.getName(), message, null, t, log, tenantId);

            }
        }
    }

[.....]}
```

Listing 2 - Extract of OutputEventAdapter project implementation

The implementation of this adapter completes the connecting structure from the Cloud Level to the Factory Level.

HW/SW Prerequisite

The main prerequisite needed for successfully use the released component is the presence of the WSO2 CEP. Prior to installing any WSO2 Carbon based product, it is necessary to have the appropriate software prerequisites. Verify that the computer has the supported operating systems and run time components before starting the installation.

Hardware requirements

Memory	~ 2 GB minimum ~ 512 MB heap size. This is generally sufficient to process typical messages but the requirements vary with larger message sizes and the number of messages processed concurrently.
Disk	~ 1 GB minimum, excluding space allocated for log files and databases

Software requirements

Java	All WSO2 Carbon-based products are Java applications can be run on any platform that is Oracle JDK 1.8.x compliant. We do not recommend OpenJDK as we do not support it or test our products with it.
Databases	All WSO2 Carbon-based products are generally compatible with most common DBMSs. It is not recommended to use Apache DS in a production environment due to issues with scalability. Instead, it is recommended to use an LDAP like OpenLDAP for user management.
Operating Systems	WSO2 CEP does not support SunOS 5.10 and IBM AIX. For environments that WSO2 products are tested with, see Compatibility of WSO2 Products on the official website.
Web Browser	To access WSO2 product's Management Console. The Web Browser must be JavaScript enabled to take full advantage of the Management console. It is compatible with Firefox 39.0 or later, Chrome 43 or later, Safari 8.0.7 or later. **N.B**: WSO2 CEP does not support Internet Explorer browser.
Message Broker	RabbitMQ 3.6.1 or later version. A RabbitMQ broker requires the installation of the Erlang/OTP run time (see the RabbitMQ documentation for details).
WSO2 Products	WSO2 Complex Event Processor v4.2.0 or later downloadable at http://wso2.com/products/complex-event-processor/. Take into account that all the basic configurations and settings not concerned with the installation of the Output Stream Adapter (e.g. the Execution Plan and Event Stream definition) will not be detailed in this document. It is essential that the user be aware of how to create and execute a WSO2 CEP workflow. Details on how to create and manage these workflows are provided within the WSO2 CEP documentation to which the reader is referred.

Installation Instructions

The released component and resources are available in the provided installation package (available at https://github.com/BEinCPPS/). To successfully run the WSO2 CEP the released JAR and the related dependences libraries must be copied in a specific location.

In particular copy the content of the folder *libsWSO2\dropins*, in the provided package, to the *<WSO2 CEP HOME>\repository\components\dropins* folder. At the end of the copy procedure, this folder will contain the following jars:

- *amqp_client_3.6.1_1.0.0.jar*: the RabbitMQ Java client library necessary to manage the connection with RabbitMQ[5];
- *org.wso2.carbon.event.output.adapter.rabbitmq.jar*: the developed component described in this report;
- *org.wso2.carbon.logging.propfile_1.0.0.jar*: provides the log4j logging capabilities, which generates administrative activities and server-side logs.

Moreover, to successfully execute WSO2 CEP workflows that use the provided Output Event Adapter an already running instance of the RabbitMQ Broker Server is required and the AMQP elements (i.e., the AMQP *Exchange*) relevant for the Output Event Adapter must already exist and properly configured. To complete the configuration of the CEP Output Event Adapter the following parameters must be acquired from the RabbitMQ server:

- Virtual host address or symbolic link (e.g. 127.0.0.1 or localhost);
- the AMQP *Exchange* it has to connect to in order to publish the events. Of course, the *Exchange* must already exist;
- the publisher specific credentials (e.g., username/password) to be able to successfully connect to the RabbitMQ Broker.

Please refer to RabbitMQ website tutorial (https://www.rabbitmq.com/tutorials/amqp-concepts.html) for more details about the involved AMQP concepts.

[5] It is important to use a RabbitMQ library in line with the version of the installed RabbitMQ Broker. Therefore, in case of use of a more recent version of RabbitMQ Broker download the related Java client library from the RabbitMQ web site

User Manual

After following the steps described in the previous section, it is possible to start the WSO2 CEP through the following command in a Command Prompt window:

<WSO2 CEP HOME>\bin>wso2server.bat

When the WSO2 CEP is up and running, the Command Prompt window will display the Management Console URL to access the CEP Management console. The URL in question could be different for each run of WSO2 CEP, then it is important to wait the CEP startup and use the provided URL. Here is an example:

[2017-11-02 11:19:08,153]INFO {org.wso2.carbon.ui.internal.CarbonUIServiceComponent} - Mgt Console URL : https://172.25.13.206:9443/carbon/

After logging in, the Management Console Home will appear as depicted in the following figure:

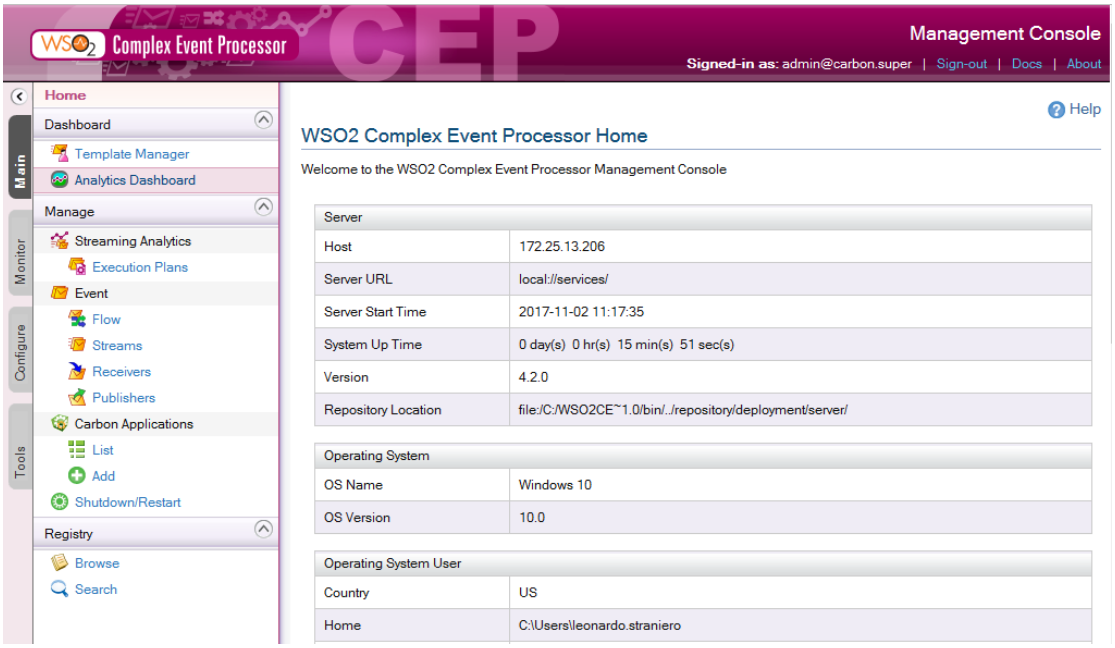

Figure 17 - The WSO2 CEP Management Console

Creating a WSO2 CEP workflow, as indicated in its documentation, has to start from the creation of a new, or the use of an existing, *Stream*. In order to use a data stream publishing events to RabbitMQ is therefore required to select or create a

Stream and then define the events you intend to provide (i.e., event's attributes, attribute's types, etc.). The following figure provides an example of an Event Stream.

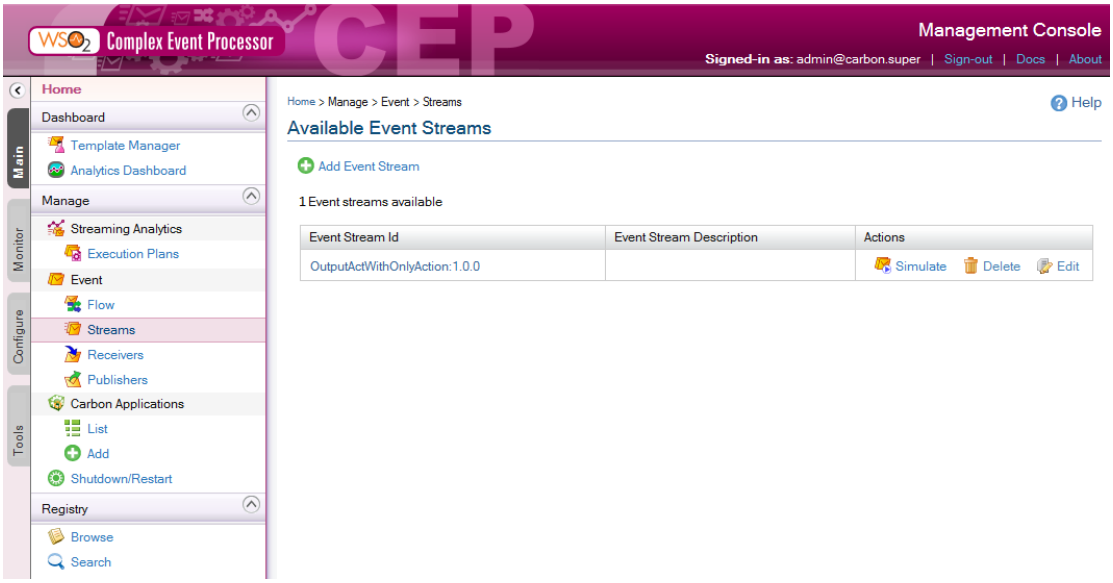

Figure 18 - WSO2 CEP Event Stream Definition

After having specified the kind of events to be provided, select the *Publishers* section and the *Add Event Publisher* button.

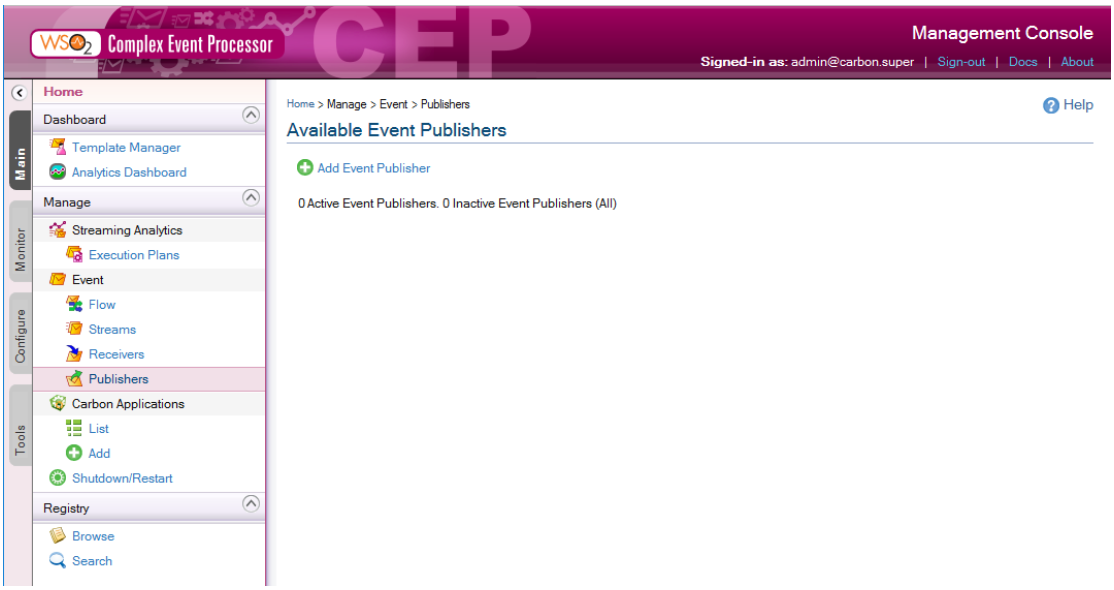

Figure 19 - WSO2 CEP Add Event Publisher

The Figure 20 contains the entire list of fields available to configure a new Event Publisher. Not all the fields are required and all the fields provide a related help text

that provides hints on how to fill up them. The following list provides more details about the most significant fields:

- *Event Publisher Name*: a custom name to identify the new Event Publisher;
- *Event Stream*: select the Output Event Stream that will provide the output data stream; it is possible specify some attributes filling the *Stream Attributes* field;
- *Output Event Adapter Type*: the type of the adapter. The installation of our component adds a new type of adapter (**wso2_rabbitmq**) that has to be selected to connect the Cep workflow to a RabbitMQ event broker;
- *Host Port, Host Name, Virtual Host*: the TCP port, host name or IP address and AMQP virtual host the CEP workflow has to connect to;
- *Username and Password*: credentials to be used to connect to the RabbitMQ instance;
- *Exchange Name*: the name of the AMQP *Exchange* through which output events will be published;
- *Queue Routing Key:* a value specifying the routing key to be used by the event broker to route the event to interested subscribers[6];
- *Message Format*: contains the format of the Event Message, in our case it is a JSON message.

All other fields can be left blank or with their default values.

[6] This *Queue Routing Key* is a default value that the CEP Output Event Adapter will use. Within the CEP workflow the *Routing Key* can be dynamically assigned so to give the possibility to change the intended event's consumers according to the specific output event

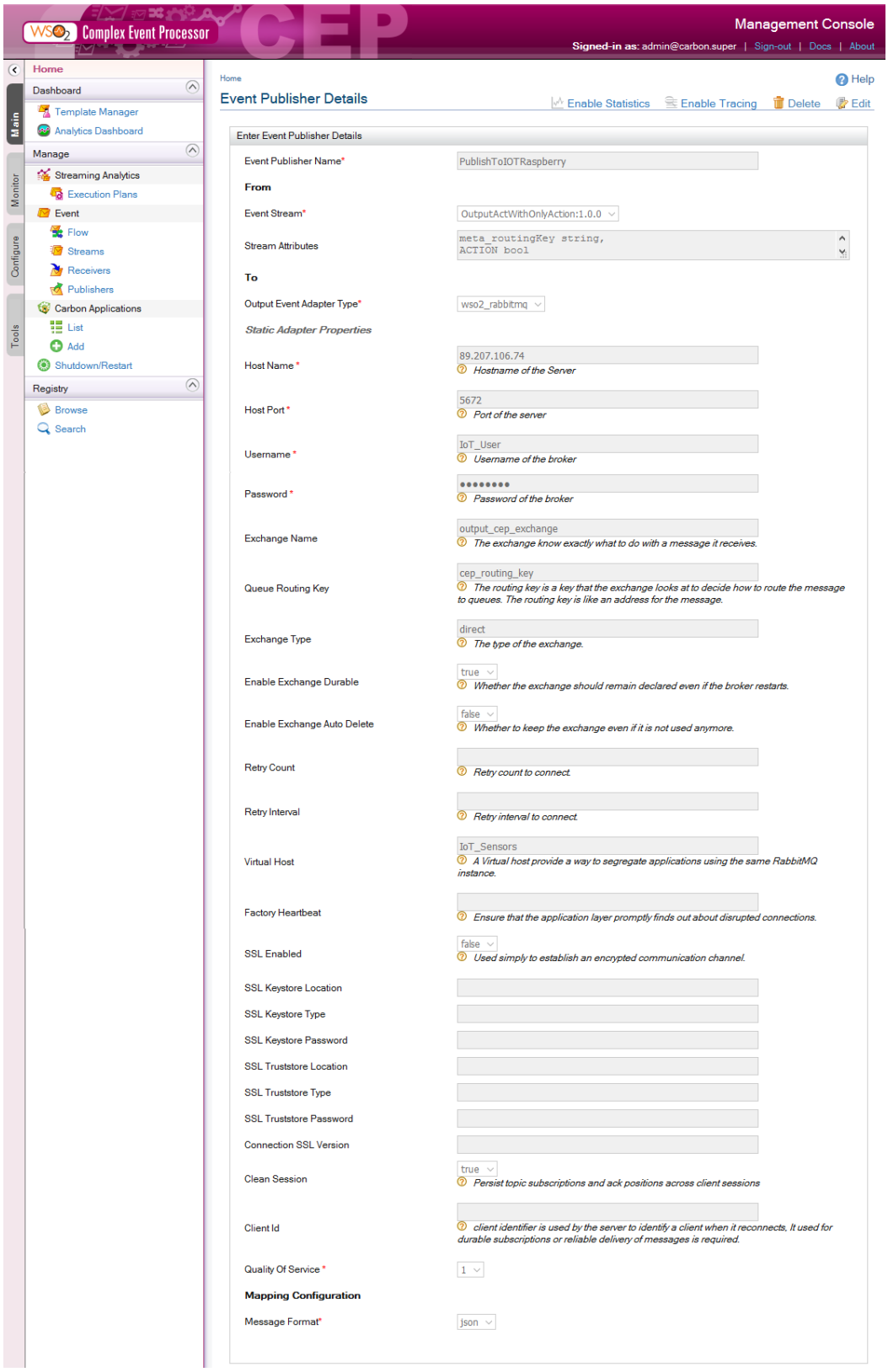

Figure 20 - WSO2 CEP Create a new Event Publisher

At the end of the new Event Publisher definition, click on the Add Event Publisher button and check if the just created publisher is active, looking if the number of active event publishers depicted in Figure 19 is increased.

Examples

The following image captured from a real WSO2 CEP workflow provides an example of usage of the released component and its definition. The released component works under the definition of the *Publisher* node as Event Publisher and represents the connection between the CEP Event Stream output and a RabbitMQ broker and making the generated events available to external systems or services as patterns are identified in the input streams (e.g. Data Stores, Dashboards, External Event Syncs, etc.).

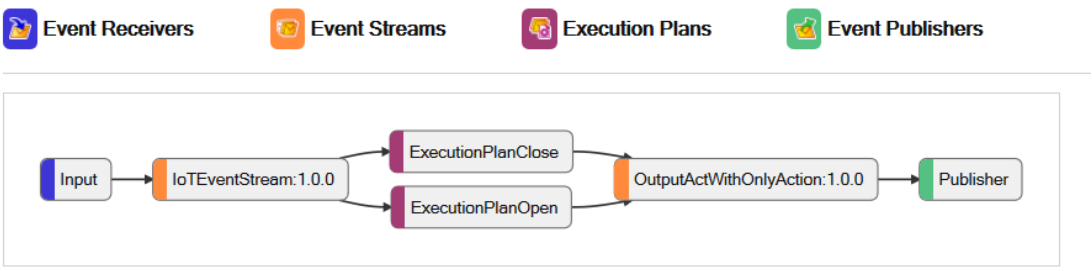

Figure 21 - Output Event Adapter example of usage

Licensing

The component released is licensed under the Apache License 2.0 (http://www.apache.org/licenses/), a permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Appendix 4 – Secure Data Exchange Middleware

BEinCPPS specific developments

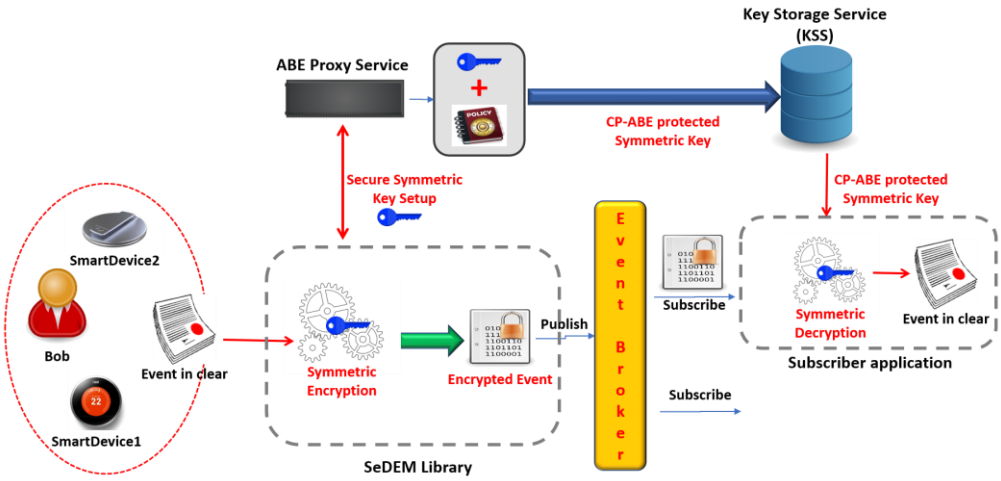

Figure 22 - SeDEM internal architecture

The SeDEM Library, represented in the Figure 22 by the grey dashed line, is located between the event broker and the data sources (on the left in the figure), to enable the publishing of Event in a confidential way, as well as between the event broker and the data consumers (on the right in the figure). The SeDEM Library is complemented by the ABE Proxy Service component which, as reported before, supports the data sources on the CP-ABE encryption activity which are more computationally intensive therefore supporting also devices with limited capabilities or contexts with an elevated data rate.

In the figure the different components have the following role and functionalities:

- *Data Source/Consumer Devices*: the events' sources or consumers, respectively usually located in the BEinCPPS Field Level and Cloud Level, that use the SeDEM Java classes to provide or consume events in a confidential way;
- *ABE Proxy*: the component responsible to perform the heavy ABE encryption operations;
- *Key Storage Service* (KSS): a storage component (or any other suitable mean) through which AES ephemeral symmetric keys encrypted via CP-ABE are made available to subscribers.

HW/SW Prerequisite

The BEinCPPS SeDEM component do not have particular hardware requirements. The following table reports the software dependencies:

Java	The component can be run on any platform that is Oracle JDK 1.8.x compliant. We do not recommend OpenJDK as we do not support it or test our products with it.
Message Broker	RabbitMQ 3.6.1 or later version. A RabbitMQ broker requires the installation of the Erlang/OTP run time (see the RabbitMQ documentation for details).
Abe Proxy	As depicted in the SeDEM internal architecture (see Figure 22) the Abe Proxy instance (whit the relative dependencies) plays a primary role during the event messages encryption and decryption phase.
CP-ABE Keys	• the data source or data consumer CP-ABE private key based on the attributes associated to the data consumer's profile; • the CP-ABE public key generated by the CP-ABE Setup operation to the ABE Proxy and used to perform ABE encryption and decryption operations. It is important to generate the public and private key through the CP-ABE Service and locate these keys in the path specified in the User Manual section. Please refer to the technical manual of CP-ABE for more details about the concepts involved in CP-ABE scenarios. For test purpose in the toolkit will be provided valid keys and policy.
Additional Dependencies	Log4J for logging purpose Java Cryptography Extension (JCE) Unlimited Strength

Installation Instructions

To facilitate the installation and configuration of the SeDEM component, the library is released with a toolkit (available at https://github.com/BEinCPPS/) containing:

- the library in the form of a JAR file (not executable) to be imported into the project where you want to use the features made available by the library;
- a zip file containing the resource folder with the settings files already set for a proper running under the Windows operating system to be unpacked in the

folder where you want to insert the various configurations of the library. If you want to use a different destination, change any occurrence of "*C:/resources/*" in the *conf.propertis* file, replacing them with the path where the zip was unpacked;

- two Java classes for testing (they use the code portions reported in the Example section):
  - to test the publisher part, you must run the *TestPublishCph_v2.java* class;
  - to test the subscriber's part, you must run the *TestSubscribeCph_v2.java* class.

To successfully execute the SeDEM Library an already running instance of the RabbitMQ Broker Server is required and the AMQP elements (i.e., the AMQP Queue, Exchange, Virtual Host, etc.) relevant for the SeDEM Library must already exist and properly configured. To complete the configuration of the components the following parameters must be acquired from the RabbitMQ server to be used in the next configuration step:

- AMQP Virtual host address or symbolic link (e.g. 127.0.0.1 or localhost);
- for data sources the AMQP Exchange they have to connect to in order to publish the events. Of course, the Exchange must already exist;
- for data consumers the AMQP Queue they have to connect to in order to receive the events. Of course, the Queue must already exist, as well as the required AMQP bindings between the Queue and the AMQP Exchanges;
- the publisher/subscriber specific credentials (e.g., username/password) to be able to successfully connect to the RabbitMQ Broker.

Please refer to RabbitMQ website tutorial (https://www.rabbitmq.com/tutorials/amqp-concepts.html) for more details about the involved AMQP concepts.

User Manual

To allow a dynamic configuration of the Logger and the Library itself, there are two configuration files located outside the library and provided in the resource folder in the Installation Toolkit:

- *log4j.configuration*: configures the Logger to allow info and message logging on a text file (or other destinations, such as a standard console or an auditing system) about the operations performed by the library. In that file, you can choose the destination of the log file that will be created. At this time, the Logger is set to print its output on console and text files.
- *conf.properties*: contains the Library configuration and the connection parameters to the mentioned broker.

Being external files, it is imperative to pass their absolute paths as "VM argument". Here is an example showing how to do this:

-Drabbitmq_config_file=C:\resources\conf.properties

-Dlog4j.configuration=file:C:\resources\log4j.properties

NOTE: for Linux-based system the path must use slash ('/') character instead of backslash ('\') as path separator.

Then, the first step to do before using the library is to modify the configuration files described above. Edit the *conf.properties*[7] file located in the resource folder and properly set the following parameters:

- *client.subjectID*: username authorized to connect to the RabbitMQ Server instance and the Virtual Host specified
- *client.accessToken*: password associated to the aforementioned username
- *client.portNumber*: AMQP port bound to RabbitMQ Service
- *client.host*: IP address of RabbitMQ Server
- *client.vhost*: AMQP Virtual Host where to publish of from which acquire the events
- *client.destinationName*: The name of the exchange / queue used for publisher / subscriber transactions. This parameter is used when the two specific parameters *client.exchange* and *client.queue* are not specified.
- *client.exchange*: name of the AMQP Exchange used for publishing (it defaults to the value of '*destinationName'*).
- *client.queue*: AMQP Queue name used for subscribe operations (it defaults to the value of '*destinationName'*).
- *client.tls*: flag used to specify if RabbitMQ connection must be established over a TLS channel
- *client.pattern*: routing key string.
- *client.cph*: flag to switch on the secure exchange of events: The value "true" activates the SeDEM encryption feature, while the value "false" transmits the events unencrypted
- *client.policy*: the CP-ABE policy to be sued to protect the AES ephemeral keys
- *consumer.public_params*[8]: the location of CP-ABE public key

[7] be aware that for simplicity and to avoid creating two separate files, the configuration file contains useful settings both for a publisher and subscriber client

- *consumer.cpabe_priv_key*: the location of CP-ABE private key

The data source clients need additional information related to the generation of the ephemeral keys (e.g., selected ECC curve, ABE Proxy information, Key Storage Service information, etc.) specified through the following parameters[9]:

- *client.alg*: indicates the algorithm used to generate the shared symmetric key. In this case, we have selected *ECDH-ES* (Elliptic Curve Diffie-Hellman Ephemeral Static key agreement using Concat KDF);
- *client.enc*: identifies the algorithm to be used to encrypt the data. Currently, we have chosen *A128GCM* (AES GCM with 128-bit key);
- *client.kty*: identifies the key type. For our case, this is Elliptic Curve (*EC*);
- *client.crv*: specifies the used cryptographic curve. We have selected the P-256 curve (*secp256r1*);
- *client.anticipated_key*: if this parameter is true, the new key is calculated before the current key expires;
- *client.anticipated_key_seconds*: specifies how many seconds in advance of the AES key expiration the generation of a new ephemeral key as to be started;
- *client.proxy_id*, *client.proxy_ip* and *client.proxy_port*: the identity name, the IP address and the port number of the ABE Proxy;
- *client.storage_type*: the type of storage where the CP-ABE encrypted ephemeral keys must be made available to data consumers. Currently this value must be *database* and the parameters for the used database[10] must be specified;
- *client.db_ip* and *client.db_port*: the ip address and the port number of the Key Storage Service;
- *client.db_auth_user* and *client.db_auth_pwd*: the credentials used to connect with the Key Storage Service;
- *client.db_database* and *client.db_table*: the database and table name.

The following parameters are relating to the testing software included in the overall package:

- *maxRnd*: upper value for the randomly generated events

[8] For a right interaction between the SeDEM library and the Abe Proxy instance it is important that the two involved parties use the same CP-ABE public key

[9] Most of the security parameters have been selected taking into account the US NIST recommendations and IETF standards

[10] Currently the used database is the OrientDB Graph database

- *minRnd*: bottom value for the randomly generated events
- *delay*: the pace, in milliseconds, at which events must be generated
- *px_data*: number of random events to be generated by the testing software.

An example of *conf.properties* file is reported below:

client.subjectID=username

client.accessToken=P4ssw0Rd

client.portNumber=5672

client.mgmPortNumber=15672

client.host=89.207.106.74

client.vhost=VHostName

client.destinationName=destination_name

client.exchange=exchange_name

client.queue=queue_name

client.pattern=routing_key

client.tls=false

EXTRA PUBLISHING SETTINGS

maxRnd=20

minRnd=10

#in ms

delay=2000

px_data=10

SECURITY SETTING

client.cph=true

client.anticipated_key=false

client.anticipated_key_seconds=3

client.alg=ECDH-ES

client.enc=A128GCM

client.kty=EC

client.crv=P-256

[.....]

#the information about the ABE Proxy and the Key Storage Service

client.proxy_ip=localhost

client.proxy_port=8080

client.proxy_id=ABE_PROXY

client.storage_type=database

client.db_ip=89.207.106.75

client.db_port=2480

client.db_auth_user=db_username

client.db_auth_pwd=db_password

client.db_database=CpAbeKeyStoreDB

client.db_table=EncSymKeys

DATA CONSUMER (for decryption data purpose)

client.policy=att1:val1 att2:val2 2of2 att3:val3 2of2

consumer.public_params=C:/resources/setup/keys/pub_10

consumer.cpabe_priv_key=C:/resources/setup/keys/key_10

In the library's resources folder, there is also a configuration file named *log4j.properties*. This configuration file includes parameters that specify the location of the output file, the maximum size of the file before a new file is created and the detail level of the Logger. Here's an example of the configuration file:

 # Root logger option

log4j.rootLogger=INFO, stdout, file

 # Redirect log messages to console

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

 # Redirect log messages to a log file, support file rolling.

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=C\:\\resources\\sedem_logging.log

#log4j.appender.file.File=/opt/logs/sedem_logging.log

log4j.appender.file.MaxFileSize=5MB

log4j.appender.file.MaxBackupIndex=10

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

NOTE: for Linux-based system the path must use *slash ('/')* character instead of *backslash* ('\') as path separator.

Developers' Guide

The Java documentation of this software, useful to use or extend the library, is available in the Javadoc folder available at https://github.com/BEinCPPS/

Examples

The released component is a java library, so given the nature of the library, will be provided some Java Snipped code about class library usage.

We provide here examples on how to make use of the library to:

1. Create an event
2. Publish events to the RabbitMQ Broker Server
3. Subscribe and receive events from RabbitMQ Broker Server

The provided examples simulate measures of Voltage, Power and Current sensors.

1. Event creation

Listing 3 reports an example of Event creation with a random value generator:

```
private static Event createEvent (int min, int max) {

        StringBuffer payload = new StringBuffer();
        Map headers = new HashMap();

        TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
        headers.put("timestamp", System.currentTimeMillis());

        float value = min + (random.nextFloat() * (max-min) + 1);
        payload.append("Power: " + value + "\n");

        value = min + (random.nextFloat() * (max-min) + 1);
        payload.append("Voltage: " + value + "\n");

        value = min + (random.nextFloat() * (max-min) + 1);
        payload.append("Current: " + value + "\n");

        try {
            return new BasicEventFactory().create(
                    headers,payload.toString().getBytes(Event.DEFAULT_CONTENT_ENCODI
            NG), false);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
}
```

Listing 3 - Event Creation example code

2. Publish

As detailed inListing 4, use the *RequestFactory* class to retrieve a client instance of *Publisher* passing an operation of type 'PUBLISH'. To test periodical publishing functionality, you can create a thread with 1 second of sleeping time:

```
public static void main(String[] args) {

        RabbitMqClient client =
    RequestFactory.startGuestApplication(ApplicationPropertiesRepository.PUBLISH);
```

```
        if(client.isConnected() && client instanceof Publisher){
                publishingRunnable = new PublishingRunnable((Publisher)client);
                publishingThread = new Thread(publishingRunnable);
                if(client.isConnected())
                        publishingThread.start();
        }
}
```

Listing 4 - Event Publishing example code

Here is detailed the *PublishingRunnable* class used in the previous code portion containing the logic for random event simulation at regular time intervals:

```
private static class PublishingRunnable implements Runnable{
        /* Time between each publication */
        private static final long DELAY =
                Long.parseLong(ApplicationPropertiesRepository.APPLICATION_PROPERTIES
                .getProperty(
                RequestFactory.ExtraSettings4Publisher.DELAY));

        /* Number of publications */
        private static final long PX_DATA =
                Long.parseLong(ApplicationPropertiesRepository.APPLICATION_PROPERTIES.get
        Property(
                        RequestFactory.ExtraSettings4Publisher.PX_DATA));

        /* min random generated measurement */
        int min_random = new
        Integer(ApplicationPropertiesRepository.APPLICATION_PROPERTIES
        .getProperty(RequestFactory.ExtraSettings4Publisher.MIN_RANDOM));

        /* max random generated measurement */
        int max_random = new
        Integer(ApplicationPropertiesRepository.APPLICATION_PROPERTIES
        .getProperty(RequestFactory.ExtraSettings4Publisher.MAX_RANDOM));
```

```
        private long num_publication = 0;
        private Publisher publisher;

    public PublishingRunnable (Publisher pubApp) {
            this.publisher = pubApp;
    }

    @Override
    public void run() {
            Event ensE;
            try {
                    while(publisher.isConnected() && num_publication < PX_DATA) {
                            ensE = createEvent(min_random, max_random);

                            publisher.publish(ensE, num_publication);
                            num_publication++;


                            Thread.sleep(DELAY);
                    }
                    System.exit(0);
            } catch (Exception e) {
                    e.printStackTrace();
            }
    }
}
```

Listing 5 - PublishingRunnable class usage

3. Subscribe

Likewise the publisher client, to create a *Subscriber* client instance you can use the *RequestFactory* class passing the 'SUBSCRIBE' operation type.

```
RabbitMqClient client = RequestFactory.startGuestApplication(ApplicationPropertiesRepository.SUBSCRIBE);


Subscriber sub;
if(client.isConnected() && client instanceof Subscriber){
```

```
sub = (Subscriber) client;
sub.registerEventListener( new SubscriberListener() );
sub.subscribe();
}
```

Listing 6 - Event Subscribing example code

A Subscriber client uses a callback function registration through the *registerEventListener()* method. An Event Listener must implement the *EventListener* interface and the *onEvent()* method. The following is an example of *onEvent()* method that simply prints out the event with a toString function:

```
private static class SubscriberListener implements EventListener {
    private static final String TIME_FORMAT = "HH:mm:ss.SSS Z";
    private DateFormat dateFormatter = new SimpleDateFormat(TIME_FORMAT);
    private int i = 0;

    public void onEvent(Event event) {
        i++;
        StringBuilder msg = new StringBuilder();
        msg.append("-------------------------------------------------------\n");
        msg.append("Message #" + i + "\n");
        msg.append("PublisherID:: " + event.getApplicationID() + "\n");
        msg.append("Timestamp:: " + dateFormatter.format(event.getTimestamp()) +
"\n");
        msg.append("Namespace pattern:: " + event.getPattern() + "\n");
        msg.append("Persistent? " + event.isPersistent() + "\n");
        msg.append("Payload media type:: " + event.getContentType() + "\n");
        msg.append("Payload encoding:: " + event.getContentEncoding() + "\n");
        byte[] payload = event.getPayload();
        if (payload == null)
            msg.append("No payload");
        else if (payload.length == 0)
            msg.append("Empty payload");
        else if (event.getContentType().startsWith("text")) {
            try {
                String body = new String(payload,
```

```
event.getContentEncoding());
                                msg.append("Payload:: " + body + "\n");
                        } catch (UnsupportedEncodingException e) {
                                e.printStackTrace();
                        }
                } else
                        msg.append("Payload size:: " + payload.length + " bytes");

                Map<String,Object> headers = event.getHeaders();
                if (headers == null) {
                        msg.append("No headers");
                } else {
                        msg.append("Headers:");
                        for (String key: headers.keySet()) {
                                msg.append("\t\n");
                                msg.append(key);
                                msg.append(": ");
                                msg.append(headers.get(key));
                        }
                }
                msg.append("\n-------------------------------------------------------\n");

                System.out.println(msg.toString());

        }
    }
```

Listing 7 - Event Listener example code

As specified above, in the toolkit provided with the library there are two ready to use Java classes for test purpose.

Licensing

The BEinCPPS SeDEM released is licensed under the Apache License 2.0 (http://www.apache.org/licenses/), a permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Appendix 5 – Task Orchestration for CPPS (Activiti Extension)

BEinCPPS specific developments

Task Orchestration for CPPS is the integration of multiple components as described in the previous sections. Particularly, it has combined and extended multiple components for IoT devices integration, devices management and BPMs support.

Regarding devices management it can be referred to the Annex 7 where Homard is described in detail, for that reason we will focuses on this section to provide the details about how Activiti BPM engine has been extended.

Activiti BPM Engine is a web-based, integrated platform for the semantically-enhanced design, execution and monitoring of Business Processes in the scope of service-oriented Manufacturing Ecosystems. Targeted at the Virtual Factory domain, and based on open standards like Business Process Model and Notation (BPMN) 2.0, it is delivered as a web portal in order to maximize its collaborative nature.

As part of the foreground works and integration as part of the Task Orchestration for CPPS, Activiti BPM Engine has been extended by HOPU to also support IoT interactions (beyond the user's interactions and IT connectors). Thereby, Task Orchestration for CPPS can launch processes based on data-driven events as part of the interconnection with the field-level platform (HOMARD).

In the framework of the HOMARD platform, the integration of Activiti has enabled the creation of industrial processes based on conditions. On the one hand, the OMA LWM2M protocol provides the semantics and operations that make it possible to establish conditions and actions directly on the resources of the devices. On the other hand, Activiti BPM Engine provides us the ability to create processes from the values of these resources. An example would be the execution of an action when the temperature exceeds a threshold value. This action can trigger a reaction in an actuator or simply conclude with a notification. Task Orchestration for CPPS provides the possibility to manage the different states in which an industrial process can be found by returning an answer for each one of them.

The Task Orchestration for CPPS can currently be used through the HOMARD platform.

Task Orchestration for CPPS adds to the functionality that Activiti BPM Engine provides the ability to define industrial processes through a simple, friendly and well-

defined web interface, directly on IoT devices connected by the OMA LWM2M protocol. Specifically, the following characteristics are provided:

- Create human-based and service tasks, connected as a workflow.
- Bind service tasks to matching web service calls.
- Deploy and execute processes.
- Interact with human-based tasks through an integrated web UI.
- Automate the call of service tasks to web services.
- Monitor the execution of processes.

HW/SW Prerequisite

As mentioned in the previous section, the component is integrated with the Homard platform. Its requirements are therefore similar to those of the platform:

- A physical or virtual machine where you can deploy the service.
- Devices that support the OMA LwM2M communication protocol.
- A user account, that will be provided by HOP Ubiquitous, from which the devices can be managed.
- An Activiti service must be added to the software requirements.

Installation Instructions

The entire Homard platform along with the services that comprise it are deployed as a docker service. By that, the installation process is limited to starting the docker service.

To know more about Docker and how to use it please visit the link below:

- https://docs.docker.com/

User Manual

Any information related to the use of the component as well as its modification can be found in the project repository, whose link is provided below:

- https://github.com/HOP-Ubiquitous

Developers' Guide

Any information related to the use of the component as well as its modification can be found in the project repository, whose link is provided below:

- https://github.com/HOP-Ubiquitous

Examples

An account on the HOMARD platform can be provided by HOP Ubiquitous for anyone who want to test the component.

The HOMARD platform could be accessed through the following link:

- https://homard.hopu.eu

Licensing

The component is released under license Open Source (Apache License 2.0)

Appendix 6 – Device URL Manager Component

BEinCPPS specific developments

Device URL Manager provides an Open Source Core for users to be able to develop their own solutions which is accessible through an API REST, which is developed in python using Open Source frameworks such as Django and Django Rest Framework that provides to the developers an ecosystem that allows an easy extension of the system for introducing layers of security or registration of devices.

This component is developed around a core that is Open source and communicates with users through an interface that provides security and device management for users.

As can be seen in the figure of the architecture, all the new features that surround the core have been added through micro services that provide new functionalities of management of devices and layers of security to the system.

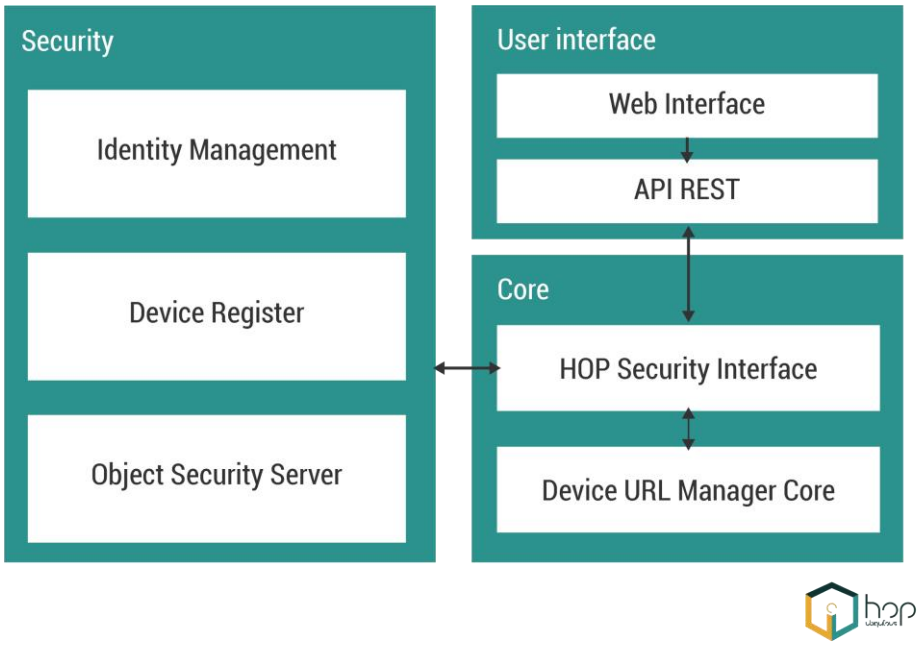

Figure 23 - Device URL Manager Architecture

Architecture Description

- Security

  This service provides the platform authentication and makes the connection between the user and his own devices for ensuring a correct device management.

  - Identity Management

    The identity manager allows the user to provide access to the various services that compose Homard through a single sign on, achieving fine-grained access control as well as facilitating the end-user interaction with the system. By using the OAuth protocol, the security can be ensured against possible attacks.

  - Register Tool

    This service is responsible for registering and deregistering the device within the platform. It also allows the devices to connect to the LwM2M server. This service contains device metadata that allow them to assign configuration and extra behaviour to them, such as predefined logic.

  - Object Security Server

    This service is responsible for registering all the proprietary information on the platform, serving as: a database for identity verification and permissions, a recovering service of user devices and, at the same time, as a backup in case of potential attack or loss of information.

- User Interfaces

  Users can communicate with the platform in two different ways: using the API rest for advanced users and by the website for users that need a seamless service.

  - API REST

    The platform contains the following two API REST:

    - Core API: It exposes all the Open Source resources but in our solution. It can be only access through the HOP Security Interface. It manages all the URLs and the devices information.

    - Users API: It exposes some extra methods enriched with security and device-user relationship. It allows developers to customize their own services.

- Web Interface

The web interface provides the user an easy way to manage their devices updating their URLs. This web interface is fully responsive and can be accessed through any device.

- Core

The core is composed by two services, one for doing the user-device relationship and managing the security, and other one, the Open Source URL manager, for doing the URL relationships and redirections.

- Hop Security Interface

This service manages the security between the security services and the Core. It will reject user messages that aren't authenticated or that try to access to unauthorized devices.

- Device URL Manager Core

This service is the Open Source Service on raw. It is developed and maintained by Hop Ubiquitous. It oversees the URL redirection, contains all the device-final URL mapping and store useful statistics for studying the impact of the web app in users.

HW/SW Prerequisite

The Device URL Manager is a software component delivered as a Docker image. The only requirement to make it work is a physical or virtual machine where the image could be started.

To be deployed in a real use case, a device with a Physical Web interface is required. The device must be managed in order to be able to set its announced URL. This URL will be the component address and a unique device identifier, such as its MAC address, that will be added to the URI. In this way, each time a user accesses the address of the Device URL Manager this will redirect him to the URL configured for his identifier.

Installation Instructions

As said before, this software component is released as a Docker image. Starting from this, the unique installation step is executing the docker run command.

To know more about Docker and how to use it please visit the link below:

- https://docs.docker.com/

User Manual

Any information related to the use of the component as well as its modification can be found in the project repository, whose link is provided below:

- https://github.com/HOP-Ubiquitous/DeviceUrlManager

Developers' Guide

Any information related to the use of the component as well as its modification can be found in the project repository, whose link is provided below:

- https://github.com/HOP-Ubiquitous/DeviceUrlManager

Examples

An example of the component already deployed can be found in the url:

- https://hpoi.info/

The service is open for anyone to perform tests.

Licensing

It is an open source solution enriched with proprietary services.

- Open Source: Device URL Manager Core. The component is released under Apache License 2.0.
- Proprietary: User Management and user interfaces.

Appendix 7 – Homard IoT Management for CPPS

BEinCPPS specific developments

The platform architecture is made up of well-defined modules. Figure 24 shows a schematic of the platform, which can be divided into 4 categories:

- Security, where you will find security related services.
- Core, which is in charge of managing OMA LWM2M devices.
- Services, where the services of HOPU that will enrich the functionality of the system are housed.
- User Interface, which will allow users to interact with services in a simple and intuitive way.

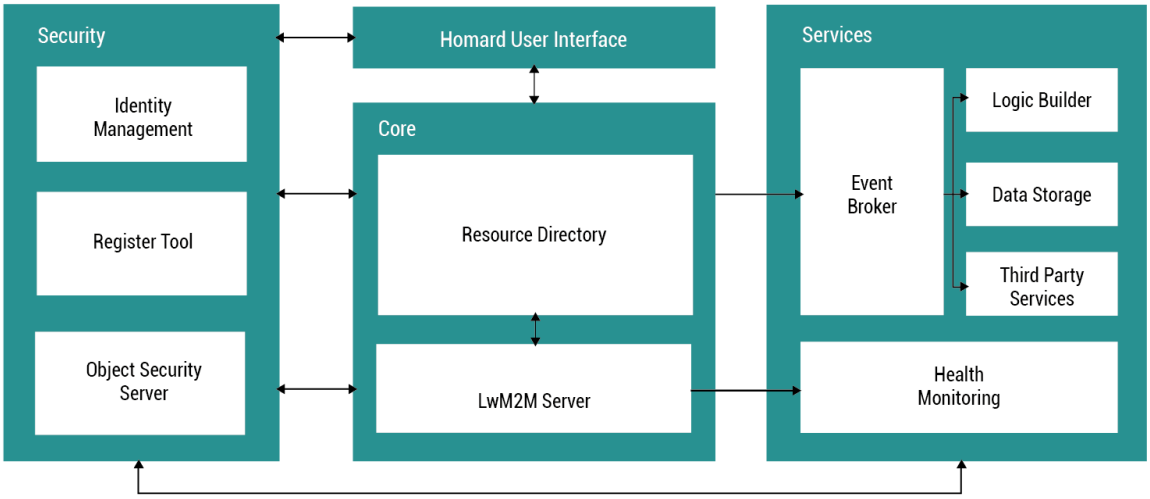

Figure 24 - Platform Architecture

The code of the platform has been restructured to give rise to independent modules as shown. More information of its main components can be found below in the document.

Services

The services provide extra functionality to the platform, such as connection monitoring, data storage or logical rule construction. The platform core communicates with the services through a broker (Event Broker) in order to relieve load to the OMA LWM2M server. The broker will be in charge of communicating to the active services through their respective interfaces. This architecture allows us to activate and

deactivate services in a simple and scalable way. These services are designed to be able to deploy individually without depending on each other.

Core

The Homard name stands for Hop Ubiquitous OMA resource directory. Its name is unequivocal since the main functionality is to provide the devices with a common server where they register their resources and capabilities. From this registry, other services can query the device, execute operations on it or subscribe to a change in the information recorded. Summing up, the protocol OMA LwM2M is used to give semantics and to provide a framework of information shared between the different institutions around the world.

The following table illustrates some of the most important and interesting OMA LwM2M Objects supported by the core.

Provider	Object name	Object ID
OMA	Security	0
OMA	Server	1
OMA	Device	3
OMA	Connectivity Monitoring	4
OMA	Firmware	5
OMA	Location	6
OMA	Connectivity Statistics	7
OMA	WLAN connectivity	12
IPSO	Digital Input	3200
IPSO	Digital Output	3201
IPSO	Analogue Input	3202

IPSO	Analogue Output	3203
IPSO	Illuminance Sensor	3301
IPSO	Temperature Sensor	3303
IPSO	Humidity Sensor	3304
IPSO	Power Measurement	3305
IPSO	Accelerometer	3313
IPSO	Barometer	3315
IPSO	Loudness	3324
IPSO	Concentration	3325
IPSO	Distance	3330
IPSO	Gyrometer	3334

User interface

The user interface allows users to interact with services in an easy and intuitive way. The goal is to provide the end user with the ability to manage and monitor their devices, implement the logic of all their devices and analyze the information they collect. In Figure 25 can be seen the platform access window.

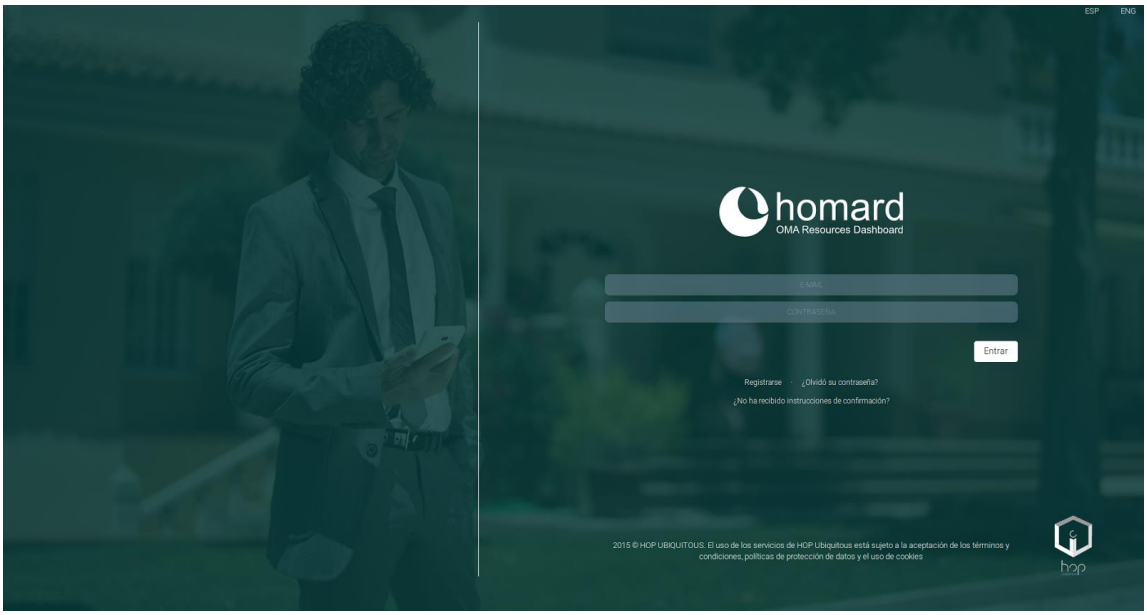

Figure 25 – Homard Login View

After accessing, the dashboard appears as shown in Figure 26. Figure 27 corresponds to the Device Management view implemented in the core through which operations can be performed on the instances and resources of the devices. The structure of the information imposes the protocol LwM2m.

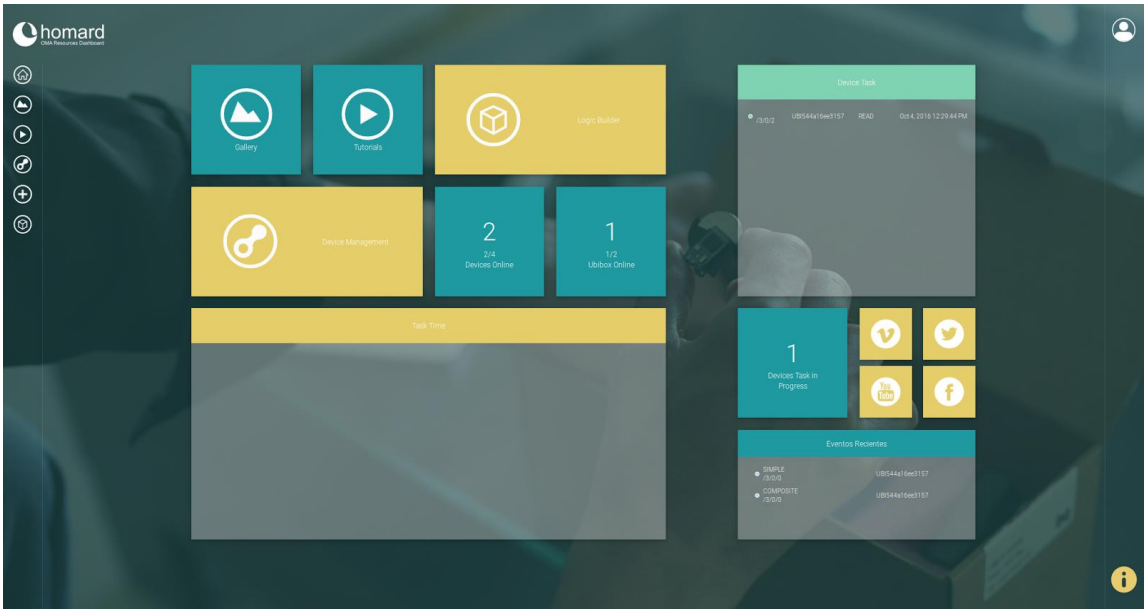

Figure 26 - Homard Dashboard

Finally, Figure 28 shows the result of a query to the Health Monitoring module.

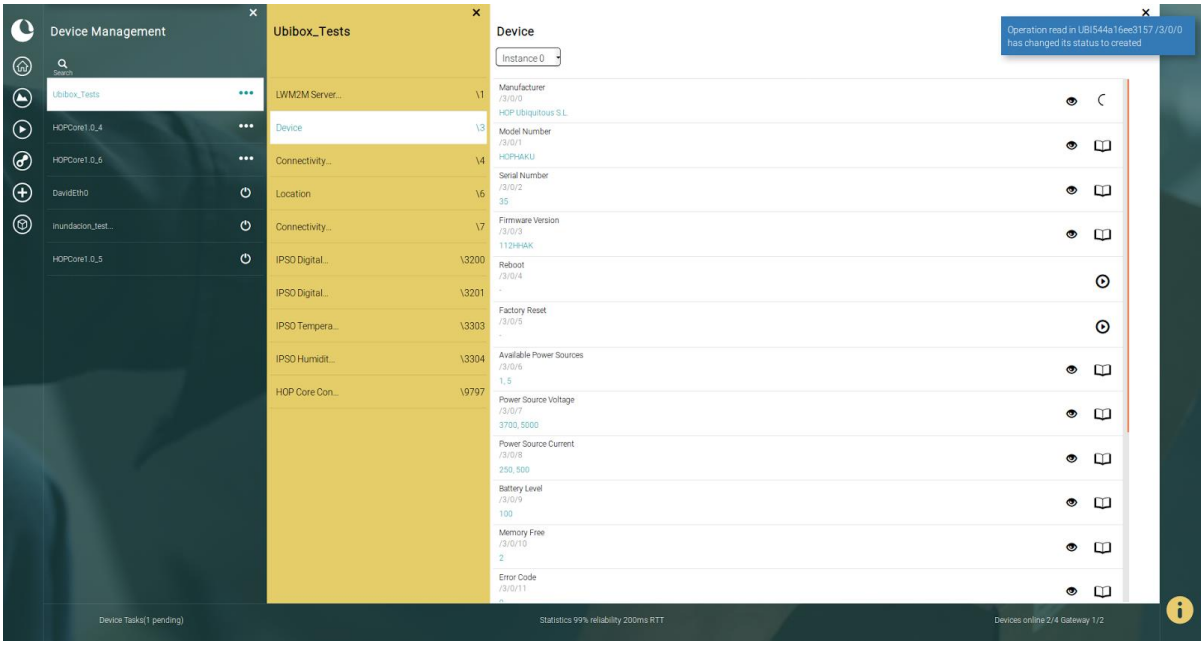

Figure 27 - Device Management View

The user interface also allows to follow the sending and receiving packets between the device and the LwM2M server.

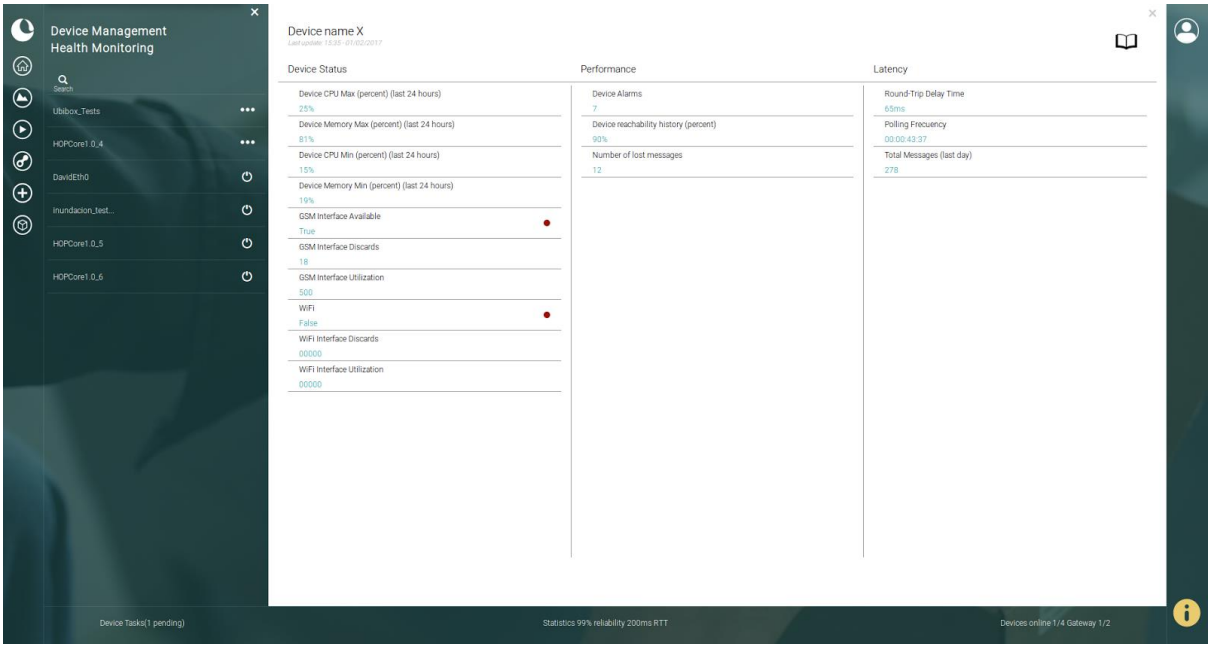

Figure 28 - Device and Health Network Monitoring View

Use cases: O2 – CPPS4ZDM (CENTIC-JALSOSA-IPYC)

CPPS4ZDM project presented on the 2nd Open Call is currently using the Homard component, which brings to the device a secured initialization and an environment where to manage, follow its health, carry out advanced settings such as add new WiFi access point configurations where the device will connect, etc...

An extended explanation about the use of Homard in the CPPS4ZDM can be found on the correspondent deliverables of the Open Call 2.

HW/SW Prerequisite

Homard is a software component delivered as a Docker service. Its software requirements are:

- A physical or virtual machine where you can deploy the service.
- Devices that support the OMA LWM2M communication protocol.
- A user account, that will be provided by HOP Ubiquitous, from which the devices can be managed.

Installation Instructions

The entire Homard platform along with the services that comprise it are deployed as a docker service. So the installation process is limited to starting the docker service.

To know more about Docker and how to use it please visit the link below:

- https://docs.docker.com/

User Manual

Any information related to the use of the component as well as its modification can be found in the project repository, whose link is provided below:

- https://homard.hopu.eu/indexPage/wiki.html

Developers' Guide

Any information related to the use of the component as well as its modification can be found in the project repository, whose link is provided below:

- https://homard.hopu.eu/indexPage/wiki.html

Examples

An account on the Homard platform can be provided by HOP Ubiquitous for anyone who want to test the component.

The Homard platform could be accessed through the following link:

- https://homard.hopu.eu

Licensing

The Engine and key components such as data storage and BPM integration are released under Eclipse Distribution License 1.0 (BSD). Other modules as the data visualization and advanced services are proprietaries.