# D3.2 Design of the monitoring and control system for each use case – Public Version

30/06/2022

Authors:
Edgar Rubión, EURECAT
Marcel Ortiz, EURECAT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

# Technical References

| | |
|---|---|
| **Project Acronym** | iWAYS |
| **Project Title** | Innovative Water recoverY Solutions through recycling of heat, materials and water across multiple sectors |
| **Project Coordinator** | Prof. Luca Montorsi – University of Modena and Reggio Emilia |
| **Scientific and Technical Director** | Prof. Hussam Jouhara – Brunel University London |
| **Project Duration** | 48 |

| | |
|---|---|
| **Deliverable No.** | D3.2 |
| **Dissemination level** [1] | PU |
| **Work Package** | 3 |
| **Task** | 3.1 |
| **Lead beneficiary** | EURECAT |
| **Contributing beneficiary(ies)** | EURECAT, BUL, ITC-AICE, CON, ALUFLUOR, TTI, ICRA, SIMAM |
| **Due date of deliverable** | 31/05/2022 |
| **Actual submission date** | 30/06/2022 |

[1] PU = Public
PP = Restricted to other programme participants (including the Commission Services)
RE = Restricted to a group specified by the consortium (including the Commission Services)
CO = Confidential, only for members of the consortium (including the Commission Services)

# Document history

| V | Date | Beneficiary | Author |
|---|---|---|---|
| V0.1 | 03-06-2022 | EUT | Edgar Rubión |
| V0.2 | 10-06-2022 | EUT | Edgar Rubión<br>Marcel Ortiz |
| V0.3 | 29-06-2022 | BUL | Hussam Jouhara<br>Bertrand Delpech |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

| | | | |
|---|---|---|---|
| V1 | 30-06-2022 | EUT | Edgar Rubión |

# Summary

## Summary of Deliverable

Deliverable 3.2 “Design of the monitoring system for the characterization of the recovered water and emissions – Public Version” deals with the activities carried out in the iWAYS project under Task 3.1 “Monitoring Solutions Design” Work Package 3 “Design of sustainable water sourcing as replacement for industrial water” and Task 3.1 “Monitoring Solutions Design” during the fourteen months of the Project (M04-M18).

The goal of Deliverable 3.2 is to design a flexible monitoring and control platform to support end-to-end process of data collection, harmonization, processing and visualization for evidence-based decision making on three industrial cases, Alufluor, Concorde and Tubacex. The design and specification of the IoT platform is based on the process and technological specification (WP2), DSS requirements (WP3) and initial designs of iWAYS innovative technologies (WP4 and WP5), providing a flexible IoT platform that can be adapted to each individual case.

Deliverable 3.2 presents the outputs of the non-confidential activities carried out in Task 3.1, which are:

- Definition of the functional and non-functional requirements of the IoT platform, identifying 34 requirements.
- Design of a IoT platform by using a modular structure (communication module, persistence module, management module and analytics module) based on services that allow for dynamically adding or removing modules without the need to re-implement or redefine any of the existing ones.
- Select the most appropriate open-source information technologies to build the IoT platform such as Mosquitto[1], Timescale[2] and Grafana[3].
- Define the interfaces, protocols and data models to exchange data with the IoT platform, both in real time and in batch mode, based on IoT and IIoT standards such as REST and MQTT.
- Define the E-R schema to manage all the metadata used by the IoT platform for its correct functioning, such as the accepted data formats, the physical resources that can publish information, among others.

Finally, it is important to note that this document will evolve in the future along with the evolution of the iWAYS designs. Then, a new version of this document will be presented that will also include a sketch of the visualization for each case.

[1] https://mosquitto.org

[2] https://www.timescale.com

[3] https://grafana.com

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

## Disclaimer

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

## Table of contents

## Table of tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

## Table of figures

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

# 1 Introduction

The aim of the iWAYS Project is the reuse of industrial waters in combination with heat and/or substances in energy and resource intensive industries by developing an innovative Heat Pipe-Based Condensing Economiser and other integrated water-smart strategies. The iWAYS project commenced in December 2021 (M01) and this report, Deliverable 3.2 “Design of the monitoring and control system for each use case – Public Version” (D3.2), is the first from the project’s Work Package 3 (WP3) “Design of sustainable water sourcing as replacement for industrial water”.

## 1.1 The Role of the Deliverable 3.2

The goal of D3.2, which corresponds with the outcome of Task 3.1 “Monitoring Solutions Design”, is to design a flexible monitoring and control platform to support end-to-end process of data collection, harmonization, processing and visualization for evidence-based decision making on three industrial cases, Alufluor, Concorde and Tubacex.
The design and specification of the IoT platform was based on the process and technological specification (WP2), DSS requirements (WP3) and initial designs of iWAYS innovative technologies (WP4 and WP5), providing a flexible IoT platform that can be adapted to each individual case. The designed IoT platform on WP3 for the monitoring and control of industrial cases will be implemented and tested on Task 3.3 and presented on D3.3.
This document will provide detailed information on how to integrate the physical resources of the technology solutions (PLCs, DAQs…) with the digital world (IoT platform) through MQTT standard.
Finally, this deliverable will be updated in the future with the advancement of the final designs of the technologies for water and heat recovery and water treatment.

## 1.2 Approach to the Development of D3.2

To progress the development of Project Task T3.1 “Monitoring Solutions Design”, Eurecat prepared the D3.2 Table Of Content (ToC) and the description of the expected content for each section and subsection. This document was uploaded to the project repository and discussed in a Teams WP3 meeting with the WP3 Task Leaders, avoiding possible overlapping between tasks. Then, the ToC was updated with the suggestions and recommendations of WP3 partners and Technical Coordinator. Later, the document facilitated the collection of data in a standardized way.
The extraction of the necessary information to define the IoT platform was done through the review of documents available in the project's document repository, and the interaction with technology suppliers, and industrial cases. Multiples templates, both word and excel, aimed at collecting relevant information about the technologies or industrial facilities and their connectivity was exchanged with the industrial partners. All gathered information was analysed and processed by Eurecat with the aim of define the IoT platform and the communication protocols.
Monthly meetings of WP3 were held with the objective of addressing the tasks and solving problems.

## 1.3 Document Outline

In line with the requirements of the project Grant Agreement and Work Package Descriptions, this report includes:

a) a detailed description of functional and non-functional requirements based in the needs of the technology providers, and of the DSS (section 2);

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

b) a detailed description of the IoT platform, including the modules, the technologies and how to interact with it (section 3);
c) a general description of the data integration, including an example of structure of the MQTT messages (section 4);
d) the conclusions and future work related to the design of the IoT platform (section 0); and
e) a full description of the API for accessing historical data, including data models and methods (section Annex a) and b)).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

# 2 Functional and Non-functional Requirements

This section identifies the main requirements, both functional and non-functional, that must be met by the monitoring and control platform, that is, IoT platform. It is important to note that these requirements have been extracted from the information available for each industrial cases and from the interaction with technology providers and industrial partners.

All three processes involve numerous physical resources, as well as multiple physical and chemical parameters related to water quality such as TN, TP, metals, and operational parameters such as temperature, pressure and valve state. Therefore, it is necessary to create a flexible IoT platform that allows to automatically integrate all these datasets in order to extract knowledge through time series analysis. Additionally, the platform should allow real-time and batch integration through standardised interfaces in order to support third party tools such as DSS. The platform must also provide secure access to information, as well as increase data reliability through the application of signal conditioning techniques.

Table 1 summarizes both the functional and non-functional requirements of the IoT platform for the monitoring and control of the 3 processes.

*Table 1 List of functional and non-functional requirements for the IoT platform based on digital technologies*

| # | F/NF | Type | Description |
|---|---|---|---|
| [1] | F | Integration | Monitor and manage any sensors involved in the process operation. |
| [2] | F | Integration | One-way communication with the different monitoring and control equipment. |
| [4] | F | Integration | Automatic integration of plant data (sensors) involved in the operation process. |
| [5] | F | Integration | Third-party data integration and real-time and batch data consumption by third parties |
| [6] | F | Integration | Heterogeneous data transfer, use of highly generic data models to support any device, equipment, software... and type of message including numeric, Boolean, alphanumeric, categorical information... |
| [7] | F | Integration | Ability to deploy models for validation, transformation and/or pre-processing of integrated data |
| [8] | F | Integration | Provide an API to access data stored in the platform (process/resource/sensors) supporting third parties as well as analytics processes. |
| [9] | F | Integration | Provide different data export formats (CSV, JSON) to accommodate third party requirements. |
| [10] | F | Integration | Offer a data subscription channel (sensors, alerts and recommendations) to access data in real time. |
| [11] | F | Conceptualisation | Represent the different cyber-physical elements involved in the process (e.g. HPCE, PFNR, Boiler Gas…) |
| [12] | F | Conceptualisation | Define typologies of messages including their structure (e.g. message with information on the status of an asset, message with information on water quality) associated with the cyber-physical elements. |
| [13] | F | Design | Modular architecture of the platform to facilitate the addition or change of new equipment/communication protocols/sensors. |
| [14] | F | Design | Ensure the flexibility of the platform by opting for a modular design in which each component performs a highly specialised function (connectors, storage, management API, subscription API, etc.), facilitating the migration of technologies and/or implementations. |
| [15] | F | Design | Manage the processes by means of a REST API to register/unregister devices, processes, resources, and associate types of messages to the elements. |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

| | | | |
|---|---|---|---|
| [16] | F | Design | Provide methods for managing user access control to APIs, ensuring security. |
| [17] | NF | Design | Ensure platform security by securing communication channels between platform modules and securing communication channels between connectors and connected equipment (for those possible). |
| [18] | F | Design | Persist information injected from the different modules (connectors, analytics processes, management APIs, ...) through a relational database for management data and a time series database for data generated by devices, analytics processes and solution performance metrics. |
| [19] | F | Design | Collect and transmit indicators of platform performance through metrics (e.g. number of connected equipment, equipment connection/disconnection events, unknown equipment connections, among others) published on the common communication channel allowing to monitor the platform status, detect failures and bottlenecks. |
| [20] | F | Design | Provide real-time and batch analysis capabilities to exploit process information, generating alerts and/or recommendations to be incorporated into the platform. |
| [21] | NF | Design | The platform will be developed with the main objective of being deployed in the Cloud or local servers with high resources. |
| [22] | NF | Design | User-friendly graphical interface. |
| [23] | NF | Design | Flexible graphical interface (responsive-design) |
| [24] | F | Design | Configurable graphical interface to access historical process information. |
| [25] | F | Design | Configurable graphical interface to access real-time process information. |
| | | | Configurable graphical interface to access synoptic view of the process information. |
| [26] | F | Design | Export of information through the graphical interface. |
| [27] | NF | Standardisation | Machine-to-platform communication protocol based on industry standards. |
| [28] | NF | Standardisation | Digital asset interfaces based on industry standards. |
| [29] | NF | Standardisation | Platform-to-third-party communication protocol based on industry standards |
| [30] | NF | Generic | The platform must be able to support the ingest of at least 1000 messages per second with an average of 10 values in each message. |
| [31] | NF | Generic | The platform must be able to support the simultaneous connection of at least 1000 devices. |
| [32] | NF | Generic | The platform must be able to serve subscribers (10 concurrently) new data with a latency of less than 2 seconds, serve a historical query of 1000 consecutive records over time in less than 1 second, and have historical data available for query in less than 10 seconds. |
| [33] | NF | Generic | The platform must be fault tolerant with each module of the solution being able to recover from failure situations (power failure, connection outages, etc...), ensuring that there is no loss of data. |
| [34] | NF | Generic | The usability of the platform should be promoted through the use of state visibility, consistency of views, error prevention, efficiency of use, aesthetic and minimalistic design, avoiding irrelevant information, avoiding unnecessary steps, providing error messages using simple language indicating the problem and constructively suggesting a solution. |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

# 3 Design of the Monitoring and Control Tool

The massive data generated throughout the industry contains a high complexity, which requires non-traditional data processing and analysis software applications to properly treat it, we are talking about IoT platforms. Traditionally, IoT platforms are in turn deployed in the Cloud through the use of a network of remote servers connected to the internet, which allow data to be stored, managed and processed ubiquitously.
These two digital enabling technologies, IoT and Cloud Computing, offer numerous benefits to the industry such as:

- Monitor, manage and maintain connected machines and devices in real time and anywhere.
- Integrate previously isolated systems.
- Automating the collection and analysis of data, facilitating a greater understanding of each of the processes, together and separately.

These benefits help the industry to boost its operational effectiveness, i.e. increase productivity, improve plant efficiency, uptime and asset quality, reduce operational risks, overall costs and changeover times.

The data generated by the new processes to recover water and heat and to treat water will be ubiquitously accessible and persisted in a non-aggregated form. This will facilitate decision-making, and the future exploitation of the knowledge stored in persistent information.

A IoT platform architecture is designed to support the data value chain in the three processes. The platform will provide the necessary capabilities to manage all the data generated, and therefore provide a system capable of supporting evidence-based decision-making. In addition, the architecture will support the deployment of intelligent modules aimed at impacting both process efficiency and data quality improvement.

The proposed conceptual architecture (see Figure 5) is defined using a modular structure based on services that allow for dynamically adding or removing modules without the need to re-implement or redefine any of the existing ones.

To ensure the interoperability of the architecture, it will be based on open interfaces compatible with most industry protocols and standards, and will also have the facility to integrate or adapt proprietary protocols (through customisation). These interfaces will ensure interoperability between multiple monitoring and control systems such as PLCs, SCADAs, machines, legacy databases, among others.

The IoT platform architecture is based on four main modules:
i) Communication module
ii) Persistence module
iii) Management module
iv) Analytics module

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

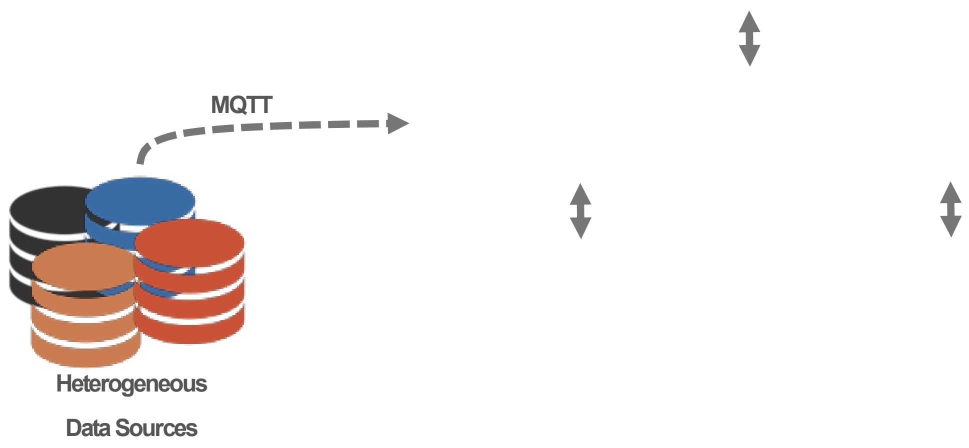

*Figure 1 Conceptual diagram of the iWAYS architecture to monitoring and control the industrial processes*

Below, more detail is provided for each module.

## 3.1 Communication Module

The communication module is responsible for managing communication, including external communication such as the integration of information from physical resources or the consultation of information by third-party software, as well as internal communication between the different elements of the architecture.

This module establishes a unidirectional communication with the management and control elements deployed in the platform (SCADAs, PLCs,... among others) and its objectives are:

- collect the data generated by these elements and transform them into a standardised structure;
- to communicate the data to all the modules; and
- to allow consultation of the data generated.

The communication is structured by using two mechanisms, REST APIs and publish/subscription. Each mechanism exploits the same data structure to facilitate the integration on the IoT Platform. Below, data structure, REST APIs and publish/subscribe is described.

**Data Structure**

In order to standardise the exchange of information through the MQTT broker and the REST APIs, a set of entities and their relationships have been defined in order to be able to model any industrial process and the information generated by its assets. The scheme described below presents these entities and relationships.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

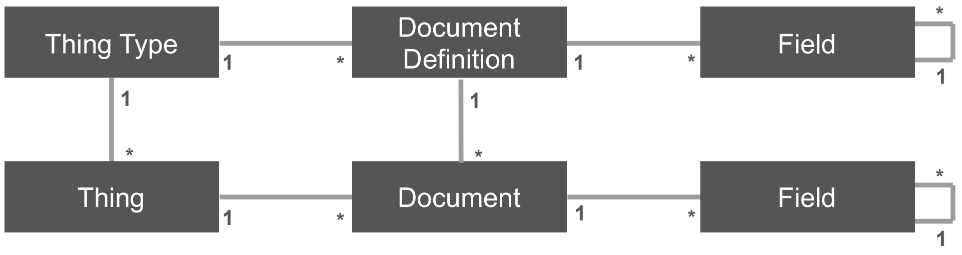

*Figure 2 Entities for the data modelling*

The top line shows the entities used to generalise the cyber-physical elements (ThingType, DocumentDefinition and Field). The bottom line shows the entities used to represent a specific instantiation of these elements (Thing, Document, Field). Each of these entities is described in detail below:

- ThingType represents an abstract element (a concept) of the cyber-physical world (e.g. Heat Pipe Condensing Economisers).
- Things represent concrete and unique instances of a ThingType (e.g. HPCE-01). ThingTypes are paired with a set of DocumentDefinition.
- DocumentDefinition is the specification of all the pieces of information that a Thing of that ThingType may generate/consume during its operation. DocumentDefinitions specify the fields (Field) that appear in a document, their type, their mandatory nature and the relation with other fields. It should be noted that DocumentDefinitions can have a completely arbitrary structure with nesting of fields of different types. This gives the platform the flexibility to integrate new cyber-physical devices with completely heterogeneous data sources. The DocumentDefinition concept can range from temperature of the water condensed by the HPCE-01 to information about the status of valve.
- Document is the minimum unit of information managed in the platform. A Document contains a collection of structured values (according to a specific DocumentDefinition) coming from readings made by the Things or values to be transmitted to them.

## REST APIs

Both access to historical data and management (consultation, creation, editing) of the solution's topology (ThingTypes, DocumentDefinitions, etc...) is carried out through the use of REST APIs (API Management and API Queries respectively) accessible from all the solution's modules. This is based and supported on a REST (REpresentational State Transfer) architecture. Representational State Transfer or REST is a style of software architecture for distributed systems, and refers to any interface between such systems that directly uses the HTTP protocol to obtain data in any format (XML, JSON, etc) without the additional abstractions of protocols based on message exchange patterns.
Its main features are:
i) a stateless client/server protocol, each HTTP message contains all the information necessary to understand the request;
ii) a well-defined set of operations that apply to all information resources, HTTP itself defines a small set of operations, the most important of which are POST, GET, PUT and DELETE;
iii) a universal syntax for identifying resources, in a REST system, each resource is addressable only through its URI;
iv) the use of hypermedia, both for application information and for application state transitions, the representation of this state in a REST system are typically HTML or XML.
The specification of such a REST API can be found in Annex I of this document.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

## Publish/Subscription

Through the real-time subscription service, (potentially web) clients can access data flowing in real-time through the platform. The platform's internal communication bus also offers the possibility for modules to receive and generate real-time data. Unlike the subscription service, the communication bus is not focused on browser-based interaction.
The most widespread option for acquiring the publish/subscribe paradigm is the use of brokers. The main benefits of using a broker are:
- Decoupling subsystems, facilitating administration.
- Increase scalability and improve responsiveness.
- Improve reliability, helping applications continue to run smoothly under increasing loads and control intermittent errors more effectively.
- Enable deferred or scheduled processing of information.
- Facilitate integration between systems using different platforms, programming languages or communication protocols, as well as between on-premises systems and applications running in the cloud.
- Facilitate asynchronous workflows.
- Improve testability because information channels can be monitored and messages can be inspected or logged as part of an overall integration testing strategy.

There are currently multiple brokers but MQTT-based brokers are the most relevant within the industry because they are a IoT standard totally aligned with the recommendations of the German Industry 4.0 and the standard OPC-UA.

MQTT, Message Queuing Telemetry Transport, is a message-centric protocol designed for M2M communications to transfer telemetry-style data in the form of messages from devices to a server or message broker.
Summarising the most important features of this technology, MQTT uses a publish-subscribe architecture that is completely agnostic to the content. An MQTT broker is capable of handling a thousand messages per second, adding a minor extra overhead into communications. Aditionally, MQTT provides capabilities related with the security, authentication and encryption of exchanged data can be handled by SSL or TLS.

The implementation of the broker will be based on Apache Mosquitto, which is distributed under an open source software licence used by the Eclipse Foundation. In addition, Mosquitto supports MQTT versions 3.1.1 and 5.0 and has docker containers, facilitating the management and deployment. As relevant points, the implementation has MQTT over WebSocket, supports message retention for new subscriptions, automatic publication of messages related to unexpected disconnections and QoS Level 2 QoS levels. Finally, it is widely used in multiple sectors, with an active community behind it.

*Table 2 Comparison of MQTT broker implementations*

**MQTT Broker implementations**

| **Attributes** | ActiveMQ | ActiveMQ Artemis | HiveMQ | JoramMQ | Mosquitto | RabbitMQ | VerneMQ |
|---|---|---|---|---|---|---|---|
| Licence | Apache 2.0 | Apache 2.0 | Commercial | LGPL, Commercial | EPL/EDL | MPL 1.1 | Apache 2.0 |
| Commercial support | YES | YES | HiveMQ | ScalAgent | TIBCO | Pivotal | Octavo Labs AG |
| Docker container | rmohr/activemq | vromero/activemq-artemis | hivemq/hivemq3 | NO | eclipse-mosquitto | rabbitmq:3 | erlio/docker-vernemq |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Windows Support | YES | YES | YES | YES | YES | YES | NO |
| MQTT version | 3.1 | 3.x | 3.x, 5.0 | 3.x | 3.1.1, 5.0 | 3.1.1 | 3.x, 5.0 |
| MQTT on WebSocket | YES | YES | YES | YES | YES | YES | YES |
| “Retain” Attribute | YES | YES | YES | YES | YES | Partial | YES |
| “Last will and testament” Message | YES | YES | YES | YES | YES | YES | YES |
| Persistent Session | YES | YES | YES | YES | YES | YES | YES |
| QoS Level 1 | YES | YES | YES | YES | YES | YES | YES |
| QoS Level 2 | YES | YES | YES | YES | YES | NO | YES |
| Puente | NO | NO | YES | YES | YES | NO | YES |
| Listener topic isolation | NO | NO | NO | NO | YES | NO | YES |
| Likes on GitHub | 1.4K | 0.5K | NA | NA | 2.4K | 5.4K | 1.7K |

It is important to highlight that the publish/subscription module will allow third party clients using different technologies to subscribe in real time to the data flowing on the IoT platform. To this end, a subscription system will be offered that will allow data to be received in JSON format, always taking into account the defined access control rules.

## 3.2 Persistence Module

This module is in charge of guaranteeing the persistence of the incoming data in the platform (coming from the physical resources of the processes) and the data generated by the platform itself, including data previously processed or without any type of transformation.

The Persistence Module is aimed at persisting temporary data flows (events generated during the process such as valve status, sensor measurements, among others) and structured data (physical resources involved and metadata associated with them). These characteristics are decisive factors when choosing the storage destination. Therefore, Timescale will be used to build the database due to it is optimized to work with time series and has relational capabilities. Moreover, it is scalable and supported by the main Cloud providers such as Amazon Web Services, Google Cloud Platform and Microsoft Azure.
The persistence module will consist of two persistence submodules, one relational and one time-series oriented, both offered by Timescale. All communication with the persistence module is centralised through the REST API that is part of the communication module.

The E-R schema that must be implemented to persist structured data is shown below.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

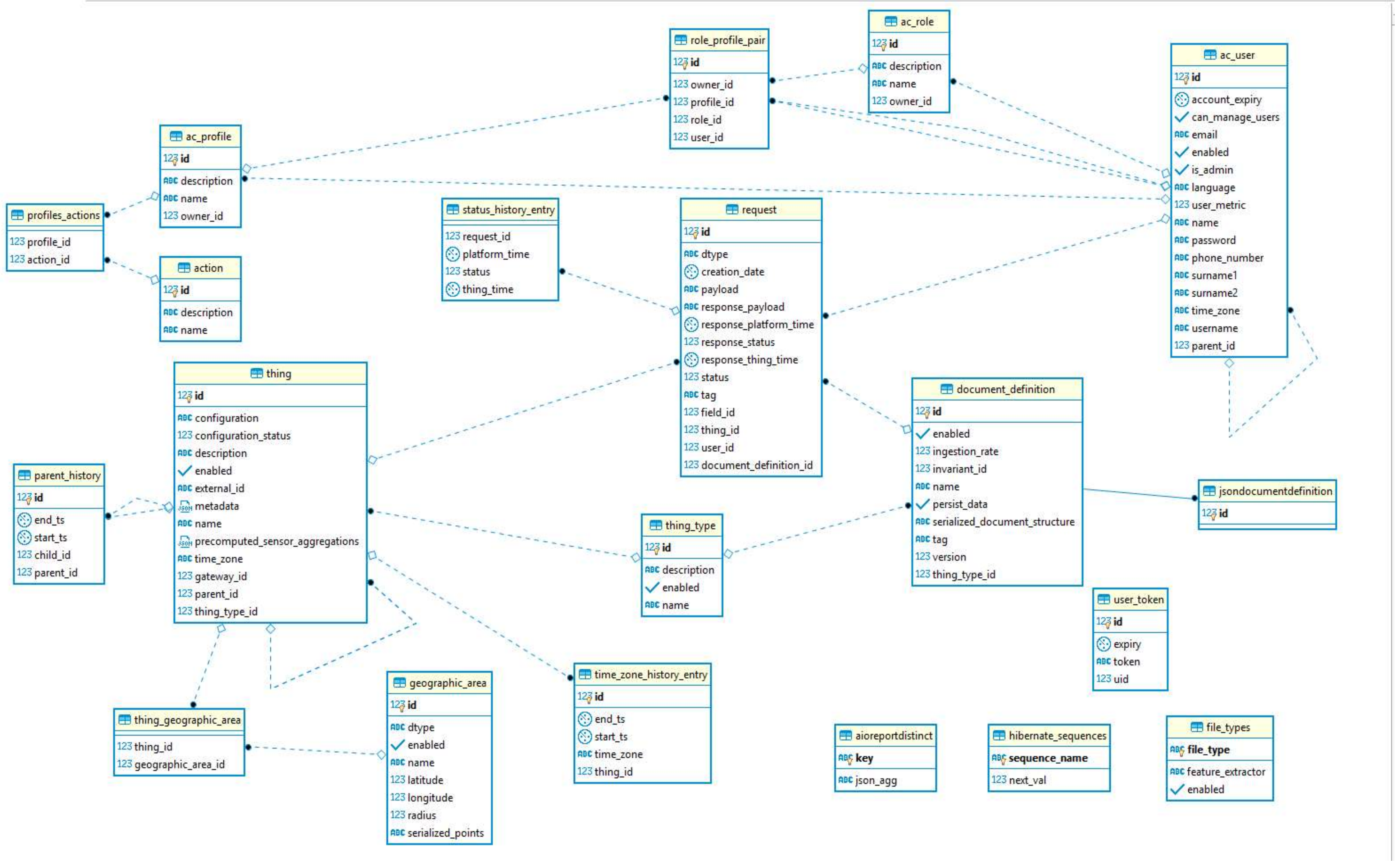

*Figure 3 E-R Schema of the database*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

## 3.3 Management Module

The management module includes the visualisation of the information; therefore, the use of Business Intelligence and Data Visualisation technologies is essential.
This module is responsible for the visualisation of the data stored in the persistence module. Thus, the user must be able to customise different graphical panels (time series oriented and synaptic), alerts and other components in order to ensure the good coordination and management of the platform's data.
In this sense, there are numerous tools designed solely and exclusively for the purpose of achieving a customised visualisation of the data. To achieve a first filtering of tools, all those without a free version have been discarded, such as Tableau, Chartier, Amazon QuickSight, Grow, DaphnaBI, look, Modo, Periscope Fecha and GoodData among others. Then, the analysis has focused on available free technologies such as Apache Superset, MetaBase, Redash, Apache Zeppelin, Kibana and Grafana.

The table presents multiple free visualization technologies and their possible integration with Timescale.

*Table 3 Integration of visualisation technologies with Timescale persistence technology*

| ***Data Source*** | **Metabase** | **Redash** | **Superset** | **Zeppelin** | **Kibana** | **Grafana** |
|---|---|---|---|---|---|---|
| *Timescale* | No | No | Yes | Yes | Yes* | Yes |

** Kibana only works with ElasticSearch. ElasticSearch can be integrated with different databases.*

Considering that the persistence system will be based on TimeScale, the only current options are Superset, Zeppelin, Kibana and Grafana. From a security (authentication) point of view, Superset, Kibana and Grafana offer authentication via Google OAuth, LDAP and Open ID. Grafana has a permission system based on roles and teams. Thus, users will be part of one or more teams, as well as have one or more roles. Kibana, on the other hand, offers similar functionalities, but only in its paid version. In contrast, Superset and Zeppelin offer different pre-defined profiles. Among the final candidates, Superset and Grafana are the most flexible tool from security point of view, offering multiples options to authenticate and manage the profiles. Nevertheless, Grafana is selected because this technology has a longer history in the community and is not in an embryonic state like Superset.

## 3.4 Analytics Module

The analytics modules are responsible for processing and analysing the data integrated in the platform in real time. Therefore, this module is in charge of processing the data transmissions coming from the various devices and data sources, to give them value and be properly stored in the Persistence Module. The module should be two main capabilities:

- Transform the data, i.e. modify all those data that require a transformation before being sent to the persistence module. For example, transformations to homogenise the information, to auto-calculate aggregates, among others.
- Analysing the data, i.e. executing the necessary calculations to generate knowledge (e.g. detect outliers) and KPIs as well as reports needed by the user.

This module will be mainly based on owner implementation, providing capabilities to analyse and process data in real time and in batches using the Java programming language. For this, the module will be subscribed directly to the MQTT broker, receiving the updated information

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

to be processed internally. Once the data is processed by the module, it will dump the generated information (e.g. alerts...) through the MQTT broker.

## 3.5 Schematic Design of the Monitoring and Control Tool

In the following, Figure 4 presents the architecture of the IoT platform described in the previous sections, together with the technologies associated with each module.

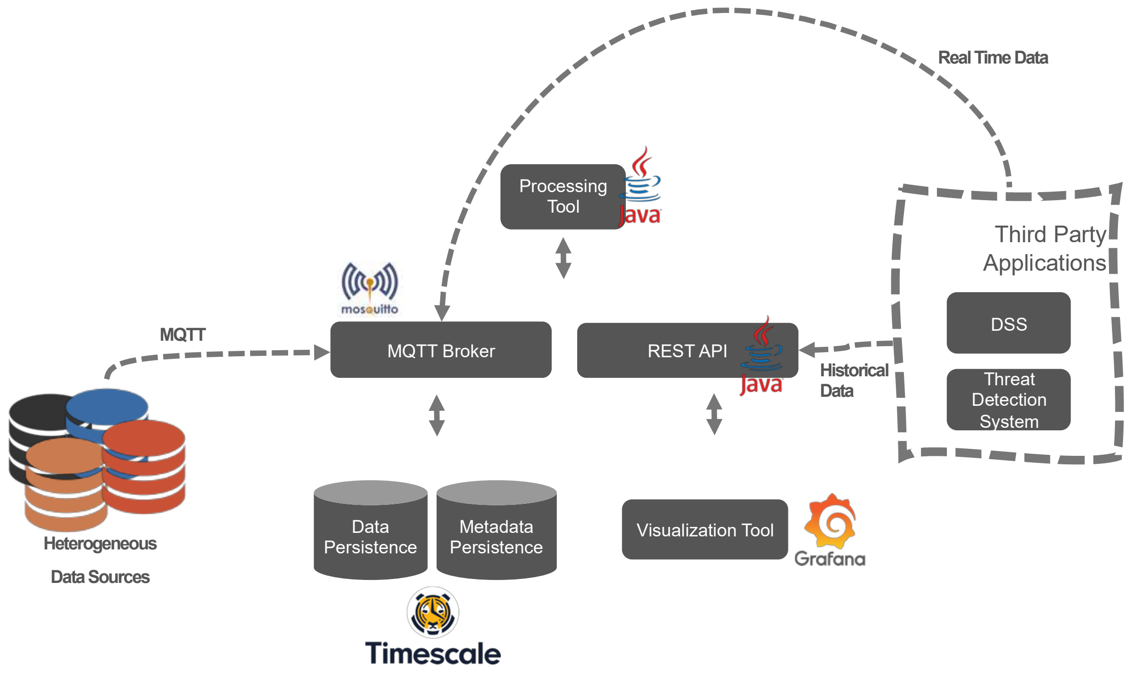

*Figure 4 Diagram of the architecture of the IoT platform for monitoring and control of iWAYS use cases*

The Figure 4 shows how the data is integrated into the platform through the communication module, more specifically the MQTT broker. The internal communication in the platform is also done through the MQTT broker, which allows the persistence module to consume the information to be stored, and the analytics model to process the information in real time. The communication with third party or external applications such as the DSS or the tool to detect unauthorised access (T3.3) is done through two interfaces: (i) the MQTT broker that provides real-time access to all the information through the publish/subscribe paradigm, and (ii) the REST API that provides access to all the historic information persisted by the IoT platform. Finally, the management module provide access to the end user to visualize the data.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

## 3.6 Sequence Diagram of the Monitoring and Control Tool

The aim of this section is to describe the behaviour of the IoT platform, including the external interaction and the internal interaction between the different modules described above. For this purpose, the 3 main operations to be performed with the platform are represented by means of sequence diagrams:

i) integration of external information (e.g., measurements) through the MQTT broker;
ii) consumption of real-time information by an external element of the IoT platform (e.g., the DSS); and
iii) consumption of historical information by an external element of the IoT platform (e.g., the DSS).

### Integration of external information

Figure 5 shows the sequence diagram related to integrate external information into the IoT platform. RAW data is received by the MQTT Broker from the PLC/Gateway by using MQTT standard protocol, notifying both the persistence module and the analytics module. The analytics module performs the necessary calculations and publishes the new information to the MQTT Broker for subsequent persistence. Therefore, the architecture persists both RAW and processed information in real time.

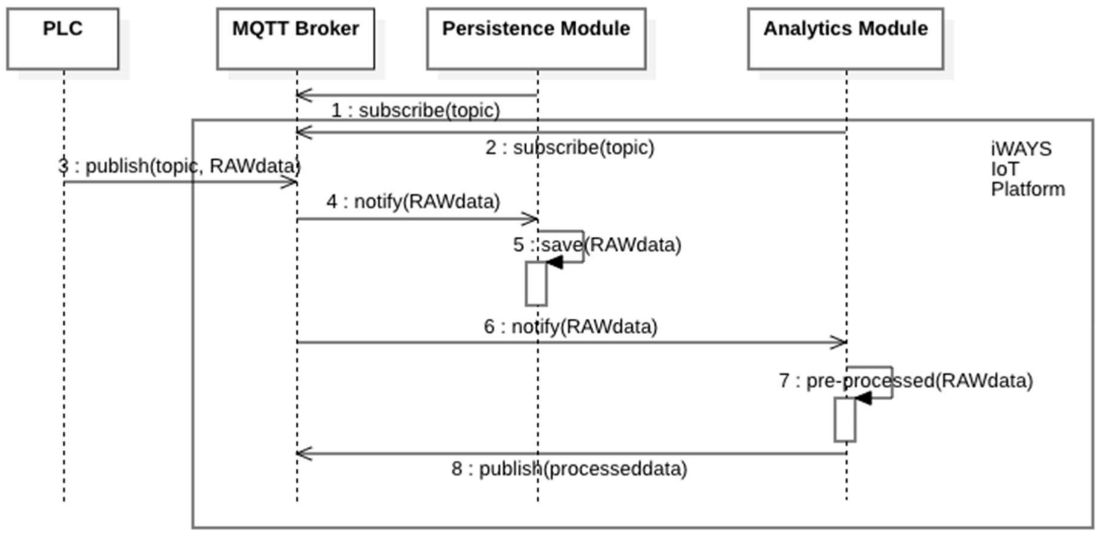

*Figure 5 Sequence diagram describing the behaviour of IoT platform when it receives information to be processed and it will persist*

### Consumption of real-time data

The Figure 6 presents the sequence diagram describing the real-time consumption of data by third party applications. Basically, the third-party application takes advantage of the benefits of the publish-subscribe paradigm, subscribing to a particular topic of the MQTT broker. Then, when a new message is received by the MQTT broker to that same topic, it will be forwarded to the third-party application. It is worth noting that multiple applications/modules can be subscribed to the same topic and therefore receive all the information in real time.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

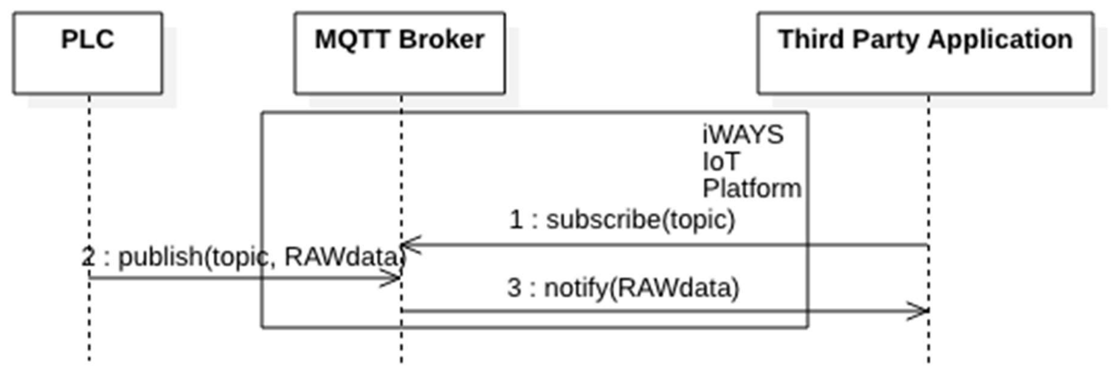

*Figure 6 Sequence diagram describing the behaviour of IoT platform when a third party application access to real-time data*

## Consumption of historical data

The Figure 7 presents the sequence diagram describing the consumption of historical data by third party applications. The third-party application takes advantage of the REST API to perform HTTP queries. Detailed information about the methods and data models of the REST API are available on Annex (see Annex A). Then, when a new query is received by the REST API, it directly accesses the database and returns the requested information.

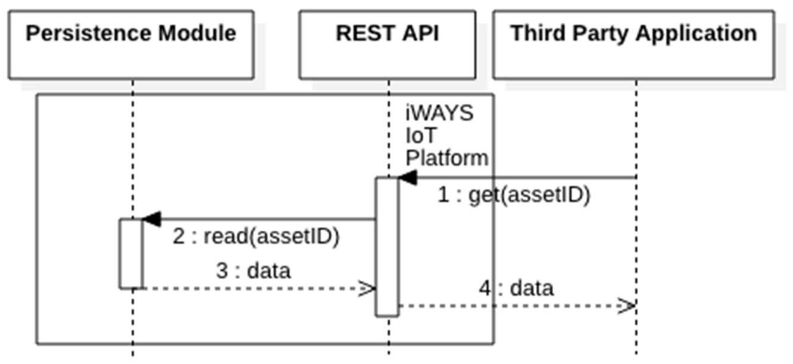

*Figure 7 Sequence diagram describing the behaviour of IoT platform when a third-party application access to historical data*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

# 4 Integration Approach and Customization of the Monitoring and Control Tool

The IoT platform is fully flexible and is designed to support all 3 case studies through the configuration of the platform. The communication between the physical world and the digital world is standardised through the MQTT communication protocol, allowing the integration of completely heterogeneous data sources. The Figure 8 shows the integration through the MQTT broker and how heterogeneous data sources can be integrated. PLCs or DAQs with MQTT capabilities can send data directly to the IoT platform, without the need for additional hardware. In the case of PLCs or DAQs without MQTT capabilities, a gateway can be integrated by using industrial standards such as Modbus and Profibus to bridge between the industrial device and the IoT platform. Finally, the same standardised channel for data integration can be used to integrate data extracted from APIs, databases or documents, both external and internal, by implementing specific connectors.

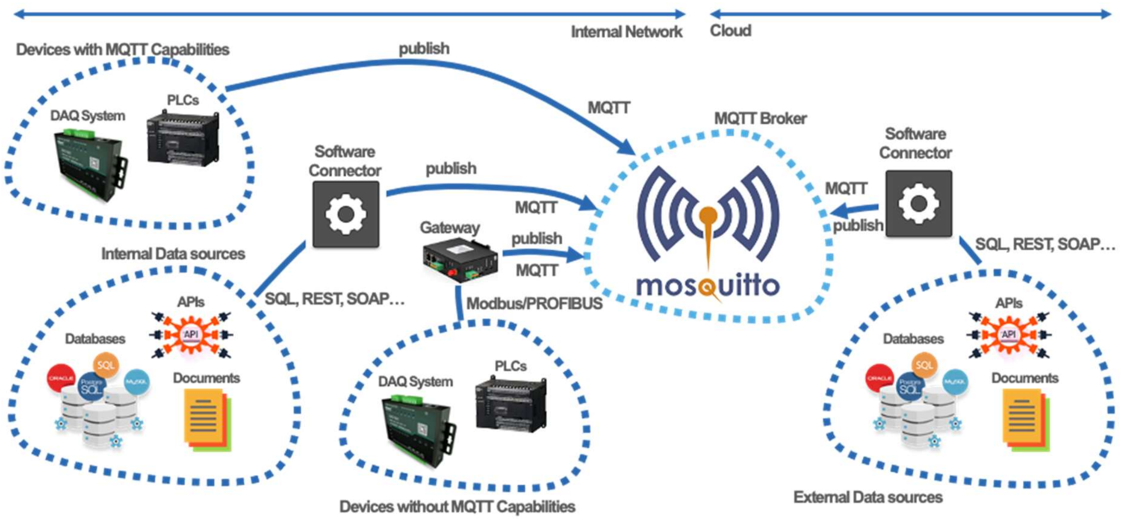

*Figure 8 Multi-source data integration through MQTT broker*

Throughout this section, detailed and use-case specific information is provided regarding the integration with the IoT platform and the visualisation of the information. More detailed information on the topics where to submit data and the structure of the data is provided.

As shown in the Figure 8, the information sent must be directed to a specific topic in the MQTT broker of the IoT platform. This topic corresponds to the physical resource of the process that is generating the information. In addition, each type of physical resource sends the information in a specific format which will be detailed in the next sections for each use case.

In general, the messages sent by the physical resources to the IoT platform contain two types of information:

- information related to the measurements such as a physico-chemical property of water or exhaust gas; and
- information related to the status of a physical resource such as the status of a valve or a pump among others.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

This information will be transferred based on a common structure of a key-value list represented through the JSON language. All transferred messages will share a common part used by the IoT platform for correct processing, which will be extended by adding information related to the physical resource such as measurements, status, among others. The following code shows this common part of all messages where the keys "id", "dd" and "t" are shown.

```
{
  "id": "resource identifier",
  "dd": "type of document",
  "t": Date and time in EPOCH format
}
```

*Code 1 Common message structure used by the IoT platform to process the messages*

Where:

- id - identifier of the physical resource from which the information originates;
- dd - type of document defining the structure of the message. It is used by the IoT platform to validate the syntax of the message; and
- t - date and time at which the information contained in the message was generated in EPOCH format

Below, an example of a possible message is presented. The message sends information about the temperature of the condensed water. The message indicates that the temperature is 85 ° C at 16:56:45 on the 4-1-2022.

```
{
  "id": "hpce-001",
  "dd": "dd_hpce_001",
  "t": 1609779405,
  "w_t_o_wcl": 1
}
```

*Code 2 Example of a message sent to the IoT Platform*

The topics and the structure of the messages processed by each topic for each industrial use case are presented on a confidential deliverable

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

# 5 Conclusions and Future Work

Connectivity and interoperability have gained enormous importance in the context of digitised industry, being responsible for connecting products, machines, people, and the environment in the same smart manufacturing system.
Cyber-physical systems augment the capabilities of physical assets through information technologies to support data storage, collection, and analysis, with the Industrial Internet of Things (IIoT) playing an important role.
These systems enable data collection, analysis and optimisation of production, increasing efficiency and reducing costs in the manufacturing process and service delivery.

The iWAYS project will enable the deployment of an IoT platform aligned with Industry 4.0 principles of connectivity and interoperability for the characterization of the recovered water and emissions, and provide the basis to deploy the DSS (Task 3.4).

Monitoring and control needs have been identified through interaction with work packages 4 and 5 for the three industrial use cases. In the case of Alufluor, 77 parameters have been identified to be monitored, of which 30 are related to DSS needs and 47 to the monitoring of the water and heat recovery process. For the use case of Concorde, 96 parameters have been identified, of which 55 water quality parameters are required by the DSS and 41 control parameters are needed for monitoring the heat and water recovery. For the use case of Tubacex, 62 parameters have been identified, of which 35 are related to DSS and 29 to the monitoring of the water recovery process.

Additionally, 34 functional and non-functional requirements for the design of the IoT Platform were identified, providing the basis to design the iWAYS IoT platform. Basically, the iWAYS IoT Platform should support the data value chain in the three industrial cases, providing the necessary capabilities to manage all the data generated, and therefore providing a system capable of supporting evidence-based decision-making. In addition, the IoT Platform should support the deployment of modules aimed at impacting both process efficiency and data quality improvement.

The IoT platform is defined using a modular structure based on services that allow for dynamically adding or removing modules without the need to re-implement or redefine any of the existing ones. It consists of 4 modules, namely communication module, persistence module, management module and analytics module.

The communication module is responsible for managing communication, including external communication such as the integration of information from physical resources or the consultation of information by third-party software, as well as internal communication between the different elements of the architecture. The communication is structured by using two standardized mechanisms, REST APIs and publish/subscription (MQTT).

The persistence module is in charge of guaranteeing the persistence of the incoming data in the platform (coming from the physical resources of the processes) and the data generated by the platform itself, including data previously processed or without any type of transformation. It is based on TimeScale technology that is optimized to work with time series and has relational capabilities.

The management module includes the visualisation of the information stored in the persistence module, providing customisable graphical panels (time series oriented and synaptic). The module is based on Grafana because it is integrable with Timescale and has a longer history in the community.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

The analytics module is responsible for processing and analysing the data integrated in the platform in real time. Therefore, this module is in charge of processing the data transmissions coming from the various devices and data sources, to give them value and be properly stored in the Persistence Module.

Finally, the designed platform is fully flexible and homogenize the communication through MQTT standard, standard totally aligned with Industry 4.0 paradigm. MQTT standard facilitates the communication between the physical world and the digital world due to novel industrial devices such as PLCs, DAQs, among other… includes MQTT capabilities. The topics and structure of the messages for the MQTT messages have been provided for the current state of the technologies developed on iWAYS. However, they will evolve in the future along with the evolution of the designs. Then, a new version of this document will be presented that will also include a sketch of the visualization for each case.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

# Annex: REST API and communication protocol

## a) Data Model

One of the key points that provides the data platform with flexibility in operations and allows the data integration from indifferent and infinite sources of information is the implemented data model. Next, this model is described together with the different entities of it, which allow the modelling of the information and open the way to the concept of interoperability.

Figure 9 shows the schematic of the designed data model. As can be seen, this seems simple, but this simplicity is what provides the great potential and capacities of abstraction of the information in it. The key lies in its structure, entities, and the relationships established between them, which allows the representation of any element or device in the IoT field and the information it sends. In this way, an attempt has been made to model the information from the definition of the concept 'thing' or 'Thing', as part of the Internet of Things.

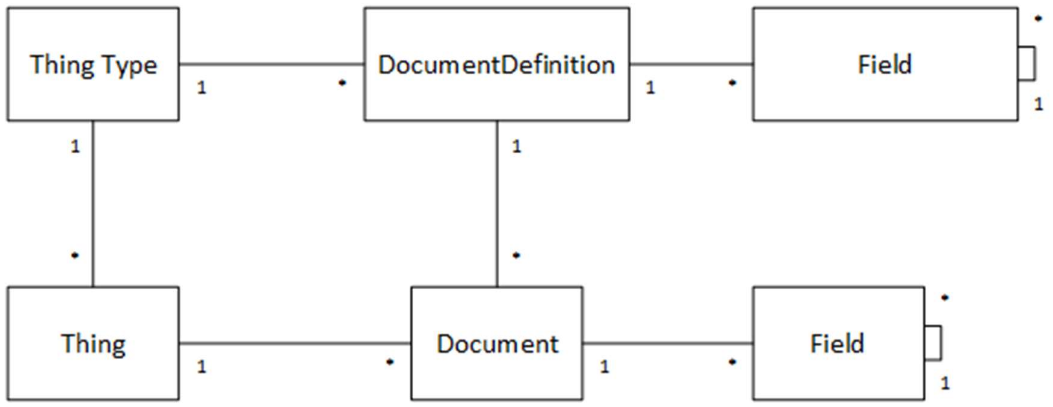

*Figure 9 Data model*

In order to correctly understand the implemented data abstraction, it is first necessary to separate the abstract entities (types) from the concrete ones (physical elements). On one hand, we find the *ThingType*, *DocumentDefinition* and *Field* entities. These elements are used to represent entities at a high level and create relationships between other entities, in the form of ontology. On the other hand, to represent concrete instantiations of the elements of the real world, we have the entities *Thing*, *Document*, and *Field*. These ones are used to instantiate elements, by way of analogy to object-oriented programming, and are containers of data. The platform uses this second group of entities for the transmission and communication of information.

<u>**Thing**</u>

The Thing entity is used primarily to instantiate elements, by way of analogy to object-oriented programming. Its purpose is to be a container for data. This entity is used mainly for the transmission and communication of information within the platform.

```
{
        id: number,
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

```
        name: string,
        description: string,
        externalId: string,
        parent: Thing,
        children: Thing[],
        gateway: Thing,
        type: ThingType,
        metadata: Map,
        precomputedSensorAggregations: Map,
        timeZone: string,
        timeZoneHistory: TimeZoneHistoryEntry[],
        enabled: boolean,
}
```

**TimeZoneHistoryEntry**

The *TimeZoneHistoryEntry* defines a delimited time range for the whole history of data provided by a certain entity (more commonly a *Thing*). It allows to attribute a time axis to the data that is contained within a physical representation.

```
{
        id: number,
        start: Date,
        end: Date,
        thing: Thing,
        timeZone: string,
}
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

**ThingType**

The main purpose of the *ThingType* entity is to encapsulate inside the entity itself a representation of a concept or abstract element of the physical or real world.

```
{
        id: number,
        name: string,
        description: string,
        documents: DocumentDefinition[],
        enabled: boolean,
}
```

**ThingWithDocument**

This is an informative entity in which a specific *Thing* is doted with the documents contained on it to facilitate the searching procedure.

```
{
        id: number,
        name: string,
        description: string,
        externalId: string,
        gateway: Thing,
        type: ThingType,
        metadata: Map,
        precomputedSensorAggregations: Map,
        timeZone: string,
        enabled: boolean
}
```

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

**DocumentDefinition**

The *DocumentDefinition* entity describes the schema and fields of a *Document* to enable this logic within the platform. As a result, *Document* entities are the platform's data transport containers, physical representations of the abstraction defined by *DocumentDefinitions*.
For each of the *ThingType*s, there is a set of *DocumentDefinition* entities. These documents are the concrete specification of the data that a *Thing* corresponding to a specific *ThingType* will contain. That is, these documents are concrete instances and concrete data schemas of a *ThingType* schema.

```
{
       id: string,
       version: number,
       lastVersion:boolean,
       name: string,
       tag: string,
       persistData: boolean,
       thingTypes: ThingType[],
       enabled: boolean,
       fields: Field[],
}
```

**Field**

Referring to the previously introduced Figure 9, within the *DocumentDefinition* and *Document* entities there are *File* entities. These fields serve as containers to describe the specific typing that will be required for the correct serialization and deserialization of the documents.

```
{
       id: string,
       name: string,
       tag: string,
       type: Type,
       containsThingTime: boolean;
       containsExternalThingId: boolean;
       containsDocumentDefinitionId: boolean;
       nullable: boolean,
       fields: Field[],
}
```

**Type**

Following the wide variety of data types, a specific set of types was isolated to define the multiple representations that the data of a specific thing can assume.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°958274

```
Type = "INTEGER_NUMBER" | "DECIMAL_NUMBER" | "STRING" | "DATE" | "BOOLEAN" |
"INTEGER_NUMBER_ARRAY" | "DECIMAL_NUMBER_ARRAY" | "STRING_ARRAY" | "BOOLEAN_ARRAY"
| "GROUP";
```

## b) Platform Methods

Once the entities are already introduced, the next step involves the methods which define calls to interact with the API. Each method consists of a definition made by:

- **Method name**: Unique identifier of each method with summarizes the objective of it.
- **Method description**: Extended definition of the main purpose of the method.
- **HTTP verb**: Base layer defined by the five most important HTTP methods[4].
- **URL**: Specific endpoint which appends to the origin API route.
- **Body**: Data sent by the client to the API. It is defined following the JSON format.
- **Response**: Beyond the result of the operation, the response field simplifies the definition by referring to the resource returned by the api.

In a similar way to the entities seen in Annex a), the request module has its own entities definition which is completely independent of the data model seen previously in Figure 9.

### Entities

TimeSeriesRequest

| TimeSeriesRequest | |
|---|---|
| **thingsIds** | Internal Ids of the things to filter |
| **thingsExternalIds** | External Ids of the things to filter |
| **start** | Start date |
| **end** | End date |
| **limit** | Limit lengths of results |
| **documents** | Documents to retrieve and their associated query |
| **downsampling** | Sting which can become: NONE, MINUTE, QUARTERLY, HALF_HOUR, TREE_QUARTERS, HOURLY. |

[4] Five most important HTTP methods: GET, POST, PUT, DELETE, and PATCH.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

|  |  |
|---|---|
| **outputFormat** | JSON or CSV output format |

```
{
	thingsIds: number[],
	thingsExternalIds: string[],
	start: Date,
	end: Date,
	limit: number,
	documents:
	  Dictionary<document_definition_id[.version]:string,query:DocumentQuery>,
	downsampling: string <NONE,MINUTE,QUARTERLY,HALF_HOUR,TREE_QUARTERS,HOURLY>,
	outputFormat: OutputFormat <JSON, CSV>
}
```

Some important considerations are highlighted referring to the usage of this specific entity:

- At least a value must be provided for *start* or *end* or *limit*.
- If a value is provided for *start* or *end*, no value can be provided for *limit*.
- At least one item must be provided in the *documents* dictionary. In case there is *outputFormat* in CSV value, only one element can be specified in the *documents* dictionary.
- The key of the *documents* dictionary corresponds to the identifier of the *DocumentDefinition* entity to be consulted and the query to be applied (optionally). By default, this call will return the documents of the latest version of the specified *DocumentDefinition*. If a different version wants to be consulted, the version number preceded by a dot (".") must be included at the end of the *document_definition_id*.

DocumentQuery

The query endpoints that return *Documents* allow the option of providing an advanced filter. The endpoint of real-time subscription to a document also accepts this parameter. In the case of combining the use of the query attribute with the downsampling option, the records will be filtered prior to the temporary addition of the same. There exist two main conceptions:

- This filter must be expressed as a serialized JSON object.
- It must be a clause or conjunction / disjunction of clauses

The clauses have the structure:

```
{operator: [{operand1, operand2, ..., operandN}]}
```

| **Operators: basic** |  |
|---|---|
| **==** | Validates that the value of a document field "A" is equal to a constant or the value of another document field "B". |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

| | |
|---|---|
| **!=** | Denial of equality. |
| **>** | Validates that the value of a document field "A" is greater than a constant or the value of another document field "B". |
| **>=** | Validates that the value of a document field "A" is greater or equal to a constant or the value of another document field "B". |
| **<** | Validates that the value of a document field "A" is less than a constant or the value of another document field "B". |
| **<=** | Validates that the value of a document field "A" is less or equal to a constant or the value of another document field "B". |
| **in** | Validates that a constant value is within the values of a field of the collection type document (the order of the operands must be constant) |
| **not_in** | Validates that a constant value is not found within the values of a collection-type document field (the order of the operands must be constant) |

Moreover, compositions of clauses can be made using advanced operators such as the conjunction and disjunction operators.

| **Operators: advanced** | |
|---|---|
| **and** | Forces to prove the truth of all the clauses contained as values. |
| **or** | Forces to prove at least one truth from all the clauses contained as values. |
| **!** | Denies the truth of all the clauses contained as values. |

| **Operand** | |
|---|---|
| **constant** | `<Number:1; String: "a", List: [1, 2, 3, 4], Boolean: false>` |
| **Field** | `{"field": {fieldId}}` |

Some examples of valid filtering techniques are:

| | | |
|---|---|---|
| **Example 1** | **Objective** | The "*integers_collection*" field contains the value 33. |
| | **Clause** | `{"in": [33, {"field": "integers_collection"}]}` |
| **Ex** | **Objective** | The "*boolean_field*" field is true. |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

| | | |
|---|---|---|
| | **Clause** | `{"==": [{"field": "boolean_field"}, true]}` |
| **Example 3** | **Objective** | The field "*boolean_field_1*" is true and the field "*boolean_field_2*" is false. |
| | **Clause** | `{"and": [`<br>`{"==": [{"field": "campo_booleano_1"}, true]},`<br>`{"==": [{"field": "campo_booleano_2"}, false]}`<br>`]}` |

For more information about this structure, see the official JsonLogic documentation (ref.).

The presentation of the content related to the methods has been carried out by grouping them under the entity that is involved.

## Methods

### Authentication

| | |
|---|---|
| **Method** | login |
| **Description** | Get a session token. |
| **HTTP verb** | POST |
| **URL** | /login?user={user}&password={password} |
| **Body** | - |
| **Response** | `{`<br>`user: string, // Username`<br>`token: string, // Token to use in requests`<br>`}` |

| | |
|---|---|
| **Method** | logout |
| **Description** | Invalidates a session token. |
| **HTTP verb** | DELETE |
| **URL** | /login |
| **Body** | - |
| **Response** | - |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

Document

| | |
|---|---|
| **Method** | page |
| **Description** | Returns a list of *DocumentDefinition*. |
| **HTTP verb** | POST |
| **URL** | /documentDefinition/page |
| **Body** | `{`<br>`        offset: number,`<br>`        pageSize: number,`<br>`        sortAsc: boolean,`<br>`        filters: {`<br>`                name: string,`<br>`                lastVersion: boolean[],`<br>`        }`<br>`}` |
| **Notes** | - If `pagination.PageSize` is set to -1, the response will not be grouped into pagination.<br>- The pagination sorting is done taking as reference the names. |
| **Response** | Page<DocumentDefinition> |

| | |
|---|---|
| **Method** | get |
| **Description** | Returns a list of the documents associated with the *Thing* in the specified period and with the required granularity. If "limit" is specified, this limit is global for all *DocumentDefinitions* associated with the *Thing*. |
| **HTTP verb** | POST |
| **URL** | /documents/query |
| **Body** | *TimeSeriesRequest* |
| **Response** | List<*Document*> |

Thing

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

| | |
|---|---|
| **Method** | page |
| **Description** | Returns a list of *Things*. |
| **HTTP verb** | POST |
| **URL** | /thing/pageWithLastDocuments |
| **Body** | <pre>{<br>    offset: number,<br>    pageSize: number,<br>    sortAsc: boolean,<br>    filters: {<br>        externalId: string,<br>        enabled: boolean[],<br>        name: string,<br>        type: number[],<br>        parentId:number,<br>    };<br>    sortBy: string;<br>}</pre> |
| **Notes** | - If `pagination.PageSize` is set to -1, the response will not be grouped into pagination.<br>- An example of a valid `filters` structure is:<br><pre>filters: {<br>        "externalId": ["eid1"],        // Unique string<br>        "enabled": ["true", "false"], // Boolean's list<br>        "name:": ["name_test"],        // Unique string<br>        "type": ["1", "2", ...],       // List of *ThingType* ids<br>        "parentId": ["23"]             // Unique Thing id<br>}</pre> |
| **Response** | Page<ThingWithDocument> |

| | |
|---|---|
| **Method** | get |
| **Description** | Returns a *Thing* along with its associated documents (based on the query) |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

| | |
|---|---|
| **HTTP verb** | POST |
| **URL** | - Internal ID: /thing/get?id={thingId}<br>- External ID: /thing/get?externalId={externalId} |
| **Body** | *TimeSeriesRequest* |
| **Response** | *ThingWithDocuments* |

ThingType

| | |
|---|---|
| **Method** | page |
| **Description** | Returns a paginated list of ThingType objects |
| **HTTP verb** | POST |
| **URL** | /thingType/page |
| **Body** | `{`<br>`        offset: number,`<br>`        pageSize: number,`<br>`        sortAsc: boolean,`<br>`        filters: {`<br>`                enabled: boolean[],`<br>`                name: string,`<br>`        }`<br>`        sortBy: string [id,name],`<br>`}` |
| **Notes** | - If `pagination.PageSize` is set to -1, the response will not be grouped into pagination.<br>- Documents are not included in the *ThingType* response. |
| **Response** | Page<ThingType> |

| | |
|---|---|
| **Method** | get |
| **Description** | Returns a *ThingType* object from its `thing_type_id`. |
| **HTTP verb** | POST |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274

|  |  |
|---|---|
| **URL** | /thingType/get/{thingTypeId} |
| **Body** | - |
| **Notes** | - Documents with all its relations are included in the *ThingType* response. |
| **Response** | *ThingType* |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°958274