




**MIDIH**  
**MANUFACTURING · INDUSTRY**  
**DIGITAL · INNOVATION · HUBS**



Grant Agreement No. 767498  
Innovation Action Project  
H2020-FOF-12-2017

## D4.4

# MIDIH Open CPS/IOT Components v2

WP4 - Open Platform architecture, development, integration  
and testing

**Version:** 1.5

**Due Date:** 31/12/2019

**Delivery Date:** 15/01/2020

**Type:** OTHER

**Dissemination Level:** PU

**Lead partner:** ATOS

**Authors:** All Partners (See List of Contributors below)

**Internal reviewers:** Gabriele De Luca (ENG)

## Disclaimer

This document contains material, which is the copyright of certain MIDIH consortium parties, and may not be reproduced or copied without permission.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the MIDIH consortium as a whole, nor a certain part of the MIDIH consortium, warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.



MIDIH has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 767498.

**Version Control:**

Version	Date	Author	Author's Organization	Changes
0.1	25/10/2019	Jesús Benedicto	ATOS	First version of the TOC
0.2	20/12/2019	Jesús Benedicto	ATOS	TOC refinement and Contributions to Chapters 2 and 3.
1.0	26/12/2019	Jesús Benedicto	ATOS	Factsheets Added
1.2	28/12/2019	Jesús Benedicto	ATOS	Executive summary and Introduction Added
1.3	08/01/2020	Jesús Benedicto	ATOS	Chapter 4 Added
1.4	13/01/2020	Gabriele De Luca	ENG	Internal Review
1.5	15/01/2020	Jesús Benedicto	ATOS	Peer review comments addressed. Final editing

**Contributors:**

Contributor	Partner
Jesús Benedicto Cirujeda	ATOS
Jonny Johansson	LTU
Jukka Koskinen	VTT
Mateusz Chodyła	PSNC
Antonio Jara	HOPU
Oliver Keil	FOKUS
Gabriele De Luca	ENG
Javier Hitado	ATOS
Nenad Stojanovic	NISSATECH

**Deliverable Title: MIDIH Open CPS/IOT Components v2**

**Deliverable Number D 4.4**

**Keywords: Smart Factory, Automation, Analytics, Data-at-Rest, Data-in-Motion, Open Digital Platforms, Digitalization, Industrial Internet of Things, Architecture, AI, FIWARE4Industry, Siemens MindSphere, OPC UA, FogFlow**

## Executive summary

---

The D4.4 “MIDIH Open CPS/IOT Components v2” is a public document delivered in the context of WP4 and is accompanying the final prototypes of the MIDIH components generated in the context of MIDIH tasks WP4.2, WP4.3 and WP4.4 in its second iteration.

- **T4.2 Edge-oriented Local Clouds for Factory Automation**, responsible for providing the MIDIH Open Source reference architecture with a distributed cloud architecture. For this purpose, the improvement of the initial Edge Computing Node as part of FogFlow project based on FIWARE integrating a subset of the Data in Motion reference implementation, has been the objective, as well as the integration of the Arrowhead framework in the MIDIH final architecture, developing a specific translator able to communicate Arrowhead with FIWARE.
- **T4.3 Brownfield Integration via Open APIs**, responsible to address brownfield integration and interoperability with proprietary solutions and standards. In this second iteration, the main objective has been the improvement of the previous implemented gateways between MIDIH platform and proprietary but open commercial solutions in the field of IoT like Siemens MindSphere. During this second iteration, the improvement of the interoperability between the two protocols NGSI (FIWARE) and OPC UA (SIEMENS MindSphere IoT platform) has been the main objective of the development. In the same way, specific components are enhanced to provide greater flexibility when connecting to other applications that allow better visualization of gathered data to provide a useful dashboard as part of a Decision Support Solution, such as MASAI with the Manufacturing Data platform.
- **T4.4 Industrial IoT and Analytics Platform**, responsible for developing the MIDIH platform for IoT and Analytics, having into account both Data-in-Motion (DiM) and the Data-at-Rest (DaR). Based on the first prototype delivered in the first iteration, the objective in this new iteration has been to provide an improved AI-enabled data analytics platform based on APACHE and FIWARE open source components, providing an interoperability model between the two different technologies.

The output of these tasks delivers a collection of functional components for implementing the final MIDIH reference architecture, described in D4.2, for the Cross-Border Industrial Experiments in WP5 (and Open Calls winners). Like in the first iteration, the developed open source components enhance, extend or integrate existing Open Source solutions available on the market or developed within previous European research projects such as BEinCPPS (Phase II I4MS).

# Table of Contents

---

- Executive summary ..... 5
- Table of Contents ..... 6
- List of Figures ..... 7
- 1 Introduction ..... 9
- 2 Edge-oriented Local Clouds for Factory Automation (T4.2) ..... 10
  - 2.1 The Arrowhead – FIWARE translator ..... 10
  - 2.2 Usage of Arrowhead Framework in a mocked factory scenario – management of production orders ..... 11
  - 2.3 An IoT platform using 5G communication for robotics applications ..... 12
  - 2.4 FogFlow Integration ..... 12
  - 2.5 Integration of microCEP on Edge ..... 15
- 3 Brownfield Integration via Open APIs (T4.3) ..... 15
  - 3.1 OPC UA Agent ..... 15
  - 3.2 FI-MIND Bridge ..... 16
    - 3.2.1 Background Components ..... 16
    - 3.2.2 Foreground Components ..... 17
  - 3.3 MIDIH MindSphere Custom Agent and MASAI MDP Connector for MIDIH ..... 17
  - 3.4 FIWARE ROS Integration via OpenMTC generic enabler ..... 18
- 4 Industrial IoT and Analytics Platform (T4.4) ..... 18
- 5 Conclusions and Future Outlook ..... 23
- Annex A. Factsheet for Arrowhead Translator with FIWARE plugin ..... 24
- Annex B. Factsheet for Arrowhead Framework demonstrator ..... 28
- Annex C. Factsheet for Industrial IoT platform for robotics application ..... 31
- Annex D. Factsheet for Edge-oriented Local Clouds for Factory Automation – FogFlow ..... 34
- Annex E. Factsheet for FIWARE OPC UA Agent ..... 41
- Annex F. Factsheet for FI-MIND ..... 46
- Annex G. Factsheet for MIDIH MindSphere Custom Agent and MASAI MDP Connector for MIDIH ..... 50
- Annex H. Factsheet for FIWARE ROS integration via OpenMTC generic enabler ..... 56

# List of Figures

Figure 1 – MIDIH Reference Architecture .....	9
Figure 2 – Arrowhead Framework Factory Scenario .....	11
Figure 3 – Architecture of the edge-computing platform .....	12
Figure 4 – Entity differences .....	13
Figure 5 – Adapter Scheme .....	13
Figure 6 – Entity stored in IoT Broker .....	14
Figure 7 – IoT Worker output stream .....	14
Figure 8 – FogFlow and Orion communication .....	14
Figure 9 – FI-MIND Interoperability Levels .....	16
Figure 10 – FIWARE and MindSphere integration .....	16
Figure 11 – FIWARE and MindSphere Data Models .....	17
Figure 12 – MASAI Interoperability.....	18
Figure 13 – MIDIH Industrial IoT and Analytics Platform.....	19
Figure 14 – D2Twin updated processing pipeline .....	19
Figure 15 – Example of control chart.....	20
Figure 16 – An example of MEWMA control chart .....	21
Figure 17 – Influence of correlation.....	22
Figure 18 – MEWMA in action .....	22
Figure 19 – Translator FIWARE plugin functional schema .....	24
Figure 20 – FIWARE plugin description .....	25
Figure 21 – Arrowhead Framework Robotic Arm Demonstrator – Architecture diagram .....	29
Figure 22 – Screenshot of the consumer Web GUI .....	30
Figure 23 – MIDIH Industrial IoT and Analytics Platform Architecture (FIWARE Pipeline) .....	31
Figure 24 – Architecture of the edge-computing platform .....	32
Figure 25 – Data Flow Processing .....	35
Figure 26 – Deployment view .....	36
Figure 27 – FogFlow welcome page.....	38
Figure 28 – OPC UA Agent Architecture .....	41
Figure 29 – Manufacturing Data Platform login .....	53
Figure 30 – Manufacturing Data Platform Monitoring.....	53
Figure 31 – Manufacturing Data Platform alarms visualization .....	55
Figure 32 – Manufacturing Data Platform Ping Monitoring .....	55
Figure 33 – Separation of Domains in OpenMTC.....	57

Figure 34 – OpenMTC FIWARE integration .....59

Figure 35 – ROS IPE's interface to Sawyer Robot Simulator .....59

Figure 36 – Overall view on all used components .....60



# 1 Introduction

This second prototype is the result of a second iteration of development activities for edge-oriented factory automation, brownfield interoperability and Industrial Analytics, three tasks of WP4 (WP4.2, WP4.3 and WP4.4 respectively) responsible to develop open source components to implement several functionalities of MIDIH Functional and Modular Reference Architecture in support to all the cross-border experiments in the CPS/IOT domain. This prototype complements and extends the one presented in the “D4.3 MIDIH Open CPS/IOT Components v1”.

As detailed in the DoA, main objectives are:

- To integrate in the Reference Architecture, open source components aiming at developing local cloud edge-computing nodes for distributed Factory Automation.
- To integrate open APIs enabling the MIDIH Architecture to interoperate with existing legacy systems, in particular those implementing IOT and Analytics functionality.
- To integrate in the MIDIH Architecture additional Open Source components to finally implement an original MIDIH solution for Industrial CPS/IOT Data Analytics.

The final MIDIH architecture reflected in Figure 1, which is detailed in the deliverable “D4.2. Functional and Modular Architecture of MIDIH CPS/IOT System v2” shows which functional components of the architecture are covered by the different tasks in this final prototype, and where the developments have been focused.

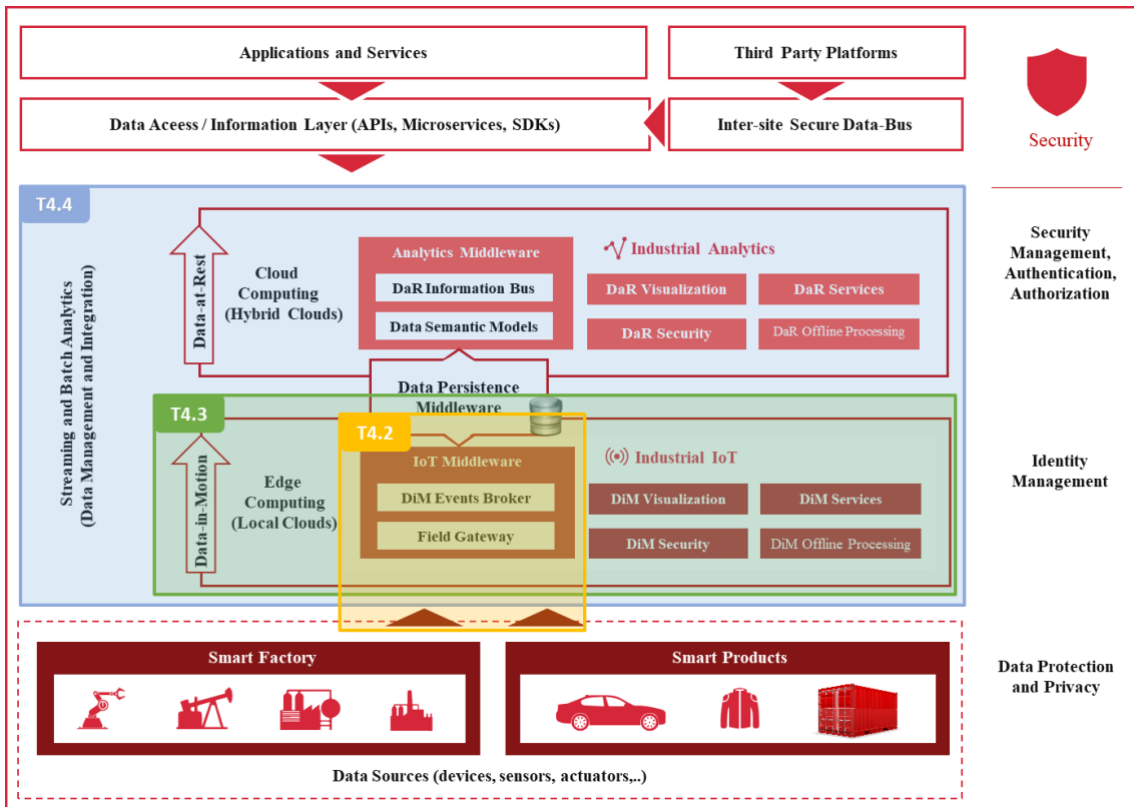


Figure 1 – MIDIH Reference Architecture

The document follows the same structure as “D4.3 MIDIH Open CPS/IOT Components v1”, where one chapter is devoted to each of the tasks and the development made within them, including the same organization:

- general information of the Composite Outcome(s)
- a short description of each Component that made up the Composite Outcome(s)
- the background and foreground for these components

In addition, for each specific component that made up the Composite Outcome(s), a factsheet is provided in the Annexes following this schema:

- general information of the component
- functional description and architecture
- manuals both technicians (installation, configuration) and end-users (usage)

## 2 Edge-oriented Local Clouds for Factory Automation (T4.2)

---

Main objectives of this task in the second iteration have been to improve the previous FogFlow developments that offer Edge Computing capabilities into the MIDIH architecture, providing new features such as local data processing, reducing the number of messages between the IoT Device and the cloud, since the data provided by sensors can be pre-processed, aggregated, privatized,... resulting in network decongestion, increasing data processing speed and security.

Similarly, new components and their usage in mocked factory scenarios have also been developed:

- The Arrowhead – FIWARE translator, which allows the Arrowhead System to look for services using NGSiv2 and FIWARE.
- The usage of Arrowhead Framework in a mocked factory scenario in order to automatize the management of production orders. This scenario uses the Edge Computing paradigm for building a robotic arm.
- The implementation of an IoT platform based on MIDIH using 5G communication for a robotic application.
- The implementation of a microCEP (Complex Event Processing) based on FogFlow or FIWARE, in order to locally process the data streams allowing act in consequence with the input stream.

### 2.1 The Arrowhead – FIWARE translator

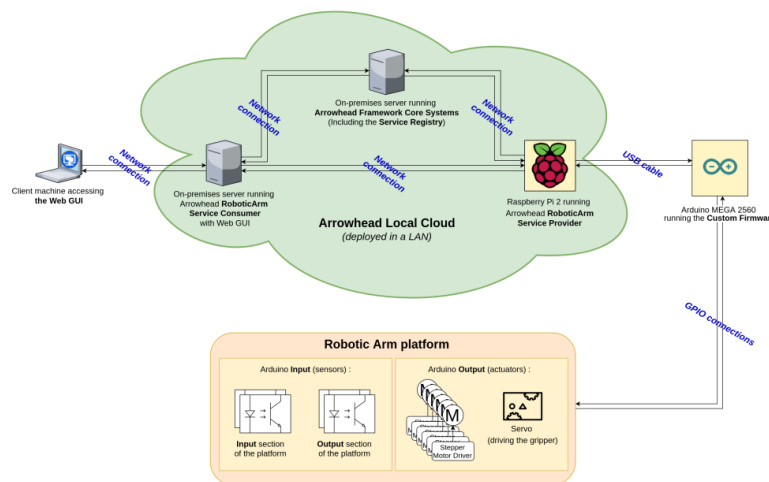
One key to expand the possibility of Edge computing and to increase the possibilities for interaction and data processing is to create interoperability between Edge computing entities. To achieve this, T4.2 has developed support in the Arrowhead Framework for the FIWARE framework and its platforms through the use of the Arrowhead Translator system. The aim is to enable a Systems of Systems approach where Arrowhead Systems can search for services using NGSiv2 and FIWARE. NGSiv2 provides a RESTful interface for querying context, subscriptions, registrations, and notifications of context information. This translation support is developed

specifically in the context of the MIDIH project to enable the further implementation of FIWARE – Arrowhead compliant systems.

To achieve the desired functionality, a plugin for the Arrowhead Translator System has been implemented. The plugin can communicate with the Orion Context broker in FIWARE, register, search for services, and provide data translation between the SenML format used in Arrowhead and the JSON format in NGSiv2. The plugin can also request from the Orion Context broker the information about all registered services and register them into Arrowhead, and later on enable a specific interface to consume each service in SenML (like any other Arrowhead System). The Translator is presented further in the attached factsheet in Annex A.

## 2.2 Usage of Arrowhead Framework in a mocked factory scenario – management of production orders

The Arrowhead Framework was designed to solve industrial automation cases. Therefore, a testing environment should contain specific attributes of, for instance, a modern factory.



**Figure 2 – Arrowhead Framework Factory Scenario**

Based on the following assumptions PSNC designed and prepared a testing environment, which included:

- manufacturing a physical layer,
- using the Edge Computing paradigm,
- writing custom code on each layer,
- using open source projects wherever applicable.

A detailed description of building the robotic arm, architecture design, code and its functionalities is available in Annex B.

### 2.3 An IoT platform using 5G communication for robotics applications

Within MIDIH was also implemented an IoT platform using 5G communication for a robotic application based on the MIDIH platform. The IoT computing platform is based on MIDIH Industrial IoT and Analytics Platform Architecture (FIWARE4Industry Pipeline), wherein a 5G network was used for data communication.

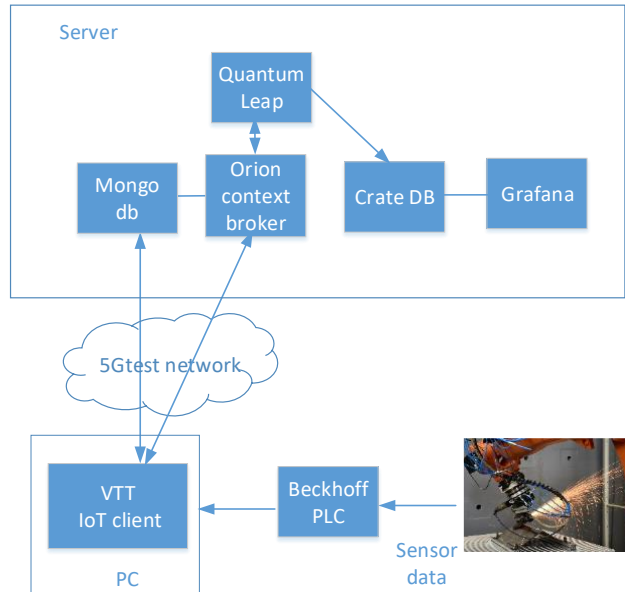


Figure 3 – Architecture of the edge-computing platform

The functionalities provided by the architecture model are mapped into software components to achieve robot control as well as feedback of robot sensor data. The platform is described in the attached factsheet in Annex C.

### 2.4 FogFlow Integration

In the previous iteration we focused on integrating the FogFlow Framework into the FIWARE reference architecture, this integration consists in allowing communication with the IoT devices that are already FIWARE compatible, to be FogFlow compatible as well.

There are several kinds of IoT devices on the market that may have different protocols for communicating (MQTT, LwM2M, UL, OPC UA, ...). For the integration of the different protocols that these devices present in the FIWARE architecture, there are a set of components called IoT Agents, which allow the grouping of devices and translate the native protocol (MQTT, LwM2M, UL, OPC UA, ...) of the device to the NGSI protocol (for communicating with the FIWARE NGSI RESTful interface).

These IoTAgents can be also used to communicate the devices directly with the FogFlow Edge Computing Framework (FIWARE Generic Enabler for Edge), so that their data can be sent to the edge and processed instead to directly to the cloud. But it presents a problem, the NGSI version used by the IoTAgents differs from that of FogFlow, because FogFlow was developed on an outdated NGSI.

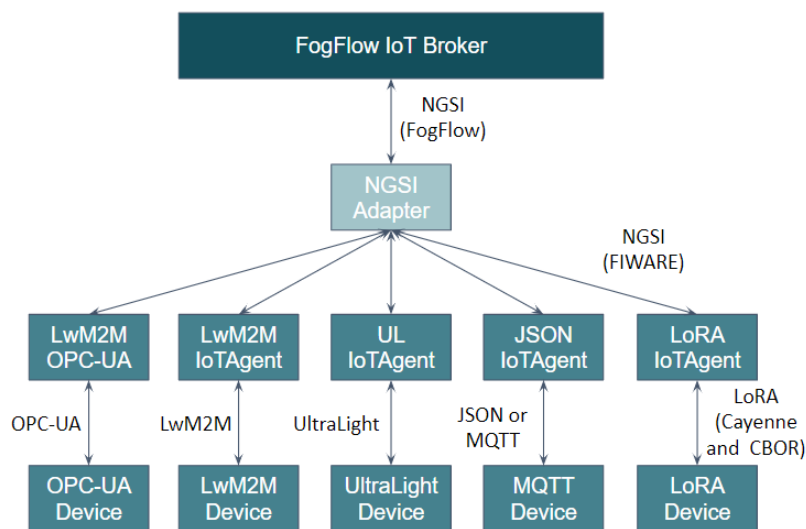
As shown in the following figure, the formats of the NGSI messages differ so a solution is needed to solve this problem

FIWARE IoTAgent NGSI example	FogFlow NGSI example
<pre> {   "contextElements": [     {       "id": "Room1",       "type": "Room",       "isPattern": "false",       "attributes": [         {           "name": "temperature",           "type": "float",           "value": "23.0"         }       ]     }   ],   "updateAction": "APPEND" } </pre>	<pre> {   "contextElements": [     {       "entityId": {         "id": "Room1",         "type": "Room",         "isPattern": false       },       "attributes": [         {           "name": "temperature",           "type": "float",           "contextValue": 23.0         }       ]     }   ],   "updateAction": "UPDATE" } </pre>

**Figure 4 – Entity differences**

To achieve the integration of the devices into the FogFlow using the IoTAgents until the update of the protocol on FogFlow side (which involves huge changes on FogFlow), we proposed a solution based on a service that adapts the small differences between these messages, this added functionality allows to transparently communicate the devices with the services deployed on the Edge, providing advantages on the data management such as low latency, reduced bandwidth, privacy and allowing high scalability and reduced operating costs.

This solution also allows to complete the FIWARE architecture with Edge Computing capabilities at the same time that allow to integrate several device communication protocols. The following picture illustrates where this adapter is placed into the FIWARE architecture.



**Figure 5 – Adapter Scheme**

This service performs the translation of NGSiv1 messages to register and update the entities of IoT devices. As the following image shows, the entities are translated and sent successfully to FogFlow's IoT Broker which allows these entities to be used in the IoT Workers to perform tasks with the data.

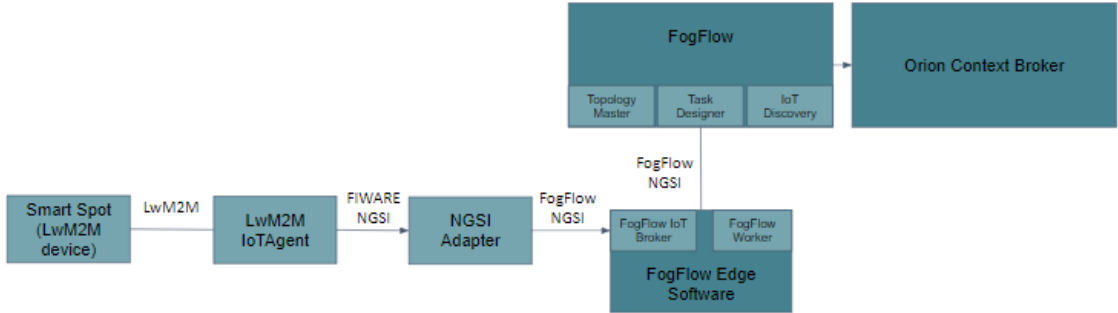
```

1  {
2    "entityId": {
3      "id": "HQP240ac4066f62:smartspot",
4      "type": "smartspot"
5    },
6    "attributes": [
7      {
8        "name": "temperature",
9        "type": "Number",
10       "contextValue": "32.2999992"
11      },
12     {
13       "name": "humidity",
14       "type": "Number",
15       "contextValue": "19.2021484"
16     }
17   ]
18 }

```

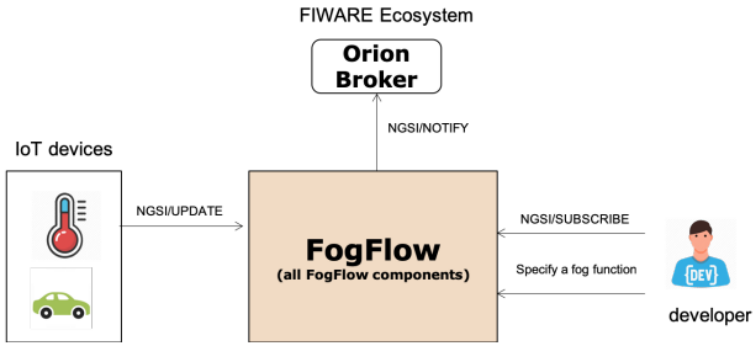
**Figure 6 – Entity stored in IoT Broker**

Thanks to the use of this service we have been able to integrate IoT devices within the FogFlow Framework to deploy tasks in the edge nodes that use the data provided by the devices as shown in the following image.



**Figure 7 – IoT Worker output stream**

FogFlow can be used to dynamically trigger data processing functions between IoT devices and Orion Context Broker, for the purpose of transforming and preprocessing raw data at edge nodes to finally send the processed data to Orion Context Broker as shown in the figure below.



**Figure 8 – FogFlow and Orion communication**

The factsheet of FogFlow integration is in Annex D.

## 2.5 Integration of microCEP on Edge

In the second iteration as described in previous deliverables, the integration of the micro Complex Event Processing (CEP) being developed by Nissatech is under study. The inclusion of the CEP at the Edge level allows to empower the Edge applications with detection of complex events without the need of implement each event behavior detection on the directly on the Edge application. The microCEP provides the capability to perform actions on the industry environment without waiting for cloud processing, providing a faster response to the event since it is processed locally. This feature is currently in work in progress status.

## 3 Brownfield Integration via Open APIs (T4.3)

---

Main objective of this task in this second iteration has been to improve the previous developments focused on achieving brownfield integration. In some cases, adapting the previous components to cover new specific requirements coming for the industrial experiments, or developing a new one in order to improve the overall functionality of the MIDIH architecture, providing for that new components dealing with interoperability purposes. Synergies with WP5 activities allowed to fit the second iteration requirements for MIDIH CPS/IoT components to the real MIDIH Industrial Cases' needs.

Therefore, a more structured and stable implementation approach to interconnect FIWARE with the Siemens MindSphere commercial OPC UA based platform has been developed with the objective of improving interoperability between the NGSI and OPC UA protocols.

The improvements implemented are based on several developments described in the following subchapters.

### 3.1 OPC UA Agent

The OPC UA Agent was improved during the second development iteration. The new version includes many improvements designed during the validation at POLIMI premises. Main changes can be summarized as follows:

- Dynamic configuration, for fully supporting the IDAS APIs. Configuration and binding between OPC UA nodes and NGSI attributes can be specified dynamically using IDAS services.
- OPC UA session parameters tuning, in order to support OPC UA data processing with a period of 50 ms.
- Mapping Tool improvement, to simplify the static configuration process mostly when the OPC UA server is quite complex.

At the end of the second iteration period, the official documentation was improved, and the end of the incubation period was proposed to the FIWARE community in order to make the OPC UA agent an official FIWARE IoT Agent. The OPC UA agent is described in the attached factsheet in Annex E.

### 3.2 FI-MIND Bridge

The FI-MIND Bridge main objective is to integrate the FIWARE lane of the MIDIH Reference Architecture with the Siemens MindSphere IoT platform via three levels of interoperability:

- **Sensors**, connecting real-world things (e.g. Smart Industry Objects) with FIWARE Orion Context Broker.
- **Platform**, defining a common data model to describe Smart Objects (Data-in-Motion), information from Enterprise systems (Data-at-Rest) and contextual data (Situational Data).
- **Application**, using application from Siemens for data visualization creating custom rules to create high-level information.

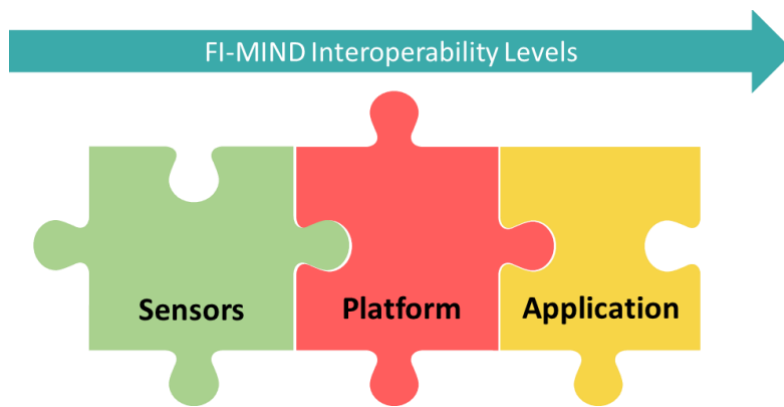


Figure 9 – FI-MIND Interoperability Levels

Supporting standard data models allows reaching a high level of interoperability making the FI-MIND a ready-to-use component for integrating FIWARE with MindSphere.

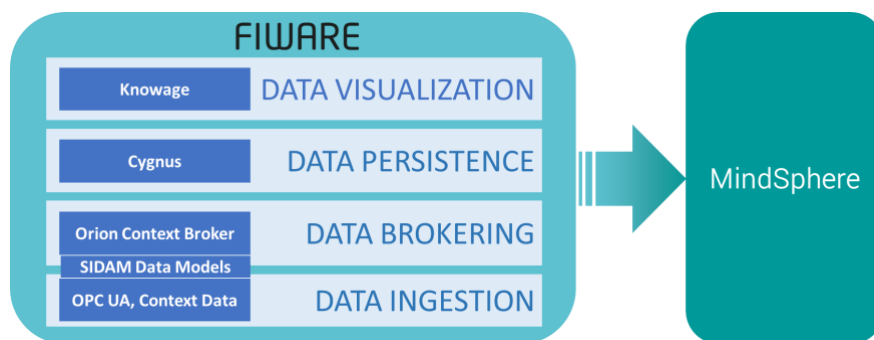


Figure 10 – FIWARE and MindSphere integration

#### 3.2.1 Background Components

The MindSphere SDK for Java enables Java developers to work with the MindSphere IoT platform easily. It provides a Java client implementation for the RESTful MindSphere Platform APIs.

The FI-MIND includes the MindSphere SDK for Java v1 in order to:

- Authenticate the client (MindSphere credentials).



- Create the Digital Twin on the MindSphere cloud, registering new assets (devices) and aspects (variables).
- Send timeseries data for registering value updates coming from the Orion Context Broker.

### 3.2.2 Foreground Components

The MIDIH FI-MIND bridge is aligned with the Smart Industry Data Model (SIDAM) enabling the integration of the FIWARE cloud with the MindSphere ecosystem. The bridge converts NGSI information described in supported data models in the MindSphere Data Model using MindSphere concepts.

The FI-MIND gathers data exposing APIs for:

- Receiving NGSI data described with supported data models from a generic external component.
- Processing Orion Context Broker notifications for data collecting.

Further details are described in the attached factsheet in Annex F.

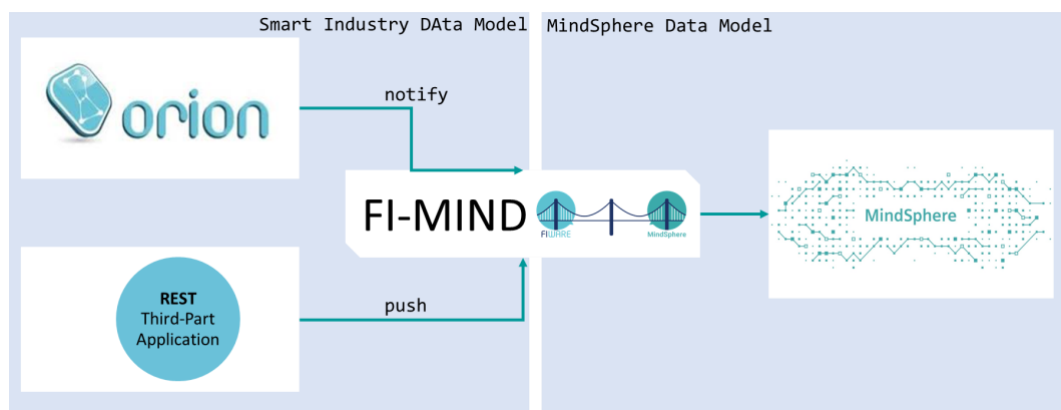
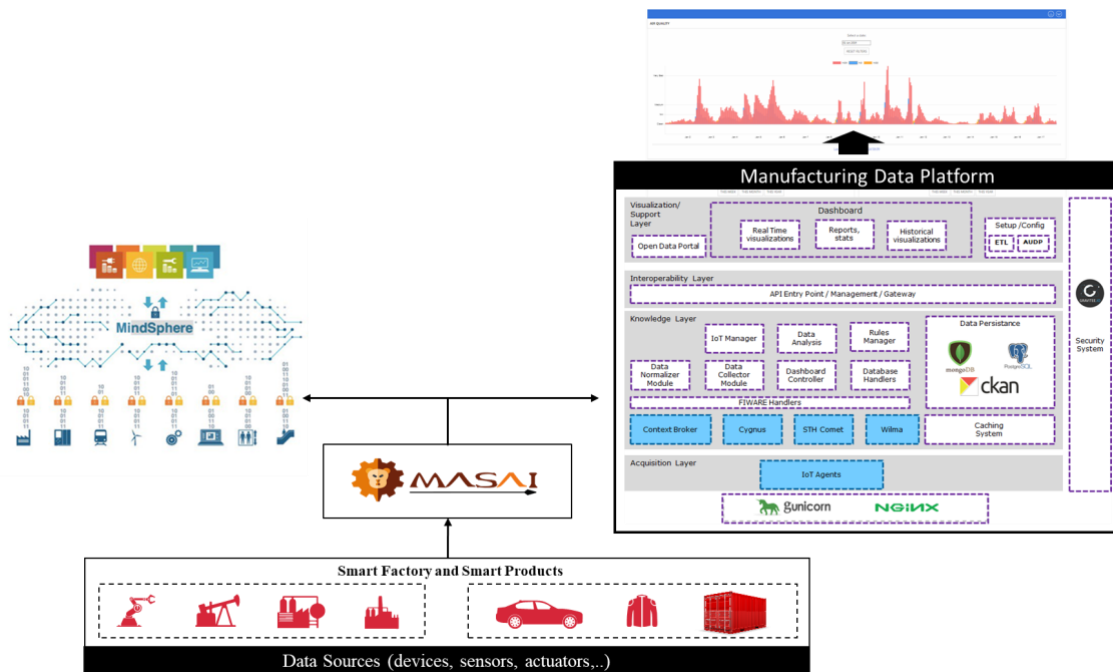


Figure 11 – FIWARE and MindSphere Data Models

### 3.3 MIDIH MindSphere Custom Agent and MASAI MDP Connector for MIDIH

The MIDIH MindSphere Custom Agent main objective is to provide an open source end-to-end solution for data capture in industrial environments covering multiple protocols like MQTT, OPC UA, etc., and their subsequent processing, storage and visualization, while providing interoperability mechanisms with other third-party tools and commercial solutions, especially with Siemens MindSphere.

The MIDIH MindSphere Custom Agent is based on FIWARE and intended to enable capturing data from IoT and OPC UA devices on the shop floor, allowing the integration of this data inside different solutions like Siemens MindSphere IoT platform or the Cross-Domain Data Platform, solution “powered by FIWARE” developed by ATOS.



**Figure 12 – MASAI Interoperability**

The factsheet for this component, where its functionality, its architecture and technical information such as the manuals for its installation and usage, are detailed is in “Annex C” of the “D4.3 MIDIH Open CPS/IOT Components v1” and it remains as described there. However, “Annex G” of this document describes the new functionalities added in the second iteration, how they have been implemented and how to use them.

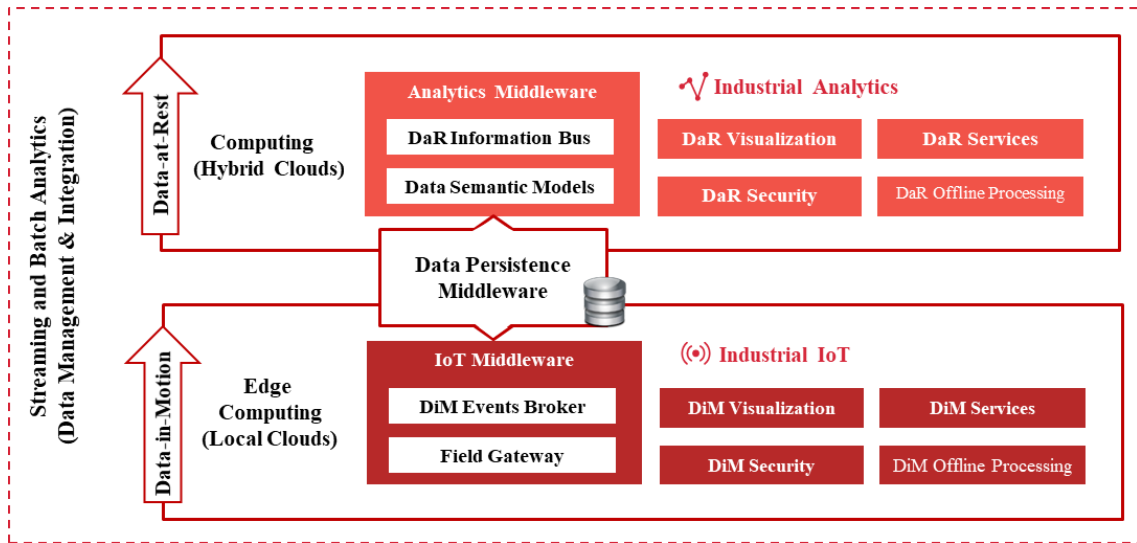
### 3.4 FIWARE ROS Integration via OpenMTC generic enabler

The integration of the M2M/IoT platform OpenMTC and the Robot Operating System (ROS) is carried out during the second iteration of T4.3. Main objective has been to integrate two complex platform, the OpenMTC and ROS in order to exchange information based on their data communication interfaces. This development helps in the inclusion of robots in the scope of MIDIH.

Complete information about the functionality, deployment and usage is reflected in “Annex H. Factsheet for FIWARE ROS integration via OpenMTC generic enabler”

## 4 Industrial IoT and Analytics Platform (T4.4)

MIDIH T4.4 responsible for the Data Analytics Platform (Figure 13), in the first iteration developed two basic lanes for Industrial IoT and Industrial Analytics, one FIWARE-based and the other based on worldwide known Open Source projects and Foundations, such as APACHE.



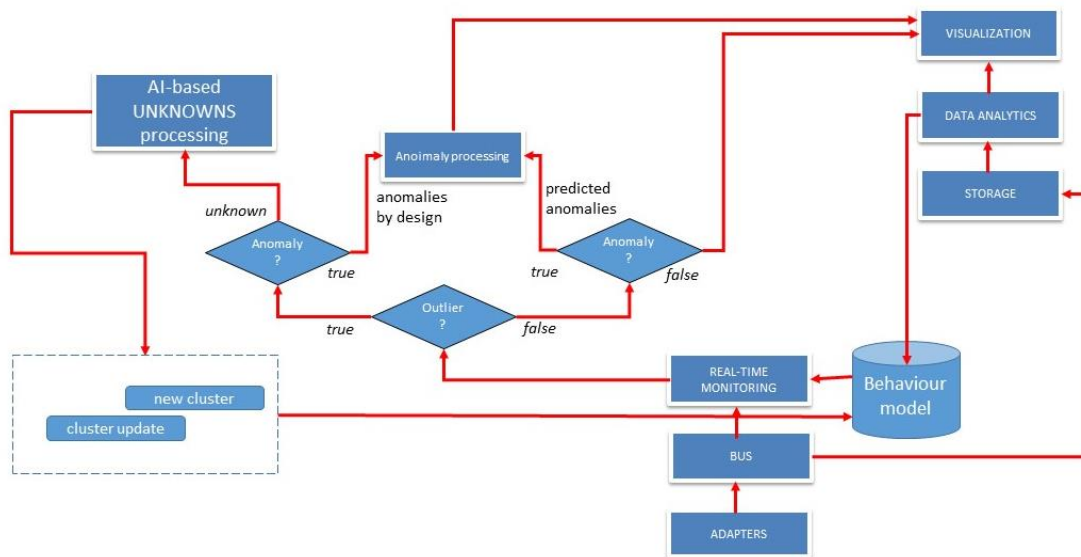
**Figure 13 – MIDIH Industrial IoT and Analytics Platform**

In this last prototype, T4.4 has evolve the Data Analytics Platform to support, in an efficient way, process monitoring and control in the situations which show unusual behavior of the system/process. This is especially important for the processes where dynamic changes in the internal or external context are “usual”. For example, it is possible that the raw material varies in the quality causing problems in the setup of the process parameters. In that case, the process monitoring system should be able to detect such a situation as early as possible and to resolve it in an efficient way. Therefore, there are two open issues in such situations:

- 1) How to detect that a situation is unusual (outlier)
- 2) How to react properly

In the rest of the text we describe how D2Twin approach resolves the first issue (part of the monitoring process). The second issue (resolution) is part of the control process, which is out of scope of the data analytics platform.

In the following Figure 14 we present the update of the processing pipeline.

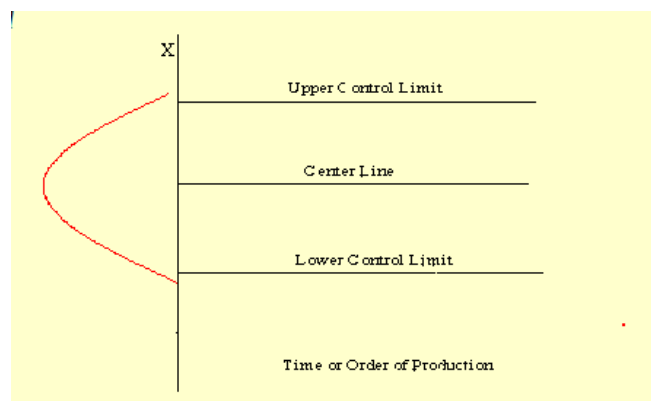


**Figure 14 – D2Twin updated processing pipeline**

The implementation of the anomaly detection is related to the usage of multivariate control charts, in particular the **Multivariate exponentially weighted moving average – MEWMA<sup>1</sup>** was adopted.

Multivariate exponentially weighted moving average – MEWMA

Control charts are used to routinely monitor quality. Depending on the number of process characteristics to be monitored, there are two basic types of control charts. The first, referred to as a univariate control chart, is a graphical display (chart) of one quality characteristic. The second, referred to as a multivariate control chart, is a graphical display of a statistic that summarizes or represents two or more quality characteristic.



**Figure 15 – Example of control chart**

If a data point falls outside the control limits, we assume that the process is probably out of control and that an investigation is warranted to find and eliminate the cause or causes.

Different types of univariate control charts were developed during last century, but in practice the quality of a product is not defined only by one of its characteristics. Usually we have to consider more than just one characteristic of product to determine its quality. For this type of analysis, we are using multivariate control charts. One of the most common used multivariate control charts, is MEWMA.

The model for a MEWMA chart is given by:

$$Z_i = \Lambda X_i + (1 - \Lambda)Z_{i-1},$$

Where:

$Z_i$  = ith EWMA vector.

$X_i$  = ith product vector (values of multiple characteristics of ith product)

$\Lambda$  = diagonal matrix  $\text{diag}(\lambda_1, \lambda_2, \dots, \lambda_p)$ ,  $p$  is the number of characteristics of a product and

$\lambda_p$  = depth of pth characteristics (like EWMA  $\lambda$ ).

The following figure will illustrate it

<sup>1</sup> Niaki S, Ershadi M. Economic and economic-statistical designs of MEWMA control charts-a hybrid Taguchi loss, Markov chain, and genetic algorithm approach. In International Journal Adv. Manuf. Technology, 2010, 48, p. 283-296

$$\begin{matrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{matrix}$$

One row represents  $p$  different characteristics of one product. One column represents measurements of one characteristic of different products.

The quantity to be plotted on the MEWMA control chart is:

$$T_i^2 = Z_i' \Sigma_{Z_i}^{-1} Z_i$$

Where  $\Sigma_{Z_i}^{-1}$  is defined as:

$$\Sigma_{Z_i}(k, l) = \frac{\lambda}{2 - \lambda} [1 - (1 - \lambda)^{2i}] \Sigma,$$

And  $\Sigma$  is the covariance matrix of input data.

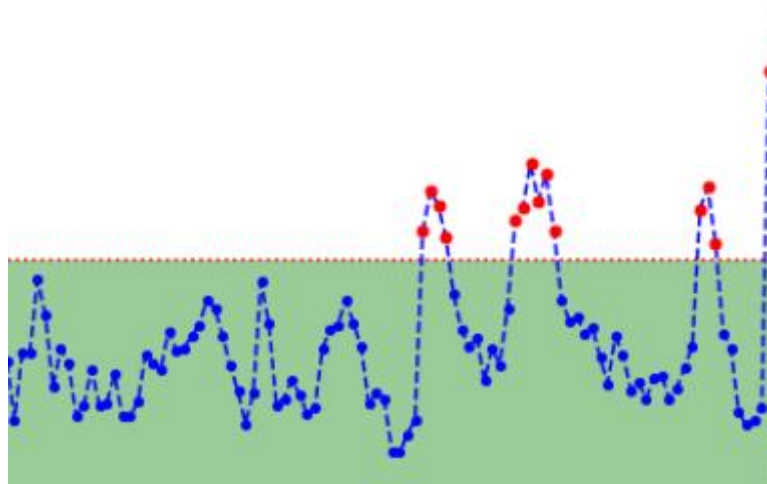
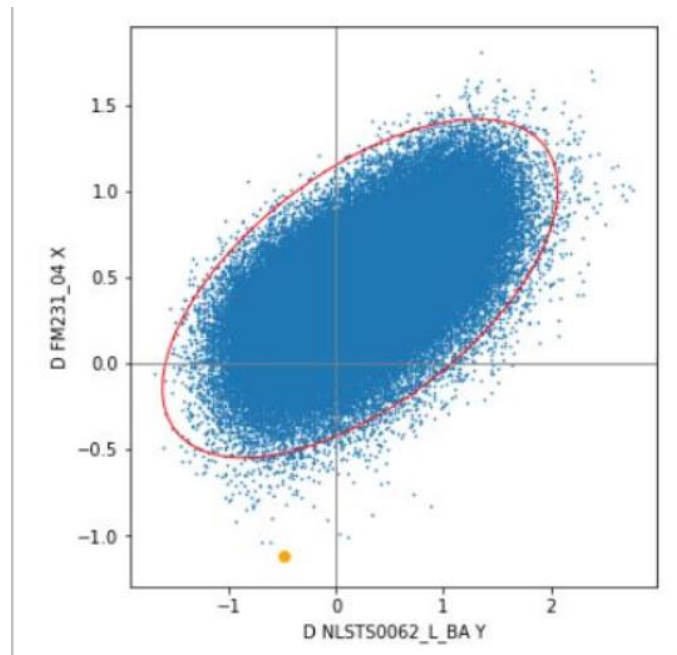


Figure 16 – An example of MEWMA control chart

We can also use  $p$  control charts (one control chart for each characteristics), but in this case we do not consider correlations between these characteristics. However, MEWMA is considering these correlations and it can detect an anomaly in multidimensional space. Following picture will illustrate how univariate control chart can “miss” an anomaly in two-dimensional case.



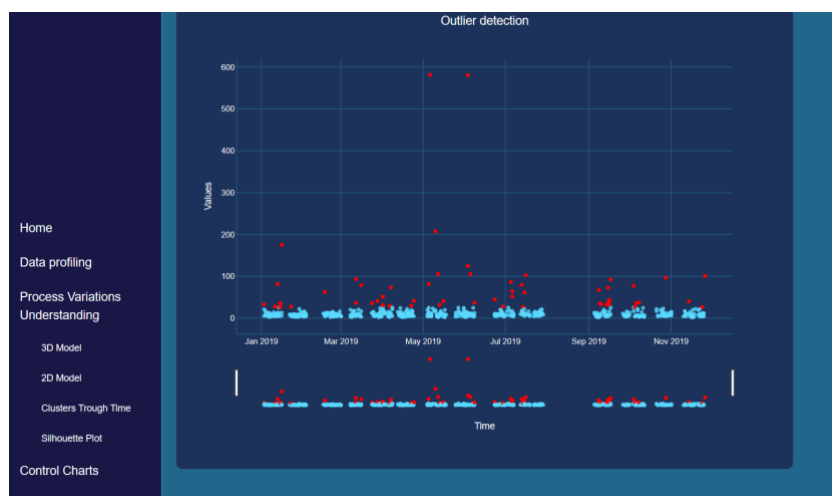
**Figure 17 – Influence of correlation**

As we can see orange dot is in control limit for both characteristics but since there is a correlation between these 2 parameters it must be inside the ellipse. Which means that we have found a product which is anomalous, but such anomaly cannot be detected using univariate control charts.

Second big advantage of MEWMA is that it allows us to weight each characteristic individually ( $\lambda\rho$ ). For example, if we have domain knowledge and we know that some characteristic is more important than others we can give it more weight.

MEWMA constraints is that it can be computational expensive. This can be solved by using smaller sample (more recent sample) than full data set since MEWMA gives less and less weight to “older” products.

In the following figure we present an illustration of the MEMWA anomaly detection (red dots) in D2Twin



**Figure 18 – MEWMA in action**

During this second iteration, the main improvement of the Industrial IoT and analytics platform has been the inclusion of a new subcomponent for anomaly processing. Then, the overall functionality, architecture and technical information about the Industrial IoT and analytics platform component for both lanes, FIWARE and APACHE, which was deeply described in “D4.3 MIDIH Open CPS/IOT Components v1” in their respective factsheets, are still valid:

- Annex D. Factsheet for Industrial IoT and Analytics Platform – Apache line
- Annex E. Factsheet for Industrial IoT and Analytics Platform – FIWARE line

## 5 Conclusions and Future Outlook

---

Deliverable D4.4 provides complete information about the components developed in the second iteration in tasks WP4.2, WP4.3 and WP4.4, in order to implement specific components of MIDIH reference architecture, providing a clear vision of the work carried out during this second period.

In **T2.2**, efforts have been focused on providing more effective Edge Computing capabilities into the MIDIH architecture, providing mechanisms for data processing, aggregation, etc..., helping in reducing the number of messages between the IoT Device and the cloud, resulting in network decongestion, increasing data processing speed and security. In the same way, new connectors have been developed like the FIWARE-Arrowhead allowing interoperability and extending the capabilities of the MIDIH architecture.

Task **T4.3**, focused on achieve a more structured approach to interconnect FIWARE with a commercial IoT platform such as Siemens MindSphere, as well as to interconnect with other third-party solutions, mainly providing interoperability between the two protocols NGSI and OPC UA.

In task **T4.4**, the Industrial IoT and Analytics Platform developed in the first iteration has been evolved in order to deal with more accurate processes and algorithms for anomaly processing. In the same way, the selection of most proper APACHE open source components that have similar characteristics for specific purposes has been carried out, mainly based on the requirements of the industrial experiments.

Until the end of the project some of the components developed in T4.2, T4.3 and T4.4 will continue evolving for improving some of the functionalities provided or covering specific requirements of industrial experiments.

# Annex A. Factsheet for Arrowhead Translator with FIWARE plugin

## 1 Arrowhead Translator with FIWARE plugin

### 1.1 INTRODUCTION

The FIWARE translator plugin has been developed to easily integrate an already deployed FIWARE network with the Arrowhead Services. It allows the FIWARE clients to produce and consume services to and from an Arrowhead local cloud and allows to register and orchestrate Systems and Services from FIWARE into Arrowhead.

### 1.2 FUNCTIONAL DESCRIPTION

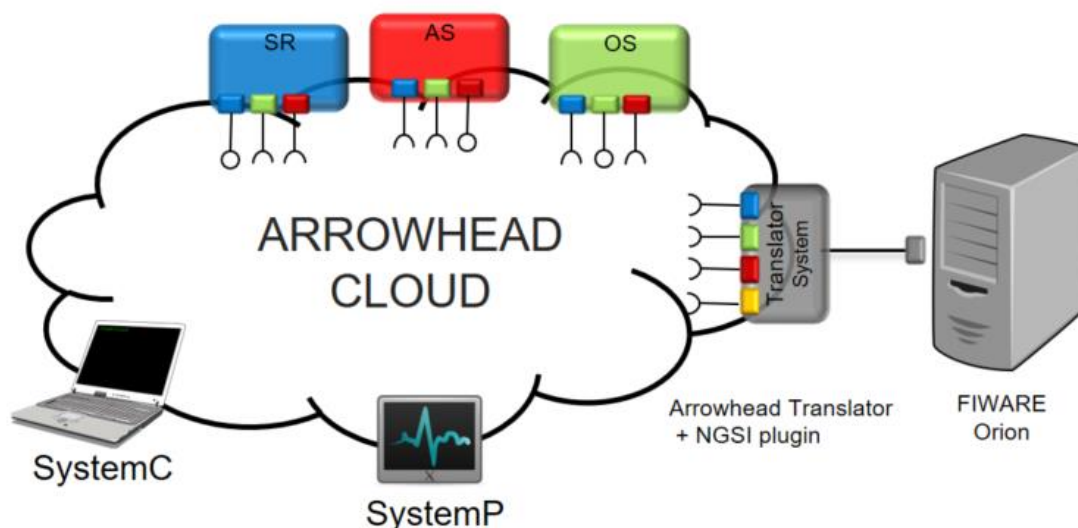


Figure 19 – Translator FIWARE plugin functional schema

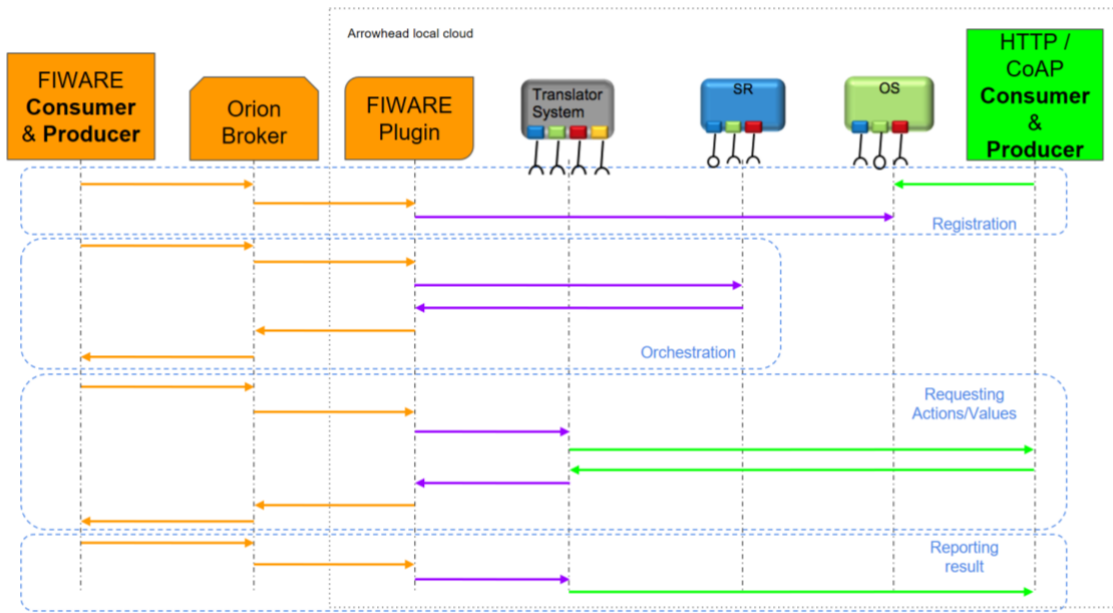
#### 1.2.1 Overview

The plugin has been developed to extend Arrowhead to FIWARE networks, two achieve this goal both ecosystems must coexist together (see Figure 19). One of the features of the plugin is that it must be transparent for producers and consumers at Arrowhead and at the FIWARE network. Therefore, the plugin must work between both networks doing all the Arrowhead tasks for the FIWARE devices like registration, and orchestration. But also, make accessible the arrowhead producers to FIWARE requests.

#### 1.2.2 Architecture and Specification

The architecture for communications is described on Figure 20. On that picture process for Registration, Orchestration, consumption and production are described.





**Figure 20 – FIWARE plugin description**

#### 1.2.2.1 Registration

When a FIWARE service is registered on the broker, the plugin generates a common http resource, and registers it on the Service registry. The resource is will translate from JSON to SenML formats.

#### 1.2.2.2 Orchestration

During a FIWARE service request, the plugin returns the information from two different sources. First form the FIWARE broker and second form the orchestrator, translating also the orchestrator response to a FIWARE response.

#### 1.2.2.3 Consuming a Service

A FIWARE “consumer” can demand new values to the broker, so at that point the plugin must request the new data to a “normal http arrowhead provider” and translate it to the FIWARE data model.

#### 1.2.2.4 HW/SW Prerequisite

The Arrowhead Translator plugin for FIWARE does not require any special Hardware of Software, as part of the Arrowhead Core Services the requirements are the same as Arrowhead Core Services. It only requires the use of Java 8 or 11.

Optionally a FIWARE broker can run in parallel with Arrowhead, to achieve that the use of Orion FIWARE broker is recommended. To use the latest version an installation of Docker is required.

### 1.2.2.5 *MIDIH Specific developments*

The complete plugin has been developed for MIDIH.

## 1.3 MANUALS

### 1.3.1 Installation

#### 1.3.1.1 *Translator FIWARE plugin*

The Translator is part of Arrowhead Core Services, there is no particular installation required for this Service. If you are running a version of Arrowhead without the Translator Service you can download, compile and run the latest version as follows.

#### 1. Get latest repository

Download it from: <https://github.com/thingwave/ArrowheadCore-Translator-Fiware>

Or Clone it:

- `git clone https://github.com/thingwave/ArrowheadCore-Translator-Fiware.git`

#### 2. Compile

- `mvn package`

#### 3. Run

- `java -jar /translator/target/arrowhead-translator-4.1.1-SNAPSHOT.jar`

#### 4. Test it

Now the Translator service is running on port 8462 (insecure) or 8463 (secure).

#### 1.3.1.2 *ORION – FIWARE broker*

The Arrowhead Translator plugin for FIWARE does not require a FIWARE broker to run properly, therefore this step is completely optional.

#### 1. Installation

The installation requires to download the latest version of Orion and MongoDB, and create a network.

- `docker pull mongo:3.6`
- `docker pull fiware/orion`
- `docker network create fiware_default`

#### 2. How to run it

Mongo will run on port 27017 and Orion at 1026.

- `docker run -d --name=mongo-db --network=fiware_default --expose=27017 mongo:3.6 --bind_ip_all --smallfiles`
- `docker run -d --name=fiware-orion -h orion --network=fiware_default -p 1026:1026 fiware/orion -dbhost mongo-db -corsOrigin __ALL`

### 3. Clean and clear all data

You need to stop the containers, remove them and delete the network.

- `docker stop fiware-orion`
- `docker rm fiware-orion`
- `docker stop mongo-db`
- `docker rm mongo-db`
- `docker network rm fiware_default`

#### 1.3.2 Configuration

The Translator plugin runs as part of the Translation Service; therefore no configuration is required.

#### 1.3.3 Other resources and tools

To test all the functionalities of the plugin there is also an example software at <https://github.com/thingwave/Arrowhead-FiwareDemo>. This demo code generates a temperature provider for arrowhead (A) with random values and a temperature provide for FIWARE (B). At the same time there are four consumers accessing the data in the four different possible ways: normal http arrowhead consumer consuming A, normal http arrowhead consuming B, normal FIWARE client requesting A, and normal FIWARE client requesting B.

##### 1. How to compile

Compile it as normal maven package.

- `mvn package`

##### 2. How to run it

Executing the following command, it will generate a table with all the consumers and produces on a terminal.

- `java -jar target/arrowhead-fiware-demo-0.0.1-jar-with-dependencies.jar`

#### 1.3.4 Licensing

The solutions are open source.

# Annex B. Factsheet for Arrowhead Framework demonstrator

---

## 1 Arrowhead Framework demonstrator

### 1.1 INTRODUCTION

In a traditional manufacturing management paradigm, path from planning to physical execution consists of a chain of command and a sequence of manual operations which need to be completed by workers who handle the manufacturing processes. In general, those operations cover defining, addressing, synchronizing, executing and supervising the processes. Effectiveness of this approach is limited by unstandardized means of communication and human labor capacity.

Modern factories still use traditional paper forms, carrying the data to endpoint operators of a production process or to the machines involved.

In order to eliminate these bottlenecks in a real-world scenario, one could add a new abstraction layer to the management chain, for example by extending a given brownfield infrastructure with interfaces between the workers (or even customers) and the machines. This can be achieved using Arrowhead Framework, which is what we describe here.

### 1.2 FUNCTIONAL DESCRIPTION

#### 1.2.1 Overview

In order to simulate a fragment of a brownfield infrastructure we decided to build a robotic arm, which was supposed to move objects from one place to another – a simple, repeatable process, which represents a generic technological process.

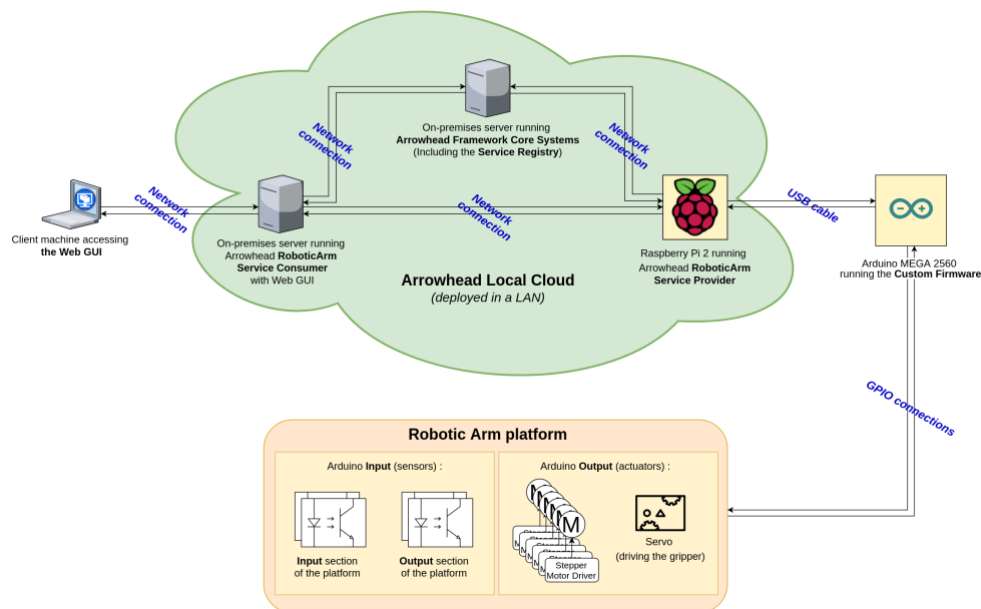
Initially then, we assumed the firmware on the controller to only contain preprogrammed moves of the arm, and *some legacy logic*.

We then try to *optimize the management* of our mocked “production line” in two ways:

- **locally (*affecting a particular machine*)**, by adding an interface between an operator (consumer), and the machine (provider), which in this case involved:
  - adding remote control capabilities to the controller’s firmware,
  - extending the controller’s interface by adding a RaspberryPi and appropriate code,
- **globally (*across many machines*)**, by using Arrowhead Framework, which consisted of:
  - launching Arrowhead Core in a local network,

- clearly defining the “producer” (the robotic arm), “consumer” (an operator) and “services” (in this case there were 2, corresponding to 2 preprogrammed moves which the robot can perform),
- heavily extending a “provider” code skeleton (supplied with the framework) to appropriately wrap the machine’s interface (Arduino + RaspberryPi) into the Arrowhead Framework platform,
- extending “consumer” code skeleton and creating a web GUI allowing the user to launch our “technological process” on any “provider” currently available – abstracting away the task of choosing a particular machine is taken care of by the Framework.

### 1.2.2 Architecture and Specification



**Figure 21 – Arrowhead Framework Robotic Arm Demonstrator – Architecture diagram**

The above diagram describes the final architecture of our demonstrator. *Note:*

- The “Service Consumer” doesn’t have to be run separately from the “Client machine”,
- Our assumed “Brownfield infrastructure” consisted of the “Robotic Arm platform” and the Arduino Board, BUT without any convenient interface.
- The Robotic Arm Platform consists of:
  - 3D-printed parts,
  - Belt-driven joints,
  - Stepper-motors and DC motor drivers,
  - Arduino MEGA 2560 board,
  - Two 3D-printed platforms (an “input” and an “output”),
  - Four optical sensors.

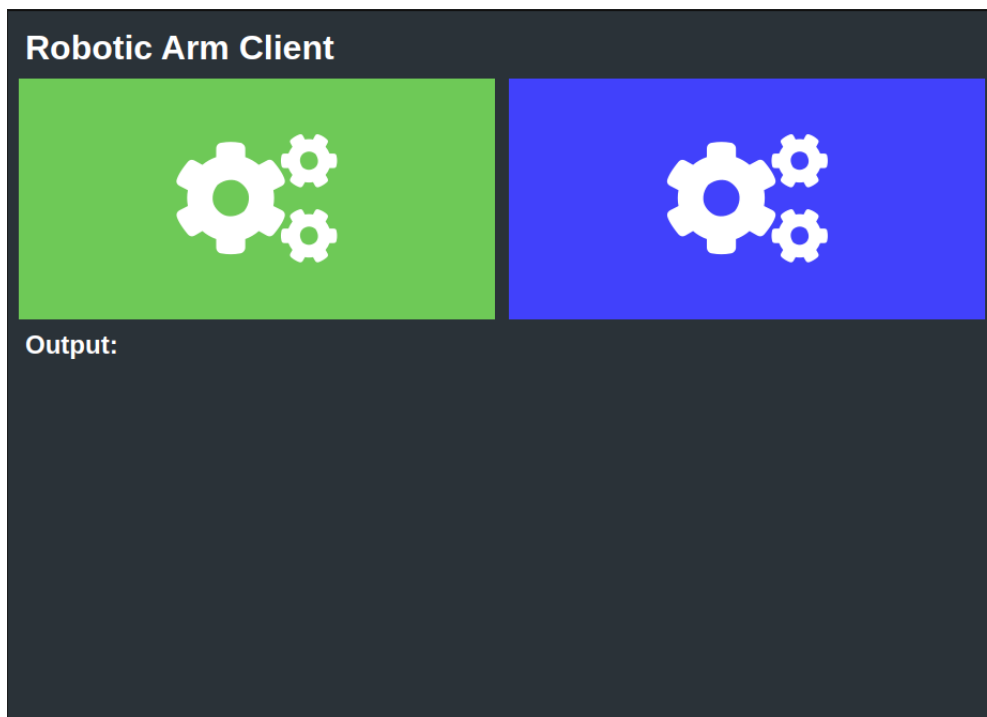


Figure 22 – Screenshot of the consumer Web GUI

### 1.3 MANUALS

- [Video of the robot in action](https://www.youtube.com/watch?v=iA08QSWg9No)
  - <https://www.youtube.com/watch?v=iA08QSWg9No>
- [The Arduino firmware](https://git.man.poznan.pl/stash/projects/KWH/repos/midih-arrowhead-arduino-firmware/)
  - <https://git.man.poznan.pl/stash/projects/KWH/repos/midih-arrowhead-arduino-firmware/>
- [The Service Provider/Consumer](https://git.man.poznan.pl/stash/projects/KWH/repos/midih-arrowhead-robotic-arm/browse)
  - <https://git.man.poznan.pl/stash/projects/KWH/repos/midih-arrowhead-robotic-arm/browse>
- [Simple web GUI](https://git.man.poznan.pl/stash/projects/KWH/repos/midih-robotic-arm-ui/browse)
  - <https://git.man.poznan.pl/stash/projects/KWH/repos/midih-robotic-arm-ui/browse>

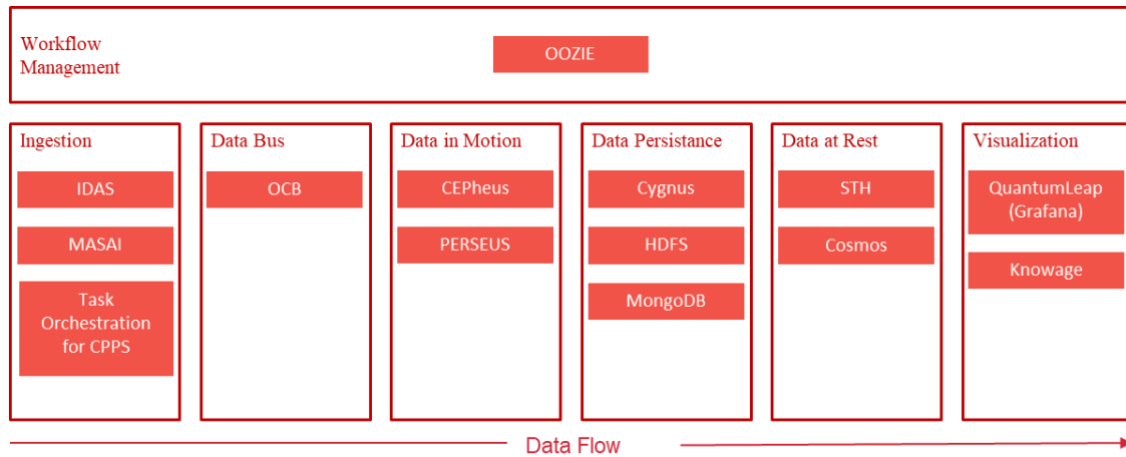
# Annex C. Factsheet for Industrial IoT platform for robotics application

---

## 1 Industrial IoT platform for robotics application

### 1.1 INTRODUCTION

In this robotic application, the IoT computing platform is based on the FIWARE lane implementation of the MIDIH reference architecture model (Figure 23). The platform<sup>2</sup> has an option for real time streaming of data via 5G.



**Figure 23 – MIDIH Industrial IoT and Analytics Platform Architecture (FIWARE Pipeline)**

The MIDIH architecture offers two lanes for implementing applications: FIWARE4Industry and APACHE lanes. FIWARE4Industry is an ecosystem that offers FIWARE-based software (open source) components for the manufacturing domain (<http://www.fiware4industry.com>). Apache has similar components as the FIWARE for developing IoT platforms. APACHE is also an open source software ecosystem ([www.apache.org](http://www.apache.org)) and it is supported by the Apache foundation.

The IoT computing platform for the robotics application is based on FIWARE lane.

### 1.2 FUNCTIONAL DESCRIPTION

#### 1.2.1 Architecture

In Figure 24 the high-level architecture of the IoT platform is shown.

---

<sup>2</sup> Koskinen, J., Tikka, P. & Tanner, H., Industrial IoT applications, Proceedings of Automaatiopäivät23, Oulu 15-16 May, 2019

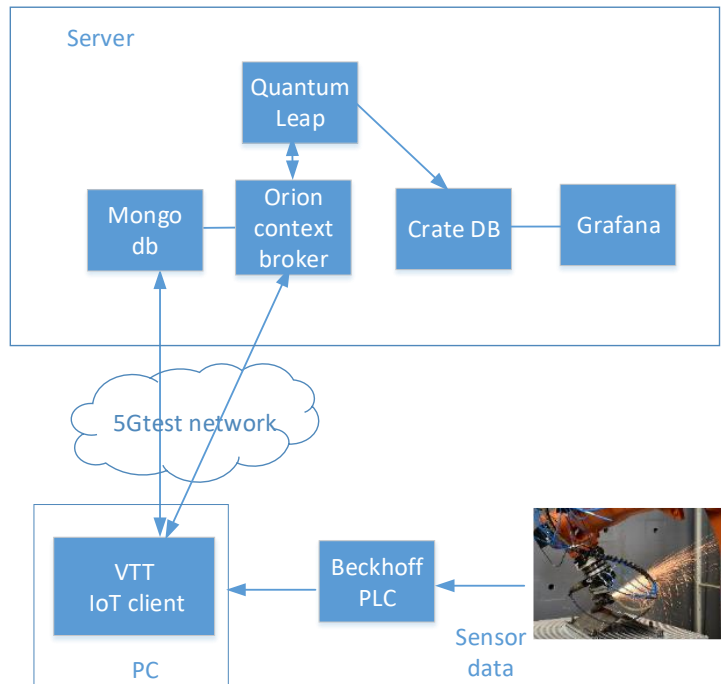


Figure 24 – Architecture of the edge-computing platform

#### 1.2.1.1 HW/SW Prerequisite

The IoT platform (server) is running in a virtual machine in a laptop Pc with Ubuntu 18.04.

The platform consists of the following open source components (see Figure 24). The instruction of the installation and are described in section 1.3.

- **Orion Context Broker:** OCB receives requests from subscribers. It delivers data from sources (sensors) to subscribers/context consumers.
- **MongoDB:** OCB uses MongoDB for storing information about NGSI data entities (registrations, subscriptions). An entity is a virtual representation of real-world objects with attributes.
- **Quantum Leap:** This component is an OCB subscriber. When a new value arrives to the OCB, Quantum Leap parses and validates the data and stores it in a time-series database.
- **Crate DB:** Crate DB is an SQL Data Base Management System, which supports real time querying and time series data. Crate supports NGSI v2 format.
- **Grafana:** This is an analytics tool for visualization of time-series data. It supports basic analytics functions such as maximum/minimum, average, smoothing etc. Grafana uses CrateDB.

The platform has also following SW and HW components:

- **VTT IoT client.** This client communicates with OCB and Beckhoff PLC.
- **Beckhoff PLC.** Robots sensors (forces sensors) are connected to the PLC.



### 1.2.1.2 MIDIH Specific developments

The platform has an option to communicate via 5g network (<https://5gtnf.fi/>) in the facilities of VTT.

VTT IoT client. This client communicates with OCB and Beckhoff PLC. It is a proprietary component developed by VTT in another project and modified to be used in this MIDIH application.

## 1.3 MANUALS

Installation and configuration of the components can be found from following links

Orion context broker & MongoDB:

[https://fiware-orion.readthedocs.io/en/master/quick\\_start\\_guide/index.html](https://fiware-orion.readthedocs.io/en/master/quick_start_guide/index.html)

Quantum Leap:

<https://documenter.getpostman.com/view/513743/RWEnkvDc?version=latest>

CrateDB:

<https://fiware-tutorials.readthedocs.io/en/1.0.0/time-series-data/index.html>

Grafana:

<https://fiware-tutorials.readthedocs.io/en/1.0.0/time-series-data/index.html#displaying-cratedb-data-as-a-grafana-dashboard>

## Annex D. Factsheet for Edge-oriented Local Clouds for Factory Automation – FogFlow

---

### 1 Edge-oriented Local Clouds for Factory Automation – FogFlow

#### 1.1 INTRODUCTION

FogFlow is a framework based on fog computing, it is based on the idea of extreme programming. The FogFlow technology was developed at NEC Laboratories Europe, whose main creator was Bin Cheng, a NEC employee expert of Fog Computing. FogFlow offers on-demand processing of contextual information, enhances the possibilities offered by FIWARE to bring computing closer to the IoT devices that control and collect data of real-world interest. FogFlow is an open source system and respects the NGSI standard also used by FIWARE environment for data transmission, this framework has been approved by the FIWARE Foundation as generic enabler in the open source FIWARE ecosystem.

#### 1.2 FUNCTIONAL DESCRIPTION

##### 1.2.1 Overview

One of the more interesting FogFlow capabilities is carry out the processing of data produced by IoT Devices in the local environment, with the aim of bringing all the necessary computing resources to the extreme to avoid having to resort to the cloud. FogFlow can be applied to any kind of work, it contains a complex task control system, consultation and management of the context data produced by the different tasks, which are controlled by IoT Discovery.

##### 1.2.2 Architecture and Specification

The framework follows a three-tier structure, the service layer that is in charge of high-level services such as docker deployment and two important elements of FogFlow:

- Task Designer: Web tool to visually manage, create and delete the tasks that will be deployed on the fog/cloud nodes.
- Topology Master: Component that based on the configured application topology, will decide when, where and which of the tasks are deployed, whether in the cloud or at the ends.

The second layer is the context layer, where all context data is located, this layer is in charge of manage, store and distribute the data through the distributed application. This layer is mainly composed by the following two components:

- IoT Discovery: it is in charge of processing context information with its id, attributes, metadata, etc., and allowing other elements to query and subscribe to the data. It is used by the local IoT Broker to query for entities located in other IoT Brokers contained on different edge nodes.

- IoT Broker: it is in charge of manage the local context entities that can be produced by nearby IoT Devices integrated on the Edge Node, providing a single view of all the entities that provides input streams to the tasks deployed on the edge node. It can be also used to provide output streams to be consumed by tasks deployed on other Edge nodes or in the Cloud.

Finally, there is the data processing layer, which, as its name suggests, where the data from input streams is processed, optionally producing output streams after process or analyse the input. This layer is mainly composed by the following two components:

- Worker: coinciding with the topology master, each worker launches his tasks in docker containers on his local machine. Defines the inputs and outputs of tasks and manages them according to their priority.
- Operator: The operators are the objects that contain all the data processing logic of a service topology. The way in which data is processed in FogFlow is through tasks, as we have said previously on this document, but the way in which FogFlow deals with tasks is through the operators. An operator can be implemented using python or JavaScript, in the FogFlow documentation there is a developed and explained example of how to implement an operator.

Figure 25 illustrates how the different interactions between the Worker, IoT Broker and deployed tasks occurs. This interaction is further explained in detail next.

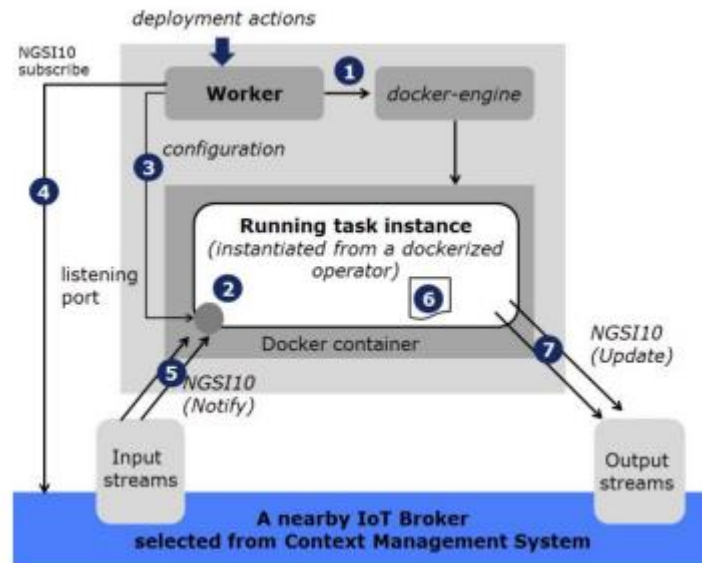


Figure 25 – Data Flow Processing

The Worker informs the docker that it wants to create a new container with an instance of a task, the inputs and outputs are configured for the task. Once this is done the task starts listening on the given port waiting to receive incoming data streams. The Worker can send a configuration object to the task through the listening port. From the IoT Broker you can make subscription requests for the task through NGSI10 messages. When the task receives incoming flows, processes the data and sends outgoing data flows, all subscriptions are notified at that time.

Next picture (Figure 26) illustrates the full deployment of the FogFlow system, you can see three parts: a first node that corresponds to the cloud, formed by the IoT Discovery, the IoT Broker, the Worker, the Task Designer and the Topology Master, which have been explained above, but

has an element called RabbitMQ, a message bus based on AMQP for the exchange of control commands between the Topology Masters and the Workers. Another node that corresponds to the border node, where the IoT Devices are located along with the worker and the IoT Broker. Finally, we have a Docker Registry that is in charge of managing the images of the operators' registration couplers.

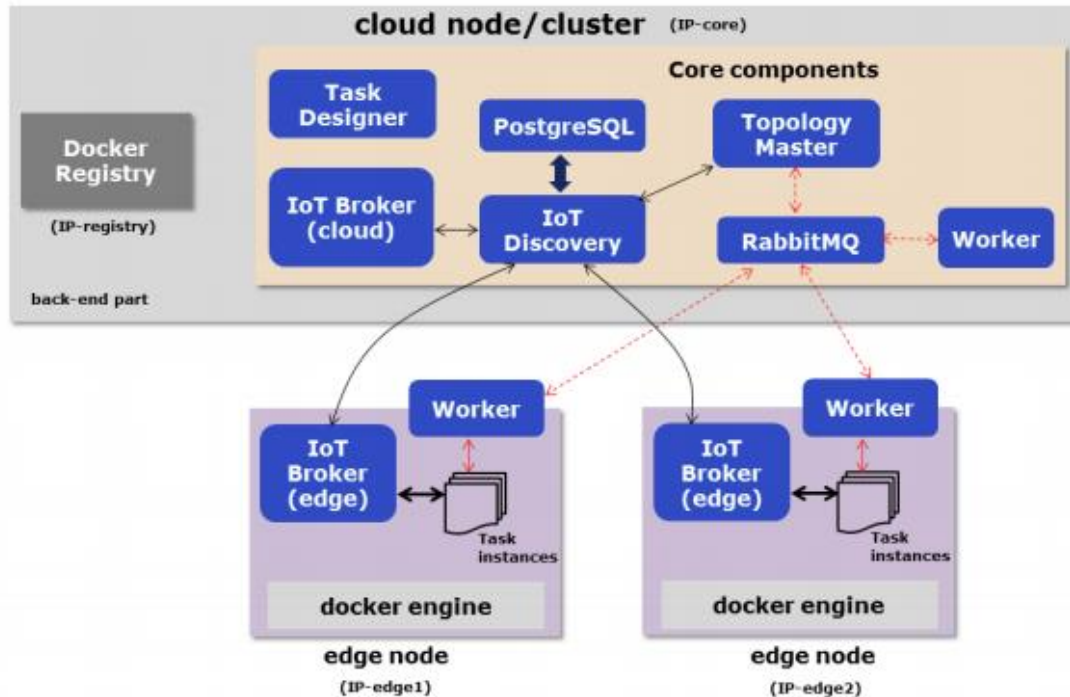


Figure 26 – Deployment view

## 1.3 MANUALS

The following is the FogFlow deployment method that has changed significantly since the version presented in D4.3 and is currently used by using Docker service deployment technology that makes it easier to configure and start up.

First the Cloud part of FogFlow is in charge of managing and handling the FogFlow Edge nodes, which are the ones that perform the workload.

### 1.3.1 FogFlow Cloud Deployment

#### 1.3.1.1 Prerequisites

Here are the prerequisite commands for starting FogFlow:

1. Docker
2. Docker-compose

For ubuntu-16.04, you need to install docker-ce and docker-compose.

To install Docker CE, please refer to [Install Docker CE](#), required version > 18.03.1-ce;

### 1.3.1.2 Fetch all required scripts

Download the docker-compose file and the configuration files as below.

```
mkdir fogflow-cloud
cd fogflow-cloud

# the docker-compose file to start all FogFlow components on the cloud node
wget https://raw.githubusercontent.com/smartfog/fogflow/master/docker/core/http/docker-compose.yml

# the configuration file used by all FogFlow components
wget https://raw.githubusercontent.com/smartfog/fogflow/master/docker/core/http/config.json

# the configuration file used by the nginx proxy
wget https://raw.githubusercontent.com/smartfog/fogflow/master/docker/core/http/nginx.conf
```

### 1.3.1.3 Configuration

You need to change the following IP addresses in config.json according to your own environment:

- **coreservice\_ip**: it is used by all FogFlow edge nodes to access the core services (e.g., nginx on port 80 and rabbitmq on port 5672) on the FogFlow cloud node; usually this will be the public IP of the FogFlow cloud node.
- **external\_hostip**: for the configuration of the FogFlow cloud node, this is the same as coreservice\_ip used by the components (Cloud Worker and Cloud Broker) to access the running FogFlow core services;
- **internal\_hostip**: this is the IP of your default docker bridge, which is the “docker0” network interface on your Linux host. For the docker engine on Windows or Mac OS, there is no “docker0” network interface; instead, you need to use the special domain name “host.docker.internal”.
- **site\_id**: each FogFlow node (either cloud node or edge node) requires to have a unique string-based ID to identify itself in the system;
- **physical\_location**: the geo-location of the FogFlow node;
- **worker.capacity**: it means the maximal number of docker containers that the FogFlow node can invoke;

### 1.3.1.4 Start all components on the FogFlow cloud node

Pull the docker images of all FogFlow components and start the FogFlow system

```
# if you already download the docker images of FogFlow components, this command can fetch the updated images
docker-compose pull

docker-compose up -d
```

### 1.3.1.5 Validate the setup

There are two ways to check if the FogFlow cloud node is started correctly:

- Check all the containers are Up and Running using “docker ps -a”

```
docker ps -a
```

```

PS C:\Users\...> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED    STATUS    PORTS                               NAMES
1f21c748b28a   fogflow/worker "/worker"               5 seconds ago Up Less than a second 0.0.0.0:1060->1060/tcp fogflow-cloud_clo
ud_worker_1
940761b34257   fogflow/master "/master"               5 seconds ago Up Less than a second 0.0.0.0:8070->8070/tcp fogflow-cloud_mas
ter_1
a1a12c94a4bb   fogflow/broker "/broker"               7 seconds ago Up Less than a second 0.0.0.0:8090->8090/tcp fogflow-cloud_dis
covery_1
a30ff7e43c7    fogflow/discovery "/discovery"           8 seconds ago Up Less than a second 0.0.0.0:8090->8090/tcp fogflow-cloud_rab
bitmq_1
6bb73e947597   rabbitmq:3     "docker-entrypoint.s..." 8 seconds ago Up 7 seconds 4369/tcp, 5671/tcp, 25672/tcp, 0.0.0.0:5672->5672/tcp fogflow-cloud_rab
bitmq_1
e1533e76f855   fogflow/lotajson-mongo "docker-entrypoint.s..." 8 seconds ago Up 7 seconds 0.0.0.0:4041->4041/tcp, 0.0.0.0:7896->7896/tcp, 27017/tcp lot-agent-JSON
draco_mongo_1
97710682d571   mongo:3.4     "docker-entrypoint.s..." 2 weeks ago Up 20 minutes 0.0.0.0:27017->27017/tcp draco_mongo_1
  
```

**Important** if you see any container is missing, you can run “docker ps -a” to check if any FogFlow component is terminated with some problem. If there is, you can further check its output log by running “docker logs [container ID]”

- Check the system status from the FogFlow DashBoard

You can open the FogFlow dashboard in your web browser to see the current system status via the URL: [http://<coreservice\\_ip>/index.html](http://<coreservice_ip>/index.html)

**Important** If the FogFlow cloud node is behind a gateway, you need to create a mapping from the gateway IP to the coreservice\_ip and then access the FogFlow dashboard via the gateway IP; If the FogFlow cloud node is a VM in a public cloud like Azure Cloud, Google Cloud, or Amazon Cloud, you need to access the FogFlow dashboard via the public IP of your VM;

Once you are able to access the FogFlow dashboard, you can see the following web page:

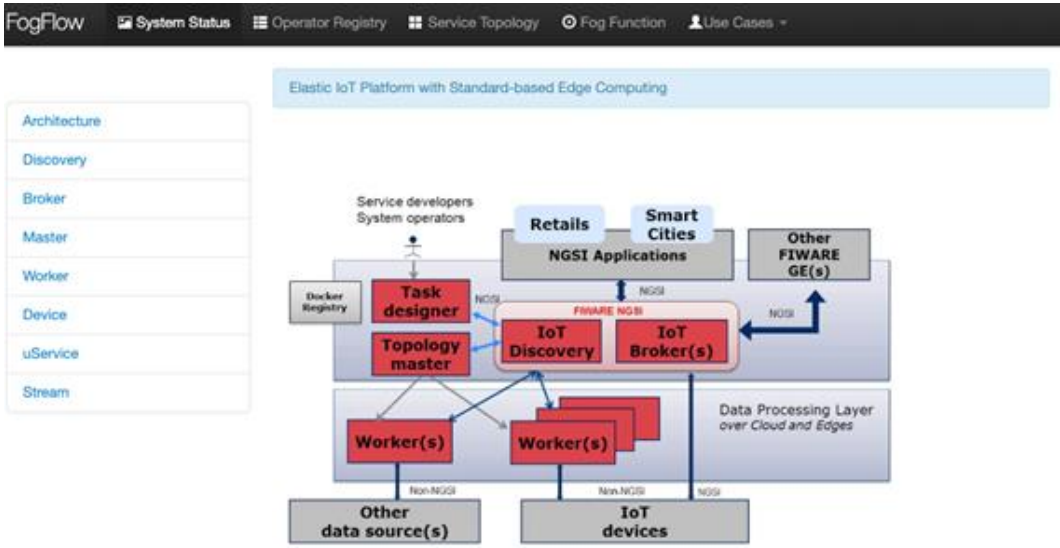


Figure 27 – FogFlow welcome page

### 1.3.2 FogFlow Edge & Adapter Deployment

Here we explain how to deploy the Edge part of FogFlow to perform the work units and its deployment with the adapter and a virtual device that reports values.

The adapter is located between the FIWARE IoT Agent and the FogFlow IoT Broker, so it is necessary to deploy the IoTAgent service, its MongoDB database, the adapter and the IoT Broker on the edge node that we want to be part of the FogFlow ecosystem.

```
mkdir fogflow-edge
cd fogflow-edge

# the docker-compose file to start the FogFlow components on the edge node
wget https://raw.githubusercontent.com/smartfog/adapter/master/GeneralPurpose-Adapter/edge-
deployment/docker-compose.yml

# the configuration file used by the FogFlow IoT Broker
wget https://raw.githubusercontent.com/smartfog/adapter/master/GeneralPurpose-Adapter/edge-
deployment/broker-config.json

# the configuration file of the LwM2M IoT Agent
wget https://raw.githubusercontent.com/smartfog/adapter/master/GeneralPurpose-Adapter/edge-
deployment/config.js
```

The configuration file of the IoT Broker (broker-config.json) needs to be set in the same way as the one used to deploy the FogFlow Cloud nodes. After configuring the Edge node that will connect to the Cloud services deployed by FogFlow as we wish, the next step is to deploy the FogFlow edge services.

```
# if you already download the docker images of FogFlow components, this command can fetch the updated images
docker-compose pull

docker-compose up -d
```

After the successful deployment of all components, the next step is to deploy a virtual appliance to connect to the Edge IoT Broker and start uploading values to FogFlow.

```
mkdir virtual-device
cd virtual-device

# the docker-compose file to start the virtual device on the edge node
wget https://raw.githubusercontent.com/smartfog/adapter/master/GeneralPurpose-Adapter/virtual-
device/docker-compose.yml

# the configuration file used by the virtual device
wget https://raw.githubusercontent.com/smartfog/adapter/master/GeneralPurpose-Adapter/virtual-
device/devices.json
```

In the devices.json file it is necessary to configure the **serverUrl** field with the IP address of the device where the IoTAgent is deployed.

Finally, the virtual appliance is deployed to start uploading values to FogFlow.

```
# if you already download the docker images, this command can fetch the updated images
docker-compose pull

docker-compose up -d
```

This device has two attributes: temperature and humidity, the values of these are generated randomly between a maximum, a minimum and an initial value specified in the devices.json file.

More detailed information for deployment can be found in the official [FogFlow Adapters repository](#).

## 1.4 Licensing

FogFlow license is BSD4

NGSI Adapter license is Apache, Version 2.0



# Annex E. Factsheet for FIWARE OPC UA Agent

## 1 FIWARE OPC UA Agent

### 1.1 INTRODUCTION

The OPC UA Agent was integrated in the first release of the MIDIH Platform and it was deployed and validated in the POLIMI business scenario.

During the experimentation, feedbacks paved the way for new requirements and improvements to be designed and developed for the second iteration.

### 1.2 FUNCTIONAL DESCRIPTION

#### 1.2.1 Overview

OPC UA is the data exchange standard for *Industrie 4.0* enabling the industrial communication (machine to machine). For that reason, the OPC UA Agent is a core component of the FIWARE for Industry Architecture for connecting the plant to the cloud.

Moreover, it can be integrated in many scenarios concerning Robotics. OPC, in fact, specified specification for robotics, using the OPC UA Information Model for the representation of a complete motion device system as an interface for higher-level control and evaluation systems. A motion device system consists out of one or more motion devices, which can be any existing or future robot type (e.g. industrial robots, mobile robots), kinematics or manipulator as well as their control units and other peripheral components.

#### 1.2.2 Architecture and Specification

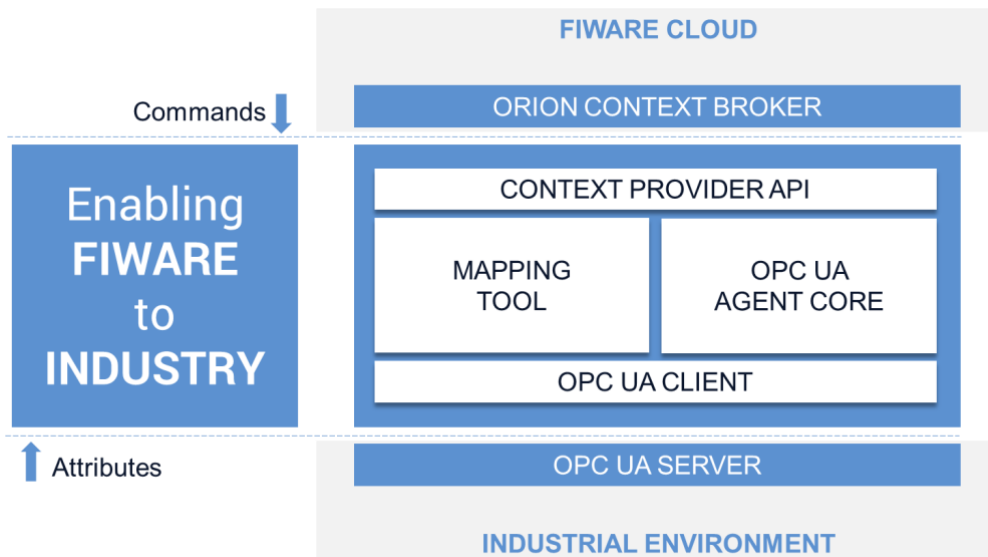


Figure 28 – OPC UA Agent Architecture

During the second development stage, the OPC UA Agent was improved, main changes concerning:

- Enabling common IDAS API for handling devices using REST services
- Tuning the OPC UA session parameters to enhance the agent performance
- Improvement the Mapping Tool in order to empower the generation of the configuration file
- Increase test coverage and documentation
- Bug fixing

#### 1.2.2.1 HW/SW Prerequisite

As described in the D4.3, Node.js is the most relevant technological prerequisites, since it constitutes the grounding technologies needed to implement the FIWARE OPC UA Agent.

In order to run the agent, it is necessary to have an OPC UA server (it must be reachable, typically in the same agent network) and an Orion Context Broker instance for exchanging data with the cloud. As depicted in Figure 28, the OPC UA server represents the physical layer (industrial environment) and can be defined as the southbound part of the architecture. Whereas, the northbound part consists of the Orion Context Broker, the entry point of the FIWARE ecosystem.

#### 1.2.2.2 MIDIH Specific developments

During the experimentation at POLIMI premises, the OPC UA agent was tested and validated. The validation process allowed to define new requirements mainly related to the OPC UA session parameters being able to register variable changes in the order of 50ms. Here a short description of the most relevant parameters:

- **samplingInterval** defines, for each monitored item, the interval used by the server to evaluate changes in the value of the variable. The actual value chosen by the server depends on the underlying hardware capabilities.
- **publishingInterval** defines the interval that has to elapse to notify possible changes to the client. Whenever the server discovers a change in the value of a variable, this new value is stored into the queue. When publishing interval expires the entire queue is sent to the client. In this way, the client can receive even very fast changes that otherwise would not have been detected.
- **queueSize** defines, for each monitored item, the size of the queue within which to store changes in the variable. When samples are sent to the client the queue is emptied.

Moreover, it was introduced the dynamic configuration, full supporting the IDAS API, a common set of services in order to manage devices dynamically. For that reason, the user can configure the agent:

- statically, creating manually or via Mapping Tool the configuration file (config.json as described in the D4.3)
- dynamically, configuring devices using the common IDAS API layer.

Furthermore, the Mapping Tool was enhanced. The Mapping Tool is a facilitation tool to support the manual (static) configuration. During the experimentation, it was introduced new configuration parameters for improving the filtering (in and out) and obtain a configuration file tailored to the user needs.

Finally, it was increased the test coverage, improved documentation and fixed bugs in order to end the incubation period of the agent and make it an official IDAS agent in the FIWARE Catalogue.

## 1.3 MANUALS

### 1.3.1 Installation

All the source code and a packed docker image can be retrieved at:

- <https://github.com/Engineering-Research-and-Development/iotagent-opcua>
- <https://github.com/Engineering-Research-and-Development/idas-opcua-agent-support-tool>

The installation process can be started by using the following command, providing the capability to download and install all the required dependencies:

```
npm install
```

### 1.3.2 Configuration

For the Agent to work an initialization phase is required. During this phase the Agent becomes aware of what variables and methods are available on OPC UA server side. These information can be provided to the agent by means of a configuration file (*config.json*) or through the REST API. Three different initialization modalities are available:

- Use a preloaded *config.json*
- Invoke a mapping tool responsible of automatically building the *config.json* (before using the MappingTool, delete the current *config.json* and comment "configuration=api" row in *config.properties*)
- Use the REST API (default empty *config.json* is preloaded)

## PROVISION A NEW DEVICE

```
curl http://localhost:4001/iot/devices \
  -H "fiware-service: {FIWARE_SERVICE}" \
  -H "fiware-servicepath: {FIWARE_SERVICE_PATH}" \
  -H "Content-Type: application/json" \
  -d @add_device.json
```

By Device we mean the set of variables (attributes) and methods available on OPC UA Server side. To provision the Device corresponding to what the OPC UA offers, use the following REST call:

Where *add\_device.json* should have the following structure:

```
{
  "devices": [
    {
      "device_id": "{DEVICE_ID}",
      "entity_name": "{NGSI_ENTITY}",
      "entity_type": "Device",
    }
  ]
}
```

```

    "attributes": [
      {
        "object_id": "{OPC-UA-NODE-ID}",
        "name": "{NGSI-ATTRIBUTE}",
        "type": "{NGSI-DATA-TYPE}"
      }
    ],
    "commands": [{
      "object_id": "{OPC-UA-NODE-ID}",
      "name": "{NGSI-ATTRIBUTE}",
      "type": "command"
    }]
  }
]
}

```

The *add\_device.json* sample payload contains several attributes even of different type. Some of them are missing on the OPC UA Server side but have been included to prove that the Agent is able to manage such situations.

## GET DEVICES

Check if the operation went well, by sending the following REST call:

```

curl http://localhost:4001/iot/devices \
  -H "fiware-service: {FIWARE_SERVICE}" \
  -H "fiware-servicepath: {FIWARE_SERVICE_PATH}"

```

The service returns a JSON indicating that there is one device.

## MONITOR AGENT BEHAVIOUR

Any activity regarding the Agent can be monitored looking at the logs. To view docker testbed logs run:

```

cd iotagent-opcua
docker-compose logs -f

```

Looking at these logs is useful to spot possible errors.

## HOW TO SEND A COMMAND

In order to send a command (method in OPC UA jargon), the request has to be sent to Orion that forwards the request to the OPC UA Agent:

```

curl -X PUT \

```

```
'http://  
{OCB_URL}:{OCB_PORT}/v2/entities/{NGSI_ENTITY}/attrs/{NGSI_ATTR}?type=Devi  
ce' \  
-H 'content-type: application/json' \  
-H 'fiware-service: {FIWARE_SERVICE}\  
-H 'fiware-servicepath: {FIWARE_SERVICE_PATH}\  
-d '{  
  "value": [2],  
  "type": "command"  
}'
```

### 1.3.3 Usage

The agent can be started by using the following command:

```
node index.js
```

It is possible to redirect the output log on a file by using:

```
node index.js > out.log
```

Using the docker compose, the agent can be run typing the command:

```
docker-compose up &
```

### 1.3.4 Licensing

The FIWARE OPC UA Agent is released under the GNU Affero General Public License v3.0.

# Annex F. Factsheet for FI-MIND

## 1 FI-MIND Bridge

---

### 1.1 INTRODUCTION

The FI-MIND bridge allows to enrich MindSphere environment with FIWARE context data (legacy factory systems, non-production IoT data sources, etc.).

**Siemens' MindSphere**, a cloud-based, open Internet of Things (IoT) operating system, enables industries worldwide to link their machines and physical infrastructure to the digital world easily, quickly and economically. Harnessing data from virtually any number of connected intelligent devices, enterprise systems and federated sources allows for analysis of real-time operational data. This analysis then leads to optimized processes, resource and productivity gains, the development of new business models and the reduction of operations and maintenance costs. Companies leveraging MindSphere boost performance, sharpen their competitive advantage and realize much more profitability<sup>3</sup>.

### 1.2 FUNCTIONAL DESCRIPTION

#### 1.2.1 Overview

The FI-MIND bridge is a JAVA web application. It exposes APIs in order to receive NGSI data manually, by a REST client, or automatically after the binding with the Orion Context Broker as destination of notifications.

#### 1.2.2 Architecture and Specification

##### 1.2.2.1 HW/SW Prerequisite

- JAVA 1.8
- MAVEN
- TOMCAT 9

##### 1.2.2.2 MIDIH Specific developments

During the MIDIH project, a Smart Industry DATA Model was defined (SIDAM) starting from FIWARE Data Models related to the Industry domain. For that reason and in order to obtain a high level of interoperability, the FI-MIND bridge was improved, making it able to process automatically data described in standard data models. In this way, the FI-MIND bridge is ready to be “plug-and-play” for every scenario using FIWARE lane and supported FIWARE Data Models.

### 1.3 MANUALS

#### 1.3.1 Installation

All the source code and documentation can be retrieved at:

---

<sup>3</sup>

[https://www.plm.automation.siemens.com/media/global/en/Siemens\\_MindSphere\\_Whitepaper\\_tcm27-9395.pdf](https://www.plm.automation.siemens.com/media/global/en/Siemens_MindSphere_Whitepaper_tcm27-9395.pdf)

<https://github.com/Engineering-Research-and-Development/fi-mind-bridge>

### 1.3.2 Configuration

Configure MINDSPHERE settings in *config.properties* file:

```
client-id={mindpsphere-client-id}
client-secret={mindpsphere-client-secret}
tenant={mindpsphere-tenant}
```

### 1.3.3 Usage

Generate WAR file executing into project root folder:

```
mvn clean install
```

Deploy WAR file (generated into target folder) as usual in your web server.

### SERVICE REQUESTS

The FI-MIND is able to receive NGSI data in both standard and normalized data format. Moreover, the bridge defines services for receiving data from a REST Client (generic machine-to-machine communication) and for processing notifications from the Orion Context Broker.

The following example describes a sample request sent by an external component to the bridge using the standard *DeviceModel* defined by the FIWARE Data Models.

```
curl -X POST \
  http://{server-url}:{server-port}/fimind/webapi/deviceModel \
  -H 'accept: application/json' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "id": "board1",
    "type": "deviceModel",
    "manufacturerName": "Sensirion",
    "modelName": "SHT10",
    "brandName": "Sensirion",
    "category": "soilBoard",
    "controlledProperty": ["temperature", "humidity"],
    "supportedUnits": ["C", "%"]
  }'
```

In the similar way, the bridge is able to process the *DeviceModel* described using the normalized way.

```
curl -X POST \
  http://{server-url}:{server-port}/fimind/webapi/deviceNormalized \
  -H 'accept: application/json' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -d '{
    "id": "board1",
    "type": "device",
    "category": {
      "value": "soilBoard"
    },
    "location": {
      "type": "Point",
      "coordinates": [125.6, 10.1]
    },
    "controlledProperty": {
      "value": ["temperature", "humidity"]
    },
    "value": {
      "value": "t=19.0;h=59",
      "metadata": {
        "timeInstant": {
          "type": "DateTime",
          "value": "2019-04-01T18:51:28.628Z"
        }
      }
    }
  }'
```

Furthermore, the FI-MIND can be subscribed to the Orion Context Broker in order to receive and process notifications about a supported FIWARE Data Model.

```
curl -X POST \
  http://{ocb-server-url}:{ocb-server-port}/v2/subscriptions \
  -H 'content-type: application/json' \
  -H 'fiware-service: {FIWARE_SERVICE}' \
  -H 'fiware-servicepath: {FIWARE_SERVICE_PATH}' \
  -d '{
    "description": "Device Test Sub",
    "subject": {
      "entities": [
        {
          "idPattern": ".*",
          "type": "Device"
        }
      ],
      "condition": {
        "attrs": [
        ]
      }
    },
    "notification": {
      "http": {

```



```
        "url": "http://{server-url}:{server-port}/fimind/webapi/fiware-  
notification"  
      },  
      "attrs": [  
      ],  
      "metadata": ["dateCreated", "dateModified"]  
    },  
    "throttling": 5  
  }'  
}
```

#### 1.3.4 Licensing

The FI-MIND bridge is released under the GNU Affero General Public License v3.0.

# Annex G. Factsheet for MIDIH MindSphere Custom Agent and MASAI MDP Connector for MIDIH

## 1 MIDIH MindSphere Custom Agent and MASAI MDP Connector for MIDIH

---

### 1.1 INTRODUCTION

**Cross-Domain Data Platform (CDP)**<sup>4</sup> is Atos' open, interoperable and standard cross-domain solution. It allows integrating and collecting information from sensors and other data sources, normalizing that data and processing it to provide aggregated and intelligent views of raw data to support the decision making. CDP is powered by **FIWARE** technology, and so it relies on NGSI standard (Next Generation Service Interface) for the exchange of context information.

Some of the main functionalities from CDP are:

- Distribute the normalized data to the diverse vertical services connected to the platform (air quality, traffic management, lighting, etc.)
- Real time data analysis, supporting the decision-making process.
- Visualization of data in a graphical and visual way through maps, graphics, alarms, etc.
- Data provisioning for developing on services from them.
- Secured access control to data and services.

Considering the previous capabilities offered by CDP, in the context of MIDIH, it has been developed a specific a CDP connector to make them accessible to any MIDIH component. This connector called “MASAI MDP Connector” has been added MASAI’s capabilities. Hence, to get benefit from the use of CDP, it should be done using MASAI as a necessary intermediate component. To demonstrate the use of CDP, it will be presented using some new characteristics developed on MASAI, like devices monitoring or raising alarms through CDP.

Currently the CDP has a specific instance for Smart Cities called Urban Data Platform (UDP) and a new one called Manufacturing Data Platform (MDP) created in order of supporting the new MASAI capabilities.

### 1.2 TECHNICAL UPDATES

#### 1.2.1 Overview

MASAI component has already been deeply presented in deliverable D4.3 - “MIDIH D4.3 Open CPS IOT Components”. The main characteristics and functionalities remain similar, although the component has been enhanced during the second iteration of T4.3 by the development of new functionalities like the monitoring of devices and the development of a CDP connector, both new features are to be explained during the next chapters.

---

<sup>4</sup><http://booklet.atosresearch.eu/node/1990>

### 1.2.2 Monitoring of Devices

The aim of this functionality is to control that some equipment in a network is always connected and if not, it allows the user to react. In manufacturing domains, there exist some critical equipment that it must be always up and running inside a network and when they are not connected, due to several reason, the can cause other systems failures. For example, when two or more external systems are exchanging information and one of them is down, the process of exchanging information will fail. Other example can be applicable to devices, in this case it can be considered a sensor that is continuously sending information. If the network connectivity of the sensor or device is not correct, the information send by the device will not arrive at its consumer. That is the reasoning behind the development of this new feature for MASAI. In the next paragraphs it is presented the work done.

In order to let MASAI that a new device will be monitored, when registering a new device, a new attribute will have to be included: a static attribute including the IP of the device. See figure below.

```

{"device_id": "device1",
 "entity_name": "Raspberry1",
 "entity_type": "Raspberry",
 "protocol": "MQTT",
 "timezone": "Europe/Madrid",
 "attributes": [
  {
    "name": "temperature",
    "type": "double"
  },
  {
    "name": "humidity",
    "type": "double"
  },
  {
    "name": "analog_temp",
    "type": "double"
  },
  {
    "name": "Device",
    "type": "string"
  },
  {
    "name": "date_time",
    "type": "string"
  }
],
 "static_attributes": [{
  "name": "static_IP",
  "type": "string",
  "value": "130.206.121.30"
}],
 "commands": [
  {
    "name": "ping",
    "type": "command"
  }
]
}

```

Once the device is register in MASAI, a python script has to be launched to start the monitoring process. This script has a high performance and efficient engine, providing a “heartbeat” functionality, capable to execute multi thread processes. This allows the system to perform multiples heartbeats at the same time, obtaining the result in a very short period of time. But before the monitoring process starts, the script queries MASAI to see which devices are registered to identify which IP’s have to be monitored. Once this information is available, the script is capable to start the monitoring process. See screenshot below:

```

-----Request time=0.000149011611938-----CHECKING IPS
2019-12-17 12:31:40,058 - system_ping - DEBUG - PID: 911, Process Name: MainProcessDEVICES SPL 0 ['device2']
2019-12-17 12:31:40,059 - system_ping - DEBUG - PID: 911, Process Name: MainProcessADDR SPL 0 ['130.206.121.30']
2019-12-17 12:31:40,060 - system_ping - DEBUG - PID: 911, Process Name: MainProcess----- IP OK -----
2019-12-17 12:31:40,166 - system_ping - DEBUG - PID: 911, Process Name: MainProcess 130.206.121.30
2019-12-17 12:31:40,167 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:31:40", "mensaje": "OK"}', 0, False)
2019-12-17 12:31:40,168 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:31:40", "mensaje": "OK"}', 0, False)]
-----Execution time=0.121779918671-----

-----Request time=0.000144958496094-----CHECKING IPS
2019-12-17 12:31:50,180 - system_ping - DEBUG - PID: 911, Process Name: MainProcessDEVICES SPL 0 ['device2']
2019-12-17 12:31:50,182 - system_ping - DEBUG - PID: 911, Process Name: MainProcessADDR SPL 0 ['130.206.121.30']
2019-12-17 12:31:50,306 - system_ping - DEBUG - PID: 911, Process Name: MainProcess----- IP OK -----
2019-12-17 12:31:50,307 - system_ping - DEBUG - PID: 911, Process Name: MainProcess 130.206.121.30
2019-12-17 12:31:50,308 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:31:50", "mensaje": "OK"}', 0, False)
2019-12-17 12:31:50,309 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:31:50", "mensaje": "OK"}', 0, False)]
-----Execution time=0.203529119492-----

-----Request time=0.000143766403198-----CHECKING IPS
2019-12-17 12:32:00,385 - system_ping - DEBUG - PID: 911, Process Name: MainProcessDEVICES SPL 0 ['device2']
2019-12-17 12:32:00,387 - system_ping - DEBUG - PID: 911, Process Name: MainProcessADDR SPL 0 ['130.206.121.30']
2019-12-17 12:32:00,725 - system_ping - DEBUG - PID: 911, Process Name: MainProcess----- IP OK -----
2019-12-17 12:32:00,726 - system_ping - DEBUG - PID: 911, Process Name: MainProcess 130.206.121.30
2019-12-17 12:32:00,728 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:32:00", "mensaje": "OK"}', 0, False)
2019-12-17 12:32:00,729 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:32:00", "mensaje": "OK"}', 0, False)]
-----Execution time=0.35558197708-----

```

In the figure presented above, where the monitoring period is defined by the user, all the IPs being monitored are responding correctly. But, in case some of the device's connectivity gets down, see screenshot below:

```

-----Request time=0.000144958496094-----CHECKING IPS
2019-12-17 12:50:10,825 - system_ping - DEBUG - PID: 1030, Process Name: MainProcessDEVICES SPL 0 ['device2']
2019-12-17 12:50:10,826 - system_ping - DEBUG - PID: 1030, Process Name: MainProcessADDR SPL 0 ['130.206.121.30']
2019-12-17 12:50:10,827 - system_ping - DEBUG - PID: 1030, Process Name: MainProcess----- IP OK -----
2019-12-17 12:50:10,844 - system_ping - DEBUG - PID: 1030, Process Name: MainProcess 130.206.121.30
2019-12-17 12:50:10,845 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:50:10", "mensaje": "OK"}', 0, False)
2019-12-17 12:50:10,846 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:50:10", "mensaje": "OK"}', 0, False)]
-----Execution time=0.13793929962-----

-----Request time=0.000146150588988-----CHECKING IPS
2019-12-17 12:50:20,959 - system_ping - DEBUG - PID: 1030, Process Name: MainProcessDEVICES SPL 0 ['device2']
2019-12-17 12:50:20,961 - system_ping - DEBUG - PID: 1030, Process Name: MainProcessADDR SPL 0 ['130.206.121.30']
2019-12-17 12:50:20,961 - system_ping - DEBUG - PID: 1030, Process Name: MainProcess----- IP OK -----
2019-12-17 12:50:21,083 - system_ping - DEBUG - PID: 1030, Process Name: MainProcess 130.206.121.30
2019-12-17 12:50:21,084 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:50:21", "mensaje": "OK"}', 0, False)
2019-12-17 12:50:21,086 - system_ping - DEBUG - (['device2/sinkData', '{"Device": "device2", "date_time": "17/12/2019, 12:50:21", "mensaje": "OK"}', 0, False)]
-----Execution time=0.13361120224-----

-----Request time=0.000144004621777-----CHECKING IPS
2019-12-17 12:50:31,098 - system_ping - DEBUG - PID: 1030, Process Name: MainProcessDEVICES SPL 0 ['device2']
2019-12-17 12:50:31,099 - system_ping - DEBUG - PID: 1030, Process Name: MainProcessADDR SPL 0 ['130.206.121.30']
2019-12-17 12:50:32,104 - system_ping - DEBUG - PID: 1030, Process Name: MainProcess----- IP NOT OK -----
2019-12-17 12:50:32,105 - system_ping - DEBUG - PID: 1030, Process Name: MainProcess 130.206.121.30
2019-12-17 12:50:32,107 - system_ping - DEBUG - (['ATOS/device/attributes', '{"Device": "device2", "date_time": "17/12/2019, 12:50:32", "mensaje": "Device not responding. Check equipment!"', 0, False)
2019-12-17 12:50:32,108 - system_ping - DEBUG - (['ATOS/device/attributes', '{"Device": "device2", "date_time": "17/12/2019, 12:50:32", "mensaje": "Device not responding. Check equipment!"', 0, False)]
-----Execution time=1.0224802991-----

```

When the monitoring system identifies that a device is not responding correctly, it informs MASAI by putting a message into a specific topic created to support this functionality. At this point, MASAI complex event processing engine is capable to get the incoming event and to raise a new one to inform a consumer to react to this situation. An example of this will be shown in the next chapter, i.e. Communication with CDP. In the figure below it is depicted how MASAI trace events.

```

2019-12-17 13:03:41.930 INFO INFO --- [nio-9080-exec-1] c.o.cephus.ccp.superEventProcessor : EventIn: Event(type='AvisoMonitoring', values=[protocol=MQTT, date_time=17/12/2019, 13:03:40, Device=device2, mensaje=Device not responding. Check equipment!, id=AvisoMonitoring])
2019-12-17 13:03:41.978 WARN INFO --- [nio-9080-exec-2] c.o.cephus.ccp.superEventProcessor : notifyConsumer: request: [mqtt4_subscription_id=ca5839d-1809-4683-b0ad-5a8c-66507a / hostIp=localhost:9080]
2019-12-17 13:03:41.987 INFO INFO --- [nio-9080-exec-1] c.o.cephus.ccp.EventSinkListener : EventOut: Monitoring / date_time:17/12/2019, 13:03:40 / Device=device2 / mensaje=Device not responding. Check equipment! / id:null from insert id= Monitoring select mensaje, Device, date_time from AvisoMonitoring

```

As a matter of reference to the python script, some of the libraries used are:

- multiprocessing (<https://docs.python.org/2/library/multiprocessing.html>).
- threading (<https://docs.python.org/3/library/threading.html>).
- requests (<https://2.python-requests.org/en/latest/user/quickstart/>).
- paho-mqtt (<https://pypi.org/project/paho-mqtt/>).

**1.2.3 Communication with CDP**

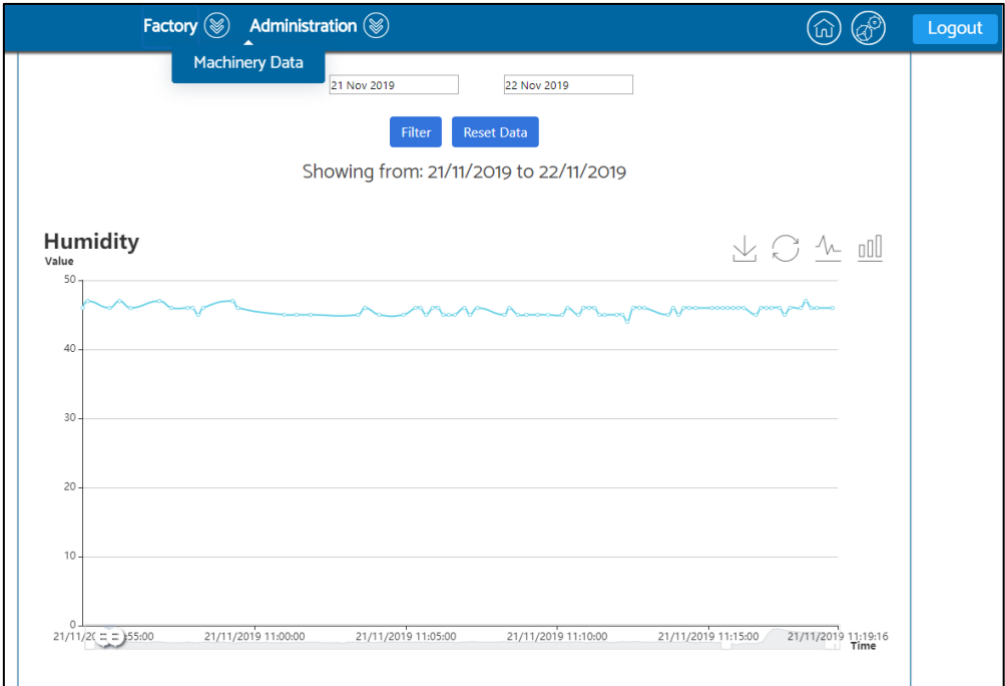
The CDP offers a dedicated instance for MASAI, called Manufacturing Data Platform (MDP). See a screenshot of the login page below:



**Figure 29 – Manufacturing Data Platform login**

As it has been previously commented, both MDP and MASAI are FIWARE based solutions, which means that they can they both use the NGSI standard to describe the entities they are managing. Having components capable to interpret the NGSI standard simplifies notably the integration activities. In this manner, to achieve the integration between both components, UDP offers their specific instance of the Orion Context Broker (OCB), that will act as the link between MASAI and MDP. Consequently, MASAI will play de role of the data producer, MDP the role of the data consumer and OCB will provide the brokering layer.

MDP offers data visualisation capabilities in different graph formats, that can be selected by the user. In the figure below it is presented how incoming data MASAI is shown to the user:



**Figure 30 – Manufacturing Data Platform Monitoring**

Additionally, and developed to match MASAI demands, there have been developed some functionalities to complement MASAI's features:

1. A user interface to display alerts or alarms coming from MASAI: MASAI component has the capacity to generate alerts or alarms by itself. This is done through the definition and appliance of business rules on the data managed by MASAI. For example: to define a rule to generate an alert when a determined measurement reaches a threshold. The next figure represents how business rules are defined in MASAI:

```

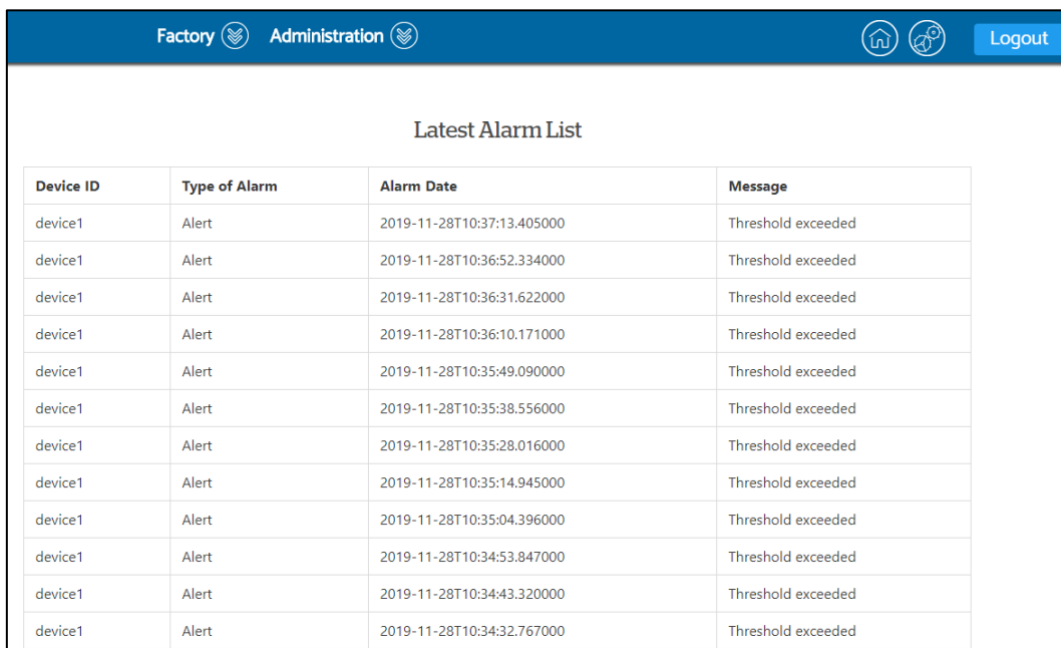
{
  "in": [
    {
      "id": "Raspberry1",
      "type": "Raspberry",
      "providers": [
        "http://localhost:8081"
      ],
      "attributes": [
        { "name": "temperature", "type": "double" },
        { "name": "humidity", "type": "double" },
        { "name": "analog_temp", "type": "double" },
        { "name": "Device", "type": "string" }
      ]
    }
  ],
  "out": [
    {
      "id": "Alert1",
      "type": "Alert",
      "attributes": [
        { "name": "temperature", "type": "double" },
        { "name": "humidity", "type": "double" },
        { "name": "analog_temp", "type": "double" },
        { "name": "Device", "type": "string" }
      ]
    }
  ],
  "statements": [
    "INSERT INTO Alert SELECT temperature, humidity, analog_temp, Device FROM Raspberry Where temperature > 25"
  ]
}

```

To create business rules in MASAI, it is necessary to provide the following information:

- a) "in": represents the incoming data. Incoming data arrives into MASAI in form of events. To describe an event, it is used an object type and one or more attributes.
- b) "out": represents the outgoing events. They are generated by MASAI when the incoming data matches the condition defined in the statements part. Each outgoing event is composed by an object type and one or more attributes.
- c) "statements": it is the business rule defined, it handles the incoming events to generate the output events when the condition is matched. Several statements can be defined in MASAI.

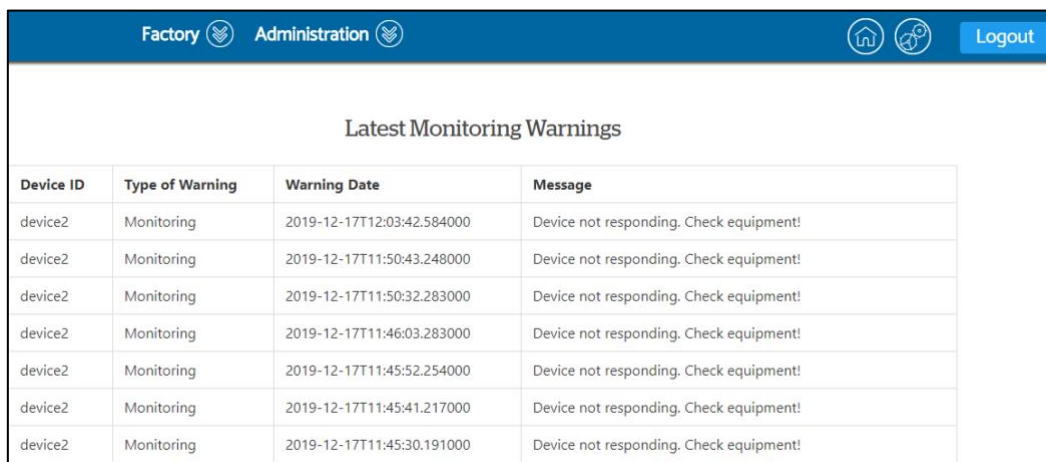
Considering the description given above about how to define business rules in MASAI, the process how alerts are generated in MASAI and displayed in the MDP is as follow: i) incoming events start feeding MASAI with data coming from the data sources; ii) those incoming events are analyzed, and the ones that match the business condition defined in MASAI will be used to generate new output events, and iii) output events are sent to the MDP through a dedicated instance of the Orion Context Broker. Once they arrive at the MDP, they are finally displayed on a dedicated page with the aim of informing the final user. An example is given below:



Device ID	Type of Alarm	Alarm Date	Message
device1	Alert	2019-11-28T10:37:13.405000	Threshold exceeded
device1	Alert	2019-11-28T10:36:52.334000	Threshold exceeded
device1	Alert	2019-11-28T10:36:31.622000	Threshold exceeded
device1	Alert	2019-11-28T10:36:10.171000	Threshold exceeded
device1	Alert	2019-11-28T10:35:49.090000	Threshold exceeded
device1	Alert	2019-11-28T10:35:38.556000	Threshold exceeded
device1	Alert	2019-11-28T10:35:28.016000	Threshold exceeded
device1	Alert	2019-11-28T10:35:14.945000	Threshold exceeded
device1	Alert	2019-11-28T10:35:04.396000	Threshold exceeded
device1	Alert	2019-11-28T10:34:53.847000	Threshold exceeded
device1	Alert	2019-11-28T10:34:43.320000	Threshold exceeded
device1	Alert	2019-11-28T10:34:32.767000	Threshold exceeded

**Figure 31 – Manufacturing Data Platform alarms visualization**

2. A user interface to display monitoring functionalities announcements: linking with what it was presented in the previous chapter, there is a specific page to inform the user about devices monitoring issues. MASAI generates the events that are displayed in MDP. An example is given below:



Device ID	Type of Warning	Warning Date	Message
device2	Monitoring	2019-12-17T12:03:42.584000	Device not responding. Check equipment!
device2	Monitoring	2019-12-17T11:50:43.248000	Device not responding. Check equipment!
device2	Monitoring	2019-12-17T11:50:32.283000	Device not responding. Check equipment!
device2	Monitoring	2019-12-17T11:46:03.283000	Device not responding. Check equipment!
device2	Monitoring	2019-12-17T11:45:52.254000	Device not responding. Check equipment!
device2	Monitoring	2019-12-17T11:45:41.217000	Device not responding. Check equipment!
device2	Monitoring	2019-12-17T11:45:30.191000	Device not responding. Check equipment!

**Figure 32 – Manufacturing Data Platform Ping Monitoring**

# Annex H. Factsheet for FIWARE ROS integration via OpenMTC generic enabler

## 1 FIWARE ROS Integration via OpenMTC generic enabler

### 1.1 INTRODUCTION

Within this section an introduction to OpenMTC, as an M2M/IoT platform, and the Robot Operating System (ROS) is given. Both OpenMTC and ROS are quite complex platforms. While this section provides certain details on OpenMTC – to allow understanding the integration efforts of ROS, described in the next section – further details on the Robot Operating System are left out in this section, to cut down the introduction. Of course, within the next section certain details of used interfaces, from ROS, for the integration effort, are described. Yet, the focus is only on data communication interfaces.

#### 1.1.1 Short Introduction to OpenMTC

OpenMTC (<https://www.openmtc.org/>) is a reference implementation of oneM2M (<http://onem2m.org/>). Using OpenMTC, users are enabled to create their own machine-to-machine M2M or IoT applications, setups, and deployments. OpenMTC, as an IoT/M2M middleware, allows interconnecting hardware sensors and actuators with software applications. This, for example, allows to aggregate data from various industrial domains, annotate the data with metadata and combine it with an application for short term centralized control -- e.g. to activate multiple signal lights (as actuators) whenever certain levels of sensed temperature and air pollution is reached. Furthermore, aggregated data can be accessed by storage applications for long term data storage. This provides the foundation for data analysis applications, for long term data analysis.

OpenMTC is implemented in Python. Its various components can be deployed directly into a Linux/Unix-based environment (single or multiple host(s)) or using a container approach (mainly Docker is prepared) for virtualized environments (i.e. cloud infrastructures).

#### 1.1.2 Introducing most relevant OpenMTC components and their functionality

OpenMTC provides various functionality and concepts that allow to create M2M systems. The main important entities of an OpenMTC-based M2M system are:

1. Interworking Proxy Entity (IPE)
2. Application Entity (AE)
3. Gateway (GW)
4. Backend (BE)

For most scenarios of OpenMTC, hardware end devices do not directly support oneM2M. To allow such a device to connect to oneM2M-based infrastructure specific entities are foreseen: Interworking Proxy Entity (IPE). These entities allow to map/translate from non-oneM2M data models into oneM2M data models and vice versa (e.g. for non-oneM2M actuators).

The different hardware end devices that send or receive oneM2M data need to be connected to one another. Using OpenMTC Gateways various oneM2M data sources and sinks can be connected. It is possible to create chains of Gateways, reflecting the actual on-site topology.



Gateways not only provide means to transfer a stream of data within a oneM2M-based network but also provide functionality to store data for certain amounts of time or events.

To be able to provide control logics within an OpenMTC-based system, Application Entities (AEs) are used. This makes it possible to subscribe to certain data changes within a specific Gateway (e.g. collecting new temperature values from a connected temperature sensor) to execute certain logics based on these changes (e.g. to increase the input variable of a fan device from 0 to 1). In addition, an Application Entity can be used to visualize sensed data or the current state of certain actuators onto a dashboard, for visual monitoring. Finally, using an Application Entity it is possible to store all collected data values into a database system.

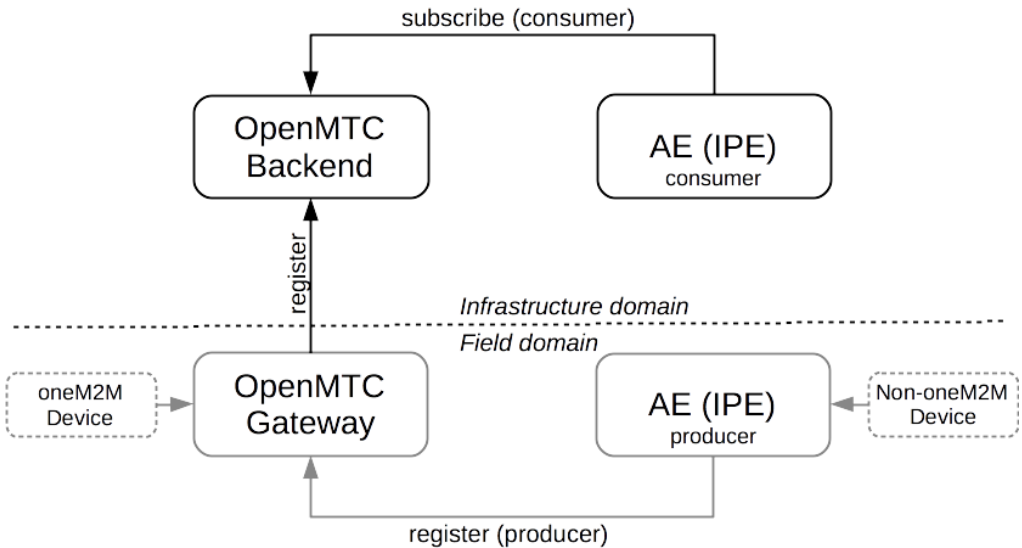
In oneM2M, different domains are distinguished:

1. Field Domain
2. Infrastructure Domain.

Within the Field Domain, the actual end devices and IPE nodes are placed, and directly interconnected via GW nodes. It is also possible to provide Application Entities for Field-local applications (for example lowering path latency and by this lowering reaction times).

The Infrastructure Domain allows to provide a centralized point, where all data computation infrastructure can be placed. This allows to separate high-performance data computation from the actual field devices. This enables for long-term data analysis using data collected in the field domain. The connecting point between AE nodes that are placed into the infrastructure domain with GW nodes from the field domain is provided through a Backend node.

**Error! Reference source not found.** depicts the domain separation:



**Figure 33 – Separation of Domains in OpenMTC**

**1.1.3 Introduction to Robot Operating System**

The Robot Operating System (ROS) is a framework focused on robot applications. This framework provides tools, libraries, and further concepts to simplify the process of getting a robot executing the tasks it expected to do.

The foundation of this framework is a common publish/subscribe-based message passing middleware. With ROS, a robot is seen as a distributed system of interacting subsystems. Each of the subsystems can interact with each other via the message passing middleware. This also allows for remote procedure calls (RPCs), to trigger code execution remotely.

ROS not only involves a software framework but also a common ground for a ROS community. Functionality provided for specific robot hardware can be shared within the community, based on commonly specified interfaces. This allows constructing new robot applications based on various parts already provided by other members of the community.

Some of the tackled problems specific to robots involve localization and navigation, as also diagnostics and pose estimation. Specific libraries and tools have evolved over the past that tackle these problems in different ways. This makes it possible to re-use functionality from others, when developing new robot applications.

#### 1.1.4 Introduction to Sawyer robot

Various organizations provide different robot-related products, using ROS as a platform. Rethink Robotics (<https://www.rethinkrobotics.com>) is one of them. They are a provider of out-of-the-box robot solutions. For example, their "Baxter" robot (<https://web.archive.org/web/20140725012438/https://www.rethinkrobotics.com/products/baxter/>) was a prototypical robot solution for manufacturing environments. For this robot also a fully-functional simulator was provided. The development and support for this product was cancelled. The new product line of Rethink Robotics is named "Sawyer" (<https://www.rethinkrobotics.com/sawyer>). Like the "Baxter" robot a simulator is provided for "Sawyer" too.

Using ROS, this simulator can be accessed in the same way as its real-world counterpart. All components within the robot can be used as sensor inputs (from) the robot. For example, each joint of a robot arm can provide data, like joint movement velocity, via a dedicated (so called) "ROS topic". Whenever an arm movement is executed, each joint of that arm will provide data update notifications to all subscribers of that topic.

## 1.2 FUNCTIONAL DESCRIPTION

### 1.2.1 OpenMTC as a FIWARE Generic Enabler

OpenMTC was prepared to allow integration into FIWARE. Therefore, OpenMTC is an Incubated Generic Enabler.

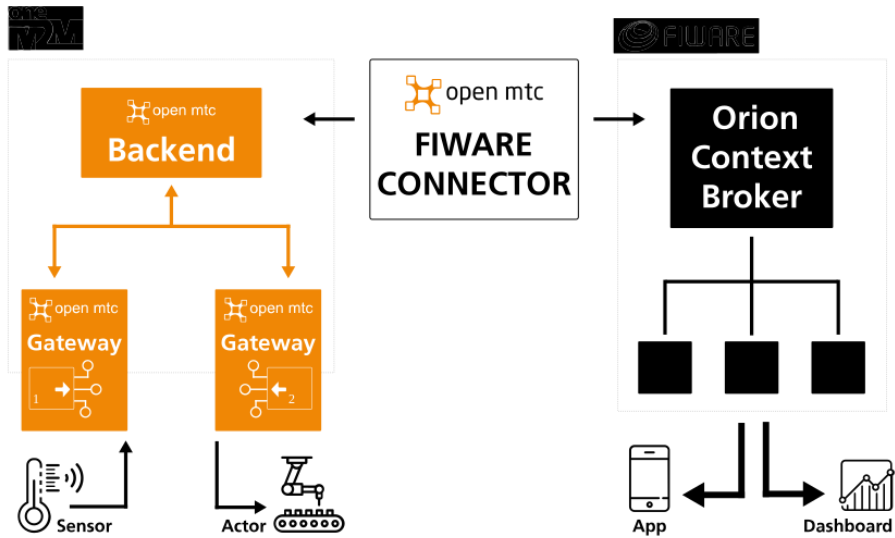


Figure 34 – OpenMTC FIWARE integration

As depicted in Figure 34, this allows data that is collected in an oneM2M-based environment to be provided as context information to the Orion Context Broker. The integration allows for bidirectional data flows. Therefore, context information available via the Orion Context Broker can flow "back" into the oneM2M-based environment. By this oneM2M end devices for actuation are usable too.

### 1.2.2 Integration of ROS into OpenMTC

ROS provides a dedicated language binding for Python: "rospy" (<https://wiki.ros.org/rospy>). This allows to directly subscribe for data change notifications that are published into specific ROS topics, via Python.

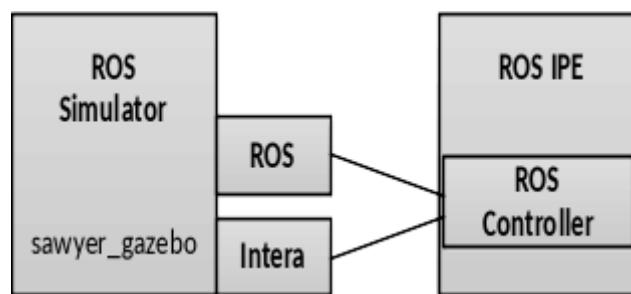


Figure 35 – ROS IPE's interface to Sawyer Robot Simulator

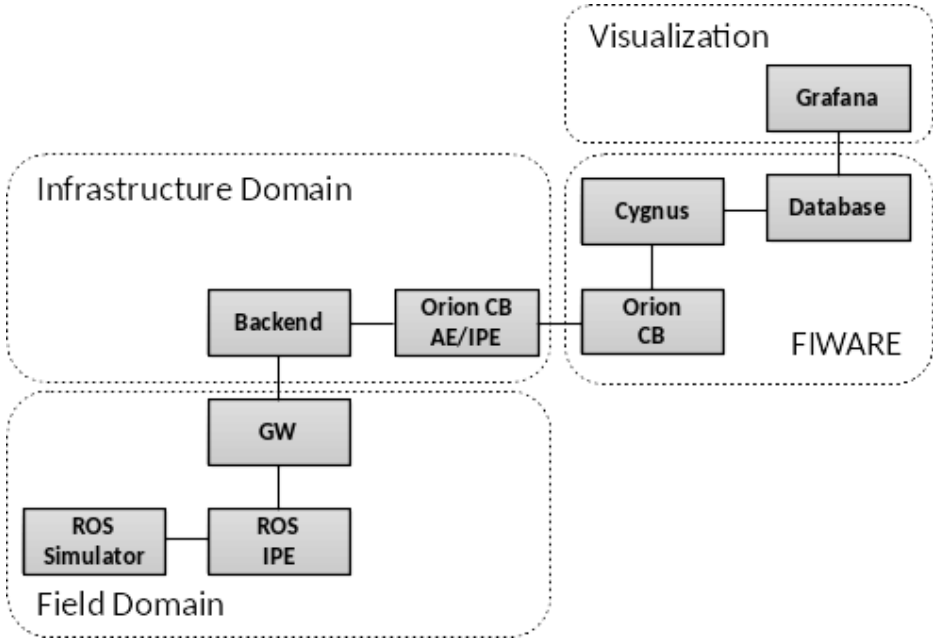
Figure 35 presents an implementation detail regarding the main interfaces from OpenMTC to the Sawyer Robot Simulator. Here, the ROS Interworking Proxy Entity (ROS IPE) provides a sub component "ROS Controller" that allows to access data update notifications from the robot simulator via ROS' Python binding. When new data updates are received, the received data will be translated into oneM2M's data model – and provided with additional meta data, like timestamps, etc.

Also seen in the figure, to allow that the simulator will constantly provide data updates, a controlling part via the IPE is provided. The controlling part will make the robot's arm "wobble". Here, the "intera" interface (provided by Rethink Robotics, [https://sdk.rethinkrobotics.com/intera/Robot\\_Interface](https://sdk.rethinkrobotics.com/intera/Robot_Interface)) is leveraged.

In summary, OpenMTC's ROS IPE allows to subscribe for data update notifications from the Sawyer robot simulator. An additional component within the IPE is provided that controls the Sawyer robot simulator in a way that constant data update notifications will be visible.

**1.2.3 Accessing robot data via OpenMTC Generic Enabler for visualization**

Figure 36 **Error! Reference source not found.** depicts all components that are used to integrate ROS to OpenMTC, as a FIWARE generic enabler, and visualize the data using Grafana. Volatile data from the robot's arm joints is collected via OpenMTC's "ROS IPE" and provided to an OpenMTC Gateway. By this data from ROS is transformed into the oneM2M data model. As depicted in above mentioned figure, all this happens in the Field Domain.



**Figure 36 – Overall view on all used components**

Also identifiable in the figure is the usage of OpenMTC's Backend. For this experiment the Backend is used to provide a centralized access to the data from the ROS simulator. This means that the data from the field domain is made accessible into an Infrastructure Domain.

Although for this experiment a robot simulator was used that cannot change location, a real-world robot may be able to move between locations. Therefore, the Backend provides an anchor point for all FIWARE components. This allows handling data path routing within OpenMTC and allows for non-interruptive data communication channels.

Next, an OpenMTC Orion Context Broker Application Entity (Orion CB AE) is provided, which acts as the connecting interface between FIWARE and OpenMTC. Via the Orion Context Broker and Cygnus, the data is persisted into a database. The visualization is provided through Grafana that accesses the persisted data directly from the database.

## 1.3 MANUALS

### 1.3.1 Installation and Preparation of ROS

In the following, the main preparations that are needed for ROS are described. As a foundation, Ubuntu 16.04 is expected as OS environment. Additionally, the description will mainly reference existing installation guides on the Internet, but additional steps needed to be executed, mainly:

- sudo apt-get install python-pip
- pip install rospymessageconverter

Preparation of ROS Kinetic and the Workstation Setup:

- <http://wiki.ros.org/kinetic/Installation/Ubuntu>
- [https://sdk.rethinkrobotics.com/intera/Workstation\\_Setup](https://sdk.rethinkrobotics.com/intera/Workstation_Setup)

Preparation of Sawyer Robot Simulator:

- [https://sdk.rethinkrobotics.com/intera/Workstation\\_Setup#Configure\\_Robot\\_Communication.2FROS\\_Workspace](https://sdk.rethinkrobotics.com/intera/Workstation_Setup#Configure_Robot_Communication.2FROS_Workspace)
  - o Especially note the adjustments as described in “Verify 'ros\_version' field”

Preparations of Gazebo-specifics for Robot Simulator:

- [https://sdk.rethinkrobotics.com/intera/Gazebo\\_Tutorial](https://sdk.rethinkrobotics.com/intera/Gazebo_Tutorial)
  - o Note: if observing segmentation faults, try setting following environment variable: SVGA\_VGPU10=0

### 1.3.2 Checking for a correctly working ROS setup

The following assumes that ROS workbench was installed into: `~/ros_ws/` directory:

- `cd ~/ros_ws/`
- `./intera.sh sim`
- `roslaunch sawyer_gazebo sawyer_world.launch`

Simple test for properly prepared ROS topics:

- `source /opt/ros/kinetic/setup.bash`
- `rostopic list`
- `rostopic echo /robot/joint_states`

Running a simple default example:

- `roslaunch intera_examples joint_torque_springs.py`

### 1.3.3 Preparations for and Installation of OpenMTC

- `sudo apt install docker.io libev-dev python-dev gcc make automake`
- `cd ~`
- `git clone https://github.com/OpenMTC/OpenMTC.git`

#### 1.3.4 Running OpenMTC Gateway

- cd ~/OpenMTC
- openmtc-gevent/run-gateway

#### 1.3.5 Running OpenMTC Backend

- cd ~/OpenMTC
- openmtc-gevent/run-backend

#### 1.3.6 Running OpenMTC Orion CB AE<sup>5</sup>

- cd ~/OpenMTC
- ./apps/orion-context-broker

#### 1.3.7 Preparations for and Installation of OpenMTC ROS IPE

- sudo apt install docker.io libev-dev python-dev gcc make automake
- cd ~
- git clone <https://github.com/OpenMTC/OpenMTC.git> OpenMTC-rosipe
- cd ~/OpenMTC-rosipe/
- git checkout legacy-python2

#### 1.3.8 Running OpenMTC's ROS IPE

- export  
PYTHONPATH=\$PYTHONPATH:\$HOME/ros\_ws/install/lib/python2.7/dist-packages/
- source /opt/ros/kinetic/setup.bash
- cd ~/OpenMTC-rosipe/ipes/
- ./ros-ipe

#### 1.3.9 Configuration

Additional Configurations are not needed or provided.

---

<sup>5</sup> A fully-out-of-the-box, simplified setup can be created using docker-compose, see: <https://github.com/OpenMTC/OpenMTC/tree/master/apps/OrionContextBroker>.