RESPOND
DEMAND RESPONSE FOR ALL

Integrated Demand REsponse SOlution Towards Energy POsitive NeighbourhooDs

WP5 SYSTEM INTEGRATION AND INTEROPERABILITY

T5.1 HOME AUTOMATION INTEROPERABILITY INTERFACES

D5.1 ENERGY GATEWAY FOR HOME AUTOMATION INTEROPERABILITY

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 768619

PROJECT ACRONYM	RESPOND
DOCUMENT	**D5.1 Energy gateway for home automation interoperability**
TYPE (DISTRIBUTION LEVEL)	☒ Public ☐ Confidential ☐ Restricted
DELIVERY DUE DATE	31.03.2019.
DATE OF DELIVERY	31.03.2019
STATUS AND VERSION	Final, version 1.0
DELIVERABLE RESPONSIBLE	IMP
CONTRIBUTORS	TEK, ENE, DEV, FEN, DEX
AUTHOR (S)	Lazar Berbakov, Nikola Tomašević, Marko Batić
OFFICIAL REVIEWER(S)	Henrik N. Knudsen and Toke Haunstrup Christensen (AAU)

DOCUMENT HISTORY

	ISSUE DATE	CONTENT AND CHANGES
0.1	14.02.2019.	Table of contents
0.2	19.02.2019.	First draft
0.3	13.03.2019.	Energomonitor contribution included
0.4	19.03.2019.	Fenie Energia nad Dexma contributions included
0.5	20.03.2019.	Tekniker contribution included
0.6	27.03.2019.	Comments and suggestions from AAU included
1.0	28.03.2019.	Minor corrections

EXECUTIVE SUMMARY

This document on “Energy gateway for home automation interoperability” reports on the results achieved in Task 5.1 – “Home automation interoperability interfaces” from Month 7 to Month 18 of the RESPOND project. The goal of this task was to deal with interoperability of RESPOND platform towards home automation and building management systems, metering equipment (both energy and comfort), and energy resources. In particular, it aims to support integration of RESPOND platform with external hardware and software systems, thus enabling not only acquisition of data, but also providing a way to issue the control actions to be performed under the cooperative demand strategy. To achieve this overarching goal, the concept of energy gateway will be deployed in the pilots. The gateway comprises open software platform OGEMA as well as custom developed modules aimed at integrating different RESPOND components and external systems.

In order to provide an overview, we briefly revisit the RESPOND platform architecture and interfaces. Next, we list the interoperability requirements and explain how these requirements have been considered in different project tasks which have been realized after the requirements definition. Then, we provide more details regarding RESPOND middleware and canonical data model which represent the core pillars supporting interoperability among different platform components. Finally, we provide a specification of OGEMA implementation, custom developed platform components and conclude the document with the main outcomes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
BMS	Building Management System
CDM	Canonical Data Model
CLI	Command Line Interface
DB	Database
DCWC	Domestic Cold Water Consumption
DHWC	Domestic Hot Water Consumption
DoW	Description of Work
DR	Demand Response
EMS	Energy Management System
HTTP	Hyper Text Transfer Protocol
HVAC	Heating, ventilation and air conditioning
HW	Hardware
ICT	Information and Communication Technology
IP	Internet Protocol
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
OGEMA	Open Gateway Energy Management Alliance
REST	Representational State Transfer
SW	Software
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus
XML	Extensible Markup Language

1. INTRODUCTION

The goal of Work Package 5 is to perform system integration and to ensure interoperability of the RESPOND system components. This document presents the outcome of Task 5.1 which, in particular, is focused on specification and design of home automation interoperability interfaces. As described in Description of work, Task 5.1 will support integration of the RESPOND platform with external hardware and software systems, enabling not only acquisition of data, but also providing a way to issue the control actions to be performed under the cooperative demand strategy. In Deliverable D1.3 [1], interoperability is defined as the ability of a system to work with or use the parts of equipment of another system. In the context of the RESPOND project, the project partners have designed the architecture of the system so that interoperability is supported from the beginning. Besides, the goal of the RESPOND project is to complement and enhance the existing smart home and building management systems, with the aim to improve the energy efficiency and optimize the costs by seamless integration of cooperative DR programs.

RESPOND system consists of a number of devices which have been previously installed as well as of those which will be deployed during the course of RESPOND project, such as: home automation devices, building management systems, metering equipment, sensors and actuators. All these devices are produced by different manufacturers, using mutually incompatible communication protocols and interfaces. In order to be able to take advantage of the legacy systems already present at pilot sites, in addition to deploying additional components, RESPOND project consortium has been focused on interoperability from the beginning of the platform architecture specification to the deployment and integration tasks which will be described in this document.

1.1 RELATION TO OTHER RESPOND ACTIVITIES

In Task 1.3 which lasted from Month 1 to Month 6 of the project, we have investigated the state of the art regarding interoperability. Besides, we presented the list of installed and planned equipment which served as valuable input to the document on RESPOND strategy to achieve interoperability (D1.3), which was delivered in Month 6. In parallel, Task 2.2 has been dealing with in the seamless integration of RESPOND technology tiers aiming to match the existing systems and underlying technology concepts with system reference architecture which was designed in Task 2.1. The output of these activities was summarized in document D2.2 [2] on integration of key RESPOND technology tiers.

1.2 REPORT STRUCTURE

In this report, we provide in details, the results of Task 5.1 which has been dealing with the development of home automation and interoperability interfaces. In particular, in Section 2 we revisit the RESPOND platform architecture and interfaces. Next, in Section 3, we list the main

interoperability requirements which have been addressed during interface development and integration tasks. In addition, we describe RESPOND middleware development and the Canonical Data Model, which was tailored to support integration of different home automation and external systems. Furthermore, Section 4. outlines customization of interfaces which were developed by the involved project partners. Finally, in Section 5 we conclude the document with the main results.

2. RESPOND PLATFORM AND INTERFACES SPECIFICATION

The aim of this section is to summarize the RESPOND architecture which was provided in Deliverable D2.1 [3] (RESPOND system reference architecture). In that document, starting from the draft of architecture given in RESPOND project proposal and inspired by previous successful projects, the involved project partners have designed the RESPOND reference architecture which will be revisited in the sequel.

2.1 PLATFORM ARCHITECTURE

In Figure 1, we present the RESPOND reference architecture, which consists of different components. The field level devices represent different sensors, actuators, building and energy management systems which have been already deployed, or will be installed in the pilot sites during the course of the project. Such devices serve the purpose of monitoring energy consumption, environmental parameters, user behaviour and at the same time enable RESPOND platform to control devices in the field via actuators. Technology providers in the consortium, Develco and Energomonitor will provide the gateways for newly deployed devices, whereas legacy devices will be integrated using Energy Gateway OGEMA.

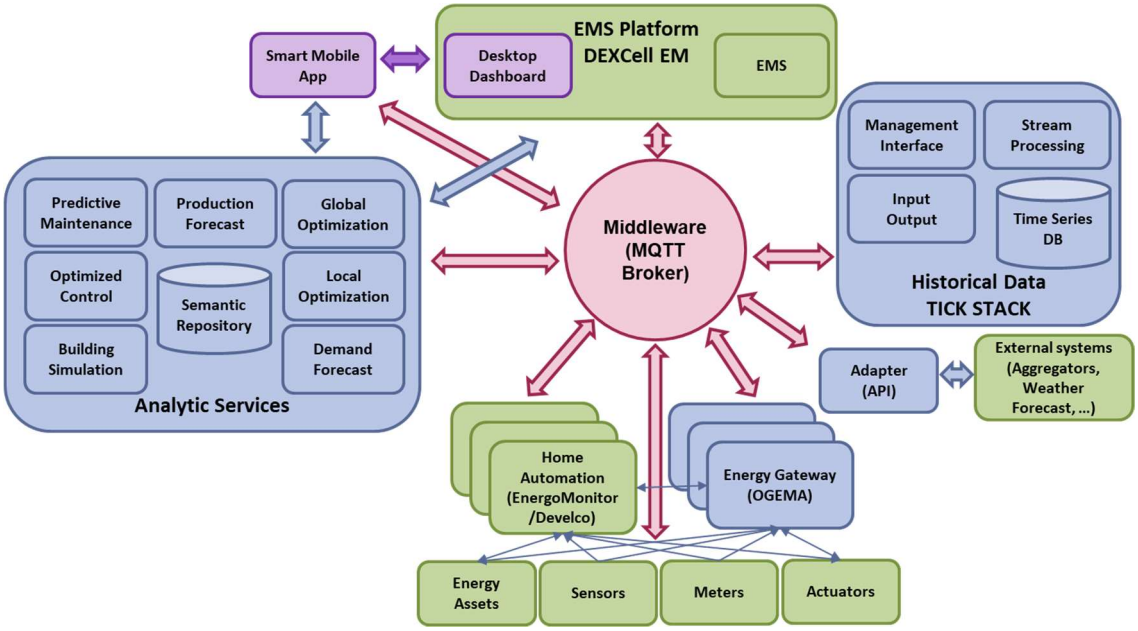

FIGURE 1: RESPOND REFERENCE ARCHITECTURE

The field level devices communicate with the rest of the RESPOND platform via message broker, based on lightweight MQTT protocol, which effectively supports a large number of simultaneous connections by using publish/subscribe communication model. All the data collected from the field level devices as well as those generated by other parts of the RESPOND system are stored in a central data repository based on open-source TICK stack. The central component of the TICK

stack is InfluxDB, a time series database particularly suitable for effective storage and retrieval of data generated by field level monitoring devices and other types of time stamped data.

RESPOND system also leverages external sources of data, such as energy aggregators and weather forecasting services, which provide data required by other parts of the RESPOND platform. The interface towards external services will be implemented by using custom developed adapters.

The main value of RESPOND platform lies in a number of analytic services, which make use of data collected by field level devices, external services and other parts of RESPOND platform to calculate the optimal energy consumption and measures to achieve it. The platform communicates with the users via intuitive mobile and web applications, where the mobile app (developed by Tekniker) is primarily designed for home owners, whereas the web application Dexcell (offered by Dexma) is directed towards building and energy managers who need more comprehensive reports as a support in their daily tasks.

2.2 INTEROPERABILITY INTERFACES

In Figure 2, we present the interfaces among different components of RESPOND platform. As can be seen, field level devices (Develco and Energomonitor), Energy gateway OGEMA, and water consumption custom adapter use MQTT protocol for monitoring data and control action transfer from/to RESPOND platform. On the other hand, external systems (weather and energy pricing services), Energy manager dashboard (Dexcell), and Smart mobile application use standard HTTP protocol for integration with RESPOND. More details regarding each component will be provided in the following sections.

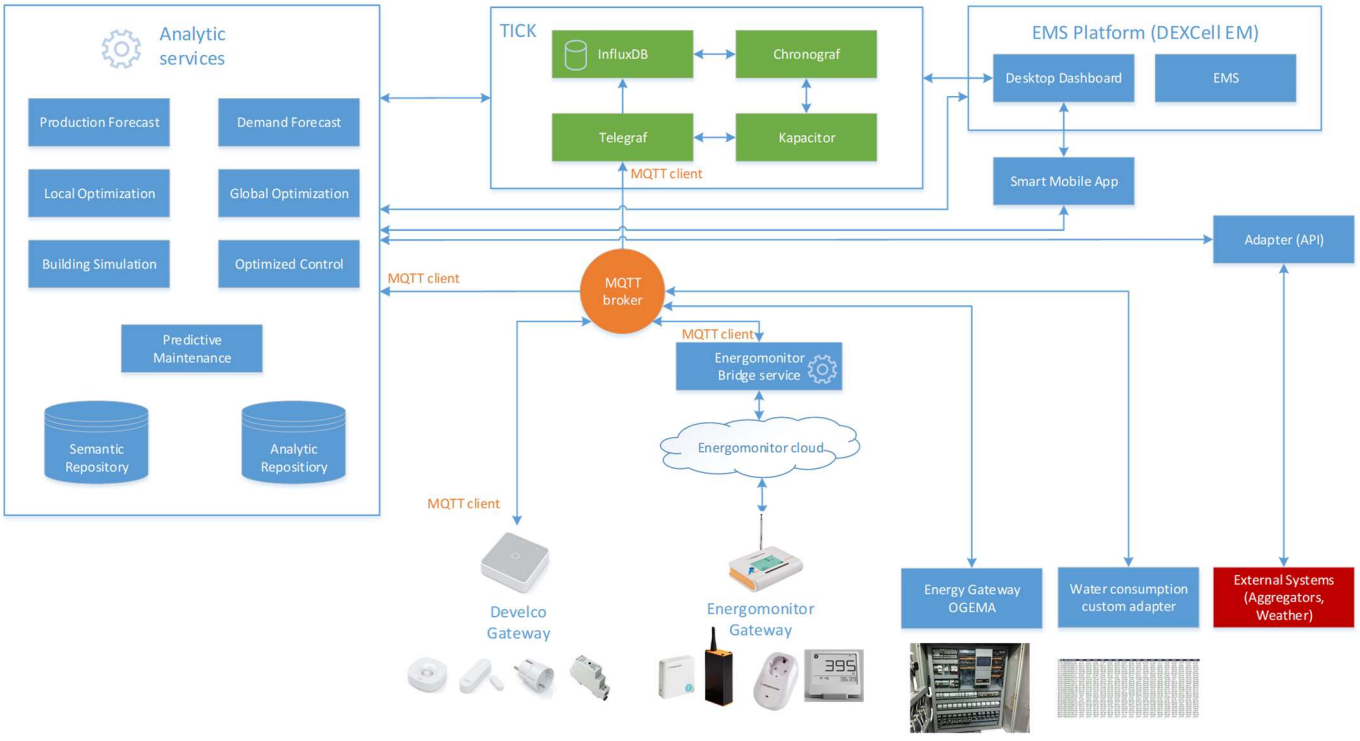

FIGURE 2: RESPOND PLATFORM INTERFACES

3. INTEGRATION AND INTEROPERABILITY CONCEPTS

3.1 INTEROPERABILITY REQUIREMENTS

In D1.3 [1], which focused on RESPOND strategy to support interoperability, we set a list of requirements with a corresponding rationale. In this section, we will revisit the interoperability requirements and explain how we have satisfied them.

TABLE 1: INTEROPERABILITY REQUIREMENTS

[Headline]	[Req Description]	[Classification]	[Type]	[Rationale]	[Requirement satisfied]
Architectural model composition to support interoperability	The Architectural model shall natively support the interconnection and composition of devices and systems produced by different manufacturers	WP2–Use case deployment and follow up	T2.1-System architecture design	The RESPOND system should support a variety of different devices to allow long-term interoperability and to prevent vendor lock-in	The data are collected from pilot sites and stored in the cloud platform.
Information model definition to support semantic interoperability	RESPOND should considering Information Modelling standard processes	WP4–ICT enabled cooperative demand response model	T4.1-Semantic information model	A way to specify and exchange energy and building assets information should be defined, preferably by using well defined standard (e.g. EEBus SPINE)	The canonical data model was designed in order to support different devices and systems. Customization of hardware and software provided by project partners has been performed. An ontology is designed to store the knowledge regarding the system.
Integration architecture	RESPOND overall architecture should be based on open and standard	WP2–Use case deployment and follow up	T2.1-System architecture design	Web services and Service Oriented Architecture are the preferable choice as integration frameworks. Also,	Integration platform is based on MQTT protocol for collecting measurements and control actions.

	integration frameworks and communication protocols			SOAP and WSDL-based APIs or REST protocols should be considered for synchronous communication. Asynchronous communication should be based on OSGI publish/subscribe mechanisms	The data repository is based on open source TICK stack composed of Telegraf, InfluxDB, Chronograf and Kapacitor modules.
Home automation interoperability	RESPOND should support well known communication protocols for home automation	WP5-System Integration and Interoperability	T5.1-Home automation interoperability interfaces	Communication with home automation system should be based on well-known communication protocols (Zigbee, Z-Wave, etc.)	RESPOND platform supports Zigbee/MBus/Zwave protocols (by Develco gateway), as well as proprietary communication protocol used by Energomonitor.
Smart metering interoperability	RESPOND should support well known communication protocols for smart metering	WP5-System Integration and Interoperability	T5.2-Smart grid connectivity	Communication with smart metering devices (Electicity, water and gas consumption) should be based on well-known communication protocols (Zigbee, Wireless M-bus, etc.)	Both Develco and Energomonitor equipment can be used on either legacy meters with pulse output. Develco provides prosumer meter which can be used to replace the legacy meters if necessary.
BMS/EMS interoperability	RESPOND should support newly deployed as well as already available legacy BMS/EMS systems	WP5-System Integration and Interoperability	T5.1-Home automation interoperability interfaces	Communication with BMS/EMS systems should be done in a protocol agnostic manner by supporting different open protocols (BACnet, KNX)	Custom implementation of OGEMA gateway has been implemented for the purpose of communication with BMS/EMS systems.
Grid interoperability and services 1	RESPOND should try to comply with the main standards and initiatives	WP4–ICT enabled cooperative demand	T4.1-Semantic information model	The main outputs from technical working-groups of CEN/CENELEC/ETSI	Interface towards smart grid will be based on OpenADR standard.

	relevant to Smart Grids and advanced metering infrastructures especially concerning: data exchange, ICT security, distribution management and tariff and load control	response model		should be considered. For instance: - TC57 WG21 (interface/protocols for systems connected to grid) - SG-CG M490 (mandate of the smart grid coordination group to define a SG reference architecture) - IEC 61850 (comm. networks and systems for s/stations automation) - CIM / IEC61968 (energy distribution management) - CEER Publications - OASIS Energy Interoperation Committee Specification Also, the AMI security profile from SG Security Working Group (UCAIug) & The NIST Cyber Security Coordination Task Group may be considered for the cyber security issues.	
Grid interoperability and services 2	RESPOND should try to align with current standardization efforts in the field of energy standard	WP4–ICT enabled cooperative demand response model	T4.1-Semantic information model	In order to implement energy working demand-response processes, it is important that energy consumers/producers and utilities share	Interface towards smart grid will be based on OpenADR standard.

	information exchange			standardized data on energy characteristics, energy availability, energy price, flexibility offers, operational schedules, building information, etc. For instance, EEBus SPINE, eMIX, oBIX, CIM, etc. are example of initiatives to consider as reference for RESPOND.	
Energy Gateway	RESPOND should start from ongoing initiatives concerning energy gateways' standardization or preliminary commercializati on	WP2–Use case deployment and follow up	T2.1-System architecture design	Various ongoing initiatives can be considered for possible reference in RESPOND. For instance, HGI (Home Gateway initiative), OGEMA, etc. and the products being commercialized by some vendors	OGEMA energy gateway will be used for communication with BMS/EMS systems.
MQTT Broker for interconnectivi ty	Interconnectivity architecture of RESPOND shall be based on an integration message brokering layer	WP2–Use case deployment and follow up	T2.1-System architecture design	MQTT broker allows information to flow between disparate applications across multiple hardware and software platforms. In RESPOND, multiple implementations of a single application layer are expected (i.e. multiple gateways) and MQTT middleware provides communication with loose component coupling.	Mosquitto MQTT broker instance has been deployed as a part of RESPOND cloud platform. Currently it is used for data collection from the pilot sites. Once the platform is fully deployed, it will also be used for control actions.

3.2 RESPOND MIDDLEWARE

Based on interoperability requirements and after reviewing current state of the art, RESPOND partners have designed and implemented middleware layer whose main task is integration of RESPOND analytic services with the deployed field level devices, external services, web dashboard and mobile application. RESPOND middleware enables plug-and-play connectivity of deployed sensing and actuating equipment with the rest of the RESPOND system in order to ensure collection, storage and processing of data obtained by sensing devices, analytic services and external systems. The RESPOND middleware is based on an open source publicly available software which is commonly used in other similar systems:

- TICK stack – stack of different standalone software components which enable data collection, processing and storage in an efficient and scalable manner.
- MQTT broker – supports lightweight MQTT protocol suitable for Internet of Things applications with publish/subscribe messaging pattern.

3.2.1 TICK STACK

InfluxData TICK stack represents a widely used scalable open source platform whose aim is to collect, process and store time-series data. It is composed of the following separate readily available modules which can be used individually or together: Telegraf, InfluxDB, Chronograf, and Kapacitor (their function is explained below). These components can be run as services on one or more computers (virtual machines) and communicate among themselves via a standard http protocol with well documented interface.

Telegraf

Telegraf is a metric collection service which can be run independently from other components of TICK stack. The main purpose of Telegraf is to collect metrics from different inputs and write them into a number of outputs. It is a plugin-based component which can be easily configured and extended with custom input/output interfaces. In addition, data collected by Telegraf can be sent in different formats such as: JSON, Value, Nagios, InfluxDB Line protocol, and many others. In RESPOND, Telegraf is used to collect the data sent by field level devices to MQTT broker and also to transform and save such data in Influx database. Therefore, the following two plugins have been configured:

Input MQTT consumer plugin

```
[[inputs.mqtt_consumer]]
  servers = ["tcp://localhost:1883"]
  persistent_session = true
  qos = 2
  connection_timeout = "30s"
  topics = [
```

```
  "+/data",
  ]
  name_override = "demand"
  client_id = "telegraf"
  username = "respond1"
  password = "asdf1234"
  data_format = "json"
```

Output InfluxDB plugin

```
[[outputs.influxdb]]
  urls = ["http://127.0.0.1:8086"]
  database = "telegraf"
  retention_policy = ""
  ## HTTP Basic Auth
  username = "telegraf"
  password = "metricstelegraf"
```

Influx database

InfluxDB belongs to a group of time-series databases which are particularly suitable for storage and retrieval of large amount of time-series data. It is designed to handle high write and query loads as well as down-sampling and deletion of old data. These properties make it the perfect candidate for the storage of data collected from field level devices, such as: energy consumption, temperature, occupancy, etc. InfluxDB uses a SQL-like language for interacting with data – InfluxQL, which allows large amount of data to be retrieved and analysed in real-time. InfluxDB schema can be described as follows:

- Database – represents a logical container for different time-series data, users, retention policies, and continuous queries.
- Series – represent the collection of data that share common measurement, tag set and retention policy.
- Measurement – stands for the data stored in the associated fields.
- Tag keys and values – used to store metadata. Tag keys are indexed so that queries that are performed on tag keys are faster.
- Field keys and values – used to store metadata and the actual measurements and can be of different types: integer, float, string and boolean. Fields are not indexed and each field value is always associated with a timestamp. Queries on field values scan all points that match the specified time range and, as a result, are not performant.
- Retention policy – describes for how long the data will be kept in the database. By default, it is set be infinite (no data are removed)
- Continuous query – an InfluxQL query that runs automatically and periodically within a database. It can be used to aggregate older data in order to save persistent storage.

There exist two ways of accessing the InfluxDB: Command Line Interface (CLI) and HTTP API.

- InfluxDB's command line interface is an interactive shell for the HTTP API. It can be used to write data (manually or from a file), query data interactively, and view query output in different formats. To access the CLI, a user first launches the influxd database process and then launches influx in the terminal. Once the user has entered the shell and successfully connected to an InfluxDB node, the following output can be seen:

```
$ influx
Connected to http://localhost:8086 version 1.7.x
InfluxDB shell version: 1.7.x
```

  The user can now enter InfluxQL queries as well as some CLI-specific commands directly in the terminal.

- The InfluxDB HTTP API provides a simple way to interact with the database, even from other clients, which do not necessarily run on the same computer where InfluxDB is installed. It uses HTTP response codes, HTTP authentication, JWT Tokens, and basic authentication, and responses are returned in JSON. The `/query` endpoint accepts GET and POST HTTP requests. This endpoint can be used to query data and manage databases, retention policies, and users. For instance, in order to query data with a SELECT statement the following request shall be sent:

```
$ curl -G 'http://localhost:8086/query?db=mydb' --data-urlencode 'q=SELECT *
FROM "measurements"'
```

  With the following response:

```
{"results":[{"statement_id":0,"series":[{"name":"measurements","columns":["time
","value","mytag1","mytag2"],"values":[["2017-03-
01T00:16:18Z",33.1,null,null],["2017-03-01T00:17:18Z",12.4,"12","14"]]}]}]}
```

  In a similar manner, The `/write` endpoint accepts POST HTTP requests. This endpoint can be used to write data to a pre-existing database. For instance, to write a point to the database mydb with a timestamp in seconds, the following request shall be sent:

```
$ curl -i -XPOST "http://localhost:8086/write?db=mydb&precision=s" --data-
binary 'mymeas,mytag=1 myfield=90 1463683075'
```

  where the following response is received:

```
HTTP/1.1 204 No Content
Content-Type: application/json
Request-Id: [...]
X-Influxdb-Version: 1.4.x
Date: Wed, 08 Nov 2017 17:33:23 GMT
```

Chronograf

Chronograf is a component of TICK stack which can be used along with other components of the TICK stack to manage databases, users, visualise collected data and create alerting and

automation rules. It supports multi user configuration with access level control. An example of Chronograf dashboard showing data collected from one of the pilot sites is given in Figure 3.

FIGURE 3: CHRONOGRAF DASHBOARD

Kapacitor

Finally, Kapacitor is the last component in TICK stack whose main goal is to support real-time data processing and creating alerts when anomalies in data are detected. It can be configured to process both data streams in real-time or the recorded data in batch fashion. In addition, it can provide any embedded transformation which is offered by InfluxQL query language. Kapacitor can use scripts with lambda expressions in order to configure data transformations and to define logic conditions which can be used to filter data, such as:

```
stream
|from()
.measurement('cpu')
// create a new field called 'used' which inverts the idle cpu.
|eval(lambda: 100.0 - "usage_idle")
.as('used')
|groupBy('service', 'datacenter')
|window()
.period(1m)
.every(1m)
// calculate the 95th percentile of the used cpu.
|percentile('used', 95.0)
|eval(lambda: sigma("percentile"))
.as('sigma')
.keep('percentile', 'sigma')
```

```
|alert()
.id('{{ .Name }}/{{ index .Tags "service" }}/{{ index .Tags "datacenter"}}')
.message('{{ .ID }} is {{ .Level }} cpu-95th:{{ index .Fields "percentile" }}')
// Compare values to running mean and standard deviation
.warn(lambda: "sigma" > 2.5)
.crit(lambda: "sigma" > 3.0)
```

3.2.2 MQTT MESSAGE BROKER

MQTT is a lightweight message transfer protocol based on publish/subscribe messaging pattern. It effectively decouples the sending party (publisher) from the receiving one (subscriber) bringing many benefits:

- One Publisher could send messages to many different Subscribers
- One Subscriber could receive messages from many different Publishers
- It is not necessary that Publishers and Subscribers are aware of each other.
- Implementation of the publisher and subscriber parties independently from each other.

The aforementioned properties of MQTT, make it ideal for distributed systems consisting of a large number of constrained devices, such as Internet of Things applications where small code footprint and limited network bandwidth are usually limiting factors for implementation. In publish/subscribe pattern, publishers and subscribers never communicate directly to each other. Instead, a so-called message broker handles connection between them. The main function of the message broker is to filter all the messages received by different publishers and distribute them only to the interested subscribers. As a result, publishers and subscribers are decoupled and do not have to be connected to the message broker at the same time.

MQTT uses message filtering based on a message subject. Each published message is assigned to a topic, which is then used by the broker to send the message only to the clients which have been previously subscribed to that topic. Topic is a string which consists of one or more topic level separated by a forward slash (e.g. building1/floor2/bedroom/temperature). This allows the receiving party to subscribe only to a part of the topic, and to receive the messages published to topics below that level (e.g. subscribe to topic building1/# to get all the messages originating from that specific building). More details on the communication scheme using MQTT protocol which will be employed by RESPOND, is provided in Section 3.3 on Canonical Data Model.

3.3 CANONICAL DATA MODEL AND COMMUNICATION PROTOCOL

In this section, we will provide a specification of protocol and data format for message exchange between RESPOND cloud platform and home automation devices provided by Develco and Energomonitor. In order to ease the integration of solutions provided by technology providers, we

propose a canonical data model. In addition, the CDM will consider the custom adapter developed by FENIE, to be used to collect data regarding domestic hot and cold water consumption.

3.3.1 MEASUREMENTS COLLECTION

In this section, we provide a specification of data format that shall be used for messages sent from sensors, smart meters and custom components to the RESPOND cloud platform by using MQTT protocol.

MQTT Payload

MQTT payload is formatted as a JSON array of the objects where each object is specified as follows in Table 2:

TABLE 2: MQTT MESSAGE PAYLOAD FIELDS FOR MEASUREMENTS

Field name	Field type	Additional description
timestamp	integer	UNIX time in milliseconds
value	float/boolean	Value of measurement without the unit (e.g. 24.0)
measurementIndex	integer	An index of the measurement (set to 1 for scalar value; 1,2,3 for e.g. 3-phase measurement)
deviceID	string	Unique ID of a particular sensor or smart meter. We propose to add a vendor specific prefix (e.g. DEV-xxxxxxxxx or ENE-xxxxxxxxx). Develco's sensor shall add DEV prefix and Energomonitor's sensor shall add ENE prefix to their proprietary hexadecimal serial number. (e.g. DEV-4567ACF79454AF6548 or DEV-4567789454AF6548). Fenie, can add FEN prefix to the identifier which will be used by their custom adapter (e.g.FEN-001)
measurementID	string	a string identifying the measurement type provided by the given sensor: • demand – power consumption in [W] • energy – accumulated energy consumption in [Wh] • acfrequency – ac frequency in [Hz] • voltage – ac voltage in [V] • current – ac current in [A] • onoff – true if smart plug is switched on, false otherwise

		• occupancy – true if occupancy sensor detects presence, false is occupancy sensor doesn't detect presence • alarm – alarm true if door/window is open or if occupancy detects presence and false vice versa • illuminance – light illuminance in [Lux] • temperature – air temperature in [°C] • humidity – relative humidity in [%] • battery – battery voltage in [V] • networklinkstrength – network link strength 0-100. Applicable to Develco devices • rssi - RSSI (signal strength) in [dB]. Applicable to Energomonitor devices • co2 – data sent by CO2 sensor in [ppm] • voc – data sent by VOC sensor in [ppb] • dhwc – domestic hot water cumulative consumption [m^3] • dcwc – domestic cold water cumulative consumption [m^3] • temperature_setpoint – thermostat setpoint in [°C] • heat_energy – heat energy in [Wh] • inlet_flow_temperature – measured inlet temperature in [°C] • return_flow_temperature – measured return temperature in [°C]

MQTT topics structure

Messages shall be published to the topics that are organized as follows:

GATEWAY_ID/data

Where:

• GATEWAY_ID – Unique ID of a gateway device = Manufacturer's prefix + 16HEX serial number (e.g. ENE-6A5F000054876542, DEV-4567789454AF6548 or FEN-000123)

Messages shall be published with QoS 0.

Note: GATEWAY_ID and deviceID is expected to be mapped to corresponding entities of Ontology which should also reflect system/device hierarchy (e.g. device – entity – feature hierarchy from EEBus SPINE).

3.3.2 CONTROL ACTIONS

In this section, we provide a description of protocol and message format for control actions that will be issued from RESPOND platform towards the actuator deployed in the field (smart plug, smart cables, smart relays and smart thermostats).

MQTT payload format

MQTT payload for control actions is formatted as a JSON object with the following fields, as shown in Table 3:

TABLE 3: MQTT MESSAGE PAYLOAD FIELDS FOR CONTROL ACTION

Field name	Field type	Additional description
deviceID	string	A unique ID of an actuator device that the service will send the command to. Develco's device shall add DEV prefix and Energomonitor's sensor shall add ENE prefix for their proprietary hexadecimal serial number. (e.g. DEV-4567789454AF6548 or ENE-6A5F000054876542)
requestID	string	A unique identifier which will be used to identify the response for the given request (explained below). It is composed of two parts (e.g. OPT-156): • Requester prefix, e.g. OPT for optimizer • An increasing integer 1,2,3….
value	boolean / float	• true if smart plug is to be turned ON • false if smart plug is to be turned OFF • value for smart thermostat to set its temperature (e.g. 24.0) • null if a service just needs to read a status of a smart plug or smart thermostat without changing its state

MQTT Protocol for control actions

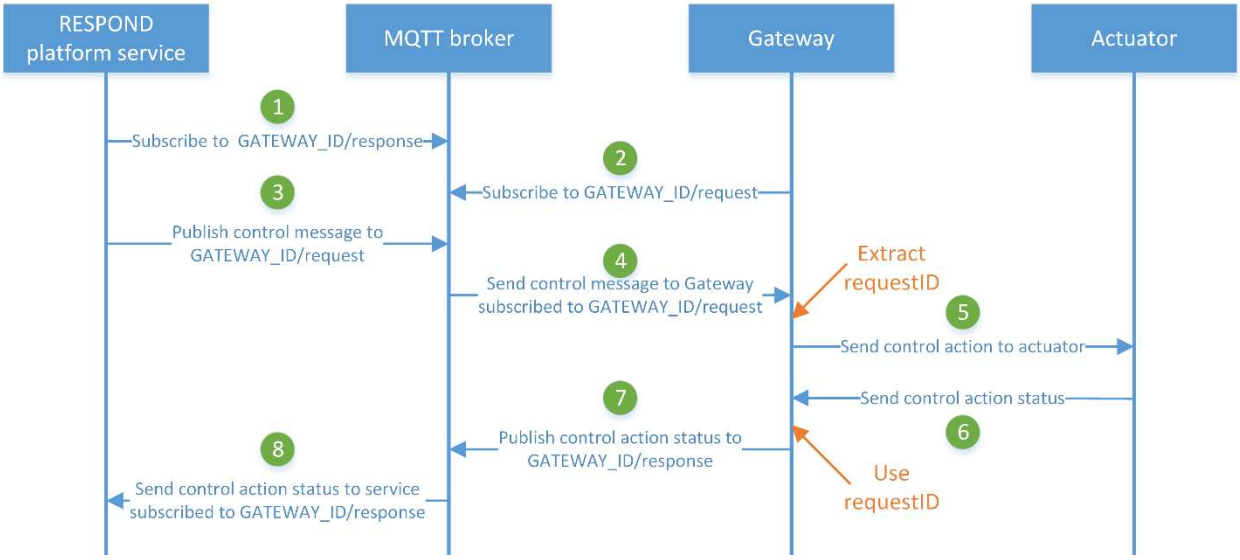

FIGURE 4: MQTT BASED PROTOCOL FOR ACTUATOR CONTROL

The MQTT based protocol used for perform control actions on actuators can be described as follows (see Figure 4 above):

1. Service (e.g. RESPOND service such as Optimization) subscribes to topic: GATEWAY_ID/response where GATEWAY_ID is a unique ID of a gateway where specific actuator is connected to. Manufacturer's prefix + 16HEX serial number (e.g. DEV-4567789454AF6548)
2. Gateway (Physical device from DEV) subscribes to topic: GATEWAY_ID/request where GATEWAY_ID is unique ID of a specific gateway, with the corresponding manufacturer's prefix as described in the previous step. This topic structure is necessary so that gateway can receive requests to turn on/off the actuator devices (smart plug, smart cable).
3. Service publishes the MQTT message for control action to the topic GATEWAY_ID/request with the payload structured according to the specification provided above in Table 3.
4. Gateway receives the control message from the MQTT broker since it has already been subscribed to the topic GATEWAY_ID/request in Step 2, and extracts the requestID field from the message payload.
5. Gateway sends the control action to actuator according to the value field set in the payload of the MQTT message.
6. Actuator sends the control action status to the gateway.
7. Gateway uses previously extracted requestID field and publishes the response in the MQTT message to the topic GATEWAY_ID/response, where the published message shall have the format according to specification provided in Table 4:
8. MQTT broker sends the control action status to the service since the service has already been subscribed to the topic GATEWAY_ID/response in Step 1.

TABLE 4: MQTT MESSAGE PAYLOAD FIELDS FOR CONTROL ACTION STATUS

Field name	Field type	Additional description
timestamp	integer	UNIX time in miliseconds
value	boolean / float	Set to the current value (new value if it is changed by control action) • true is smart plug is turned ON • false if it is turned OFF • value, if temperature setpoint for smart thermostat is set correctly • null if no action has been performed due to an error.
status	integer	Set to one of the following codes: • 0 – action performed correctly • 1 – invalid request - invalid JSON, missing field, invalid field value, etc. • 2 – gateway unreachable • 3 – actuator device (e.g. smart plug) is not reachable by gateway • 4 – actuator device is not paired with the gateway
deviceID	string	A unique ID of a particular actuator device with the corresponding manufacturers prefix (DEV for Develco)
requestID	string	The same field that has been extracted from the received message payload in Step 4.

Both control action request and control action response messages are published with QoS=0.

4. DEPLOYMENT OF RESPOND INTERFACES

In this section, our aim is to provide the information on custom interface development tasks which have been performed by different RESPOND project partners. In particular, customization of hardware and software components of field level devices has been performed by technology providers Develco and Energomonitor. Besides, Tekniker was in charge of custom implementation of OGEMA gateway. Finally, Fenie Energia and Dexma have developed water consumption and external interface adapters which feed the data from respective external services into RESPOND cloud platform.

4.1 OGEMA ENERGY GATEWAY (TEK)

OGEMA [4] is a software framework which aims to provide a hardware independent execution environment for energy management applications. It acts as a firewall between the public and private communication systems, allowing only certain interactions between the systems as defined by the gateway configuration. Applications installed in OGEMA can obtain access to customer devices, user displays, smart meters, measuring data, as well as data provided by external market participants, like tariff information or grid parameters. The goal is to provide a platform for Smart Building and Smart Home applications supporting the full range of Smart Grid applications at the customer side with a single efficient hardware platform, which features all the necessary communication connections [Figure 5]. New devices and functionalities can be connected and added in a “plug&play” manner with minimum user interaction

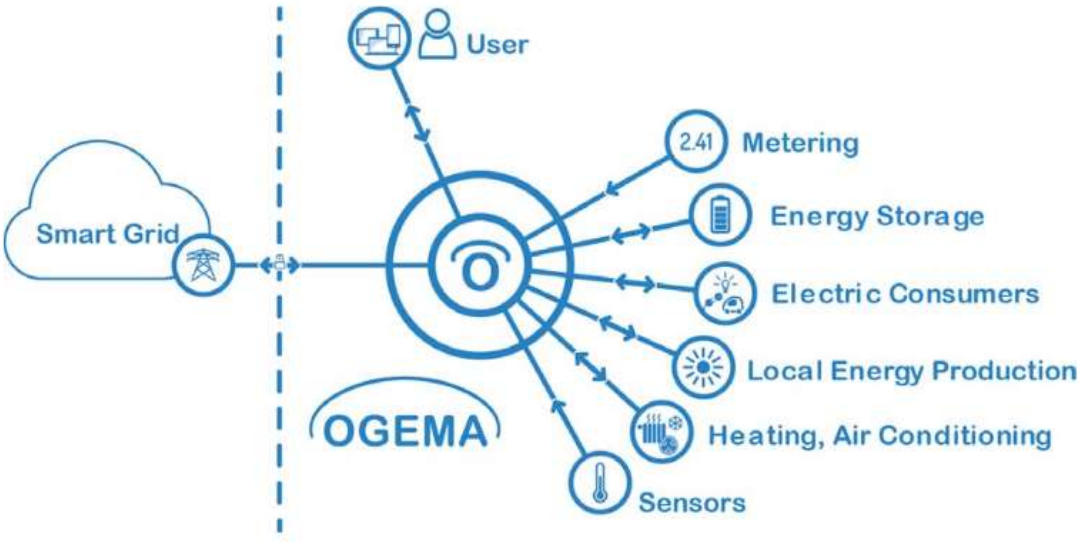

FIGURE 5: OGEMA APRROACH

The framework follows a modular architecture, using Java OSGi for deploying its components. OSGi is a lightweight approach oriented to embedded or small devices, where resources are more limited. In OSGi, the different software components are packed into units called ‘bundles’, which

are only activated when needed. Due to being Java, it can be deployed in any hardware capable of running a Java Virtual Machine (JVM).

These components can be classified in:

- OGEMA services: Core functionalities of the framework. Resource administration, data modelling, access control, logging functionalities, web servers, component management…
- Communication Drivers: The communication drivers are used to connect to the different hardware devices. Diverse home automation and industrial standards are already integrated in the OGEMA framework, and new one can be added by following the guidelines.
- Applications: Applications are the higher-level components that will use the OGEMA services and Drivers to give a specific user-oriented functionality. For example: Smart building: controlling different real-world assets using the information of different sources (sensors, metering, available energy sources…)

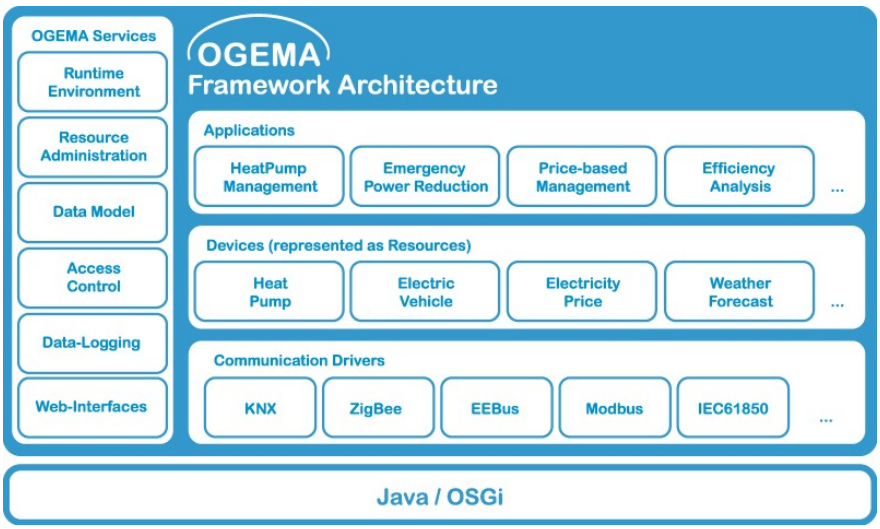

FIGURE 6: OGEMA FRAMEWORK ARCHITECTURE

The idea was to reuse the existing functionalities and drivers of the framework and develop and deploy a new application, only developing a single component more. This new application would detect changes in the values of device connected with KNX protocol and publish them in the RESPOND's MQTT broker.

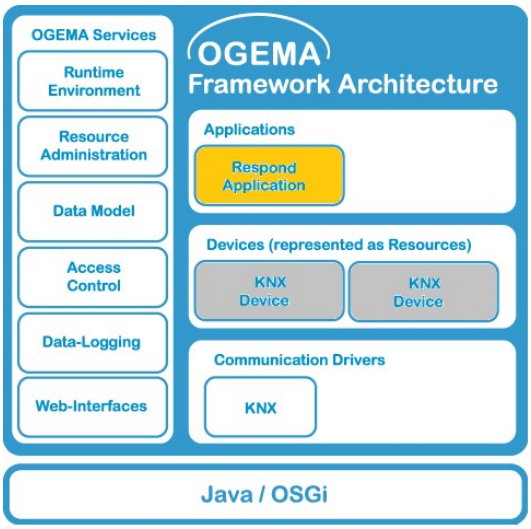

FIGURE 7: OGEMA GATEWAY IN RESPOND

4.1.1.1 ALTERNATIVE FRAMEWORK: OPENMUC

During the early testing phases, it was noticed that OGEMA have some shortcomings that could affect the deployment in the Madrid pilot:

- Reduced KNX datatypes: KNX has defined a wide range of datatypes, which define the size of the data records and the measuring units. The OGEMA framework only had a reduced set of them available. Not knowing if these datatypes would be enough for the pilot, it was decided to update the driver.
- Driver/framework not fully agnostic: At first it was noticed that the KNX driver was not a generic one, but rather oriented to the reduced KNX datatype set. It would check if the ranges where the expected for those specific types and even transform the data, like switching temperature from Celsius to Kelvin. After trying to modify it by adding additional datatypes, it was noticed that not only the driver was too oriented to the reduced datatype set but other components as well, like the framework's inner model. Adding new datatypes was not an easy task as it involved changing many of the core components of the framework.

TABLE 5: AVAILABLE KNX TYPES IN OGEMA

OGEMA type (sensor)	KNX DPT		OGEMA type (actuator)	KNX DPT	
MotionSensor	1.011	1-bit status	OnOffSwitch	1.001	1-bit switching
OccupancySensor	1.011	1-bit status	ElectricDimmer	5.001	8-bit unsigned
TouchSensor	1.011	1-bit status	ThermalValve	9.001	16-bit float

OGEMA type (sensor)	KNX DPT		OGEMA type (actuator)	KNX DPT	
ElectricPowerSensor	12.001	4x8 bit unsigned			
LightSensor	9.004	16-bit float			
TemperatureSensor	9.001	16-bit float			

Due to these reasons another framework was targeted as an alternative, openMUC [5]. It is a similar framework which also uses the Java OSGi methodology and a "core/drivers/applications" based architecture. While OGEMA is more centered to energy management solutions, openMUC is more open to all kind of monitoring/control systems. It includes different protocols and drivers, including KNX. This driver does not limit the KNX datatypes and follows a more generic approach, which can also be said from the rest of this framework components.

OGEMA has better web-based user interfaces, allowing a more complex management of the resources, bundles and components. On the other hand, openMUC's web interfaces don't offer that many functionalities, relying on more simple interfaces.

Another reason for choosing this framework was that the developer team was already familiar with it.

4.1.2 ENERGY GATEWAY DEVELOPMENT

Thanks to the functionalities of the OGEMA framework, the development team is ready to deploy it. In order to use it in RESPOND, an application for data collection was needed to be developed. After checking the pilot configuration, there was some concern about the framework not covering the pilot requirements and an alternative was developed for the openMUC, in order to avoid possible future problems.

4.1.2.1 MQTT CDM PUBLISHER

An application was developed using the pre-existing ones as references. The application would:

- Listen to the changes in the monitored devices. OGEMA services offer a functionality to add custom listeners that would be executed each time the data values of a monitored resource change.
- Create the CDM payload. Using the defined canonical data model, a message is built, filling up the necessary attributes: deviceID, measurementID, timestamp and detected value change.

- Publish the CDM message to the RESPOND's platform MQTT broker. On activation, the application will instantiate a MQTT client. Once the data change is detected, the CDM message is created and published to the configured topic.

The application can be configured at two levels:

- MQTT client configuration: host, port, user, passwords, quality of service (QoS), topic
- For each individual measurement: KNX group address, KNX datatype, CDM deviceID, CDM measurementID.

Measurement or datapoint configuration can be achieved through the web-used interfaces or by directly editing configuration files. MQTT client can only be configured editing the properties files. This initial application was later adapted to the openMUC framework

4.1.2.2 KNX DRIVER UPDATE

Initially OGEMA's KNX driver is ready to be used, as long as these conditions are met:

- The monitored datapoints are of the OGEMA available KNX datatypes.
- The KNX network has a reachable IP bridge. KNX networks can be interfaced by different means (TP1 cable, Ethernet, USB…). The driver requires that a node/device of the KNX networks has a defined IP, which will be used to be accessed.

OGEMA KNX was updated in order to be able to handle additional KNX datapoints, but without the actual physical devices and pilot configuration it could not be tested properly, which was one of the reasons of considering an alternate framework.

The openMUC KNX driver didn't need any change in order to be used under the same conditions, but an additional functionality was added, as it was considered appropriate for easing the deployment. This functionality allowed connecting to a KNX network using a USB bridge, instead of an IP one, as it is further explained in the Deployment section.

4.1.3 DEPLOYMENT

This section explains deployment related issues, like connection to physical devices in the KNX network, hardware requirements and pilot final configuration.

4.1.3.1 CONNECTING TO KNX NETWORK

One of the issues with the deployment was connecting to the existing KNX networks, as it didn't have an IP bridge, necessary for using the KNX drivers. One alternative solution was to use an already existing software called Calimero server.

Calimero is a library used for KNX implementation on Java. It's open source and in its repository, different software can be downloaded, one of which is the Calimero server. It's a configurable

server that allows bridging different KNX connections and even creating virtual devices for testing.

The server was configured to bridge a USB connection, creating an IP endpoint that could be accessed by the KNX driver. The OGEMA deployment was tested against real world devices using a USB to KNX adapter and a small KNX network with a switch for data input and an actuator with two LEDs. The Calimero server and the OGEMA gateway were installed in the same PC.

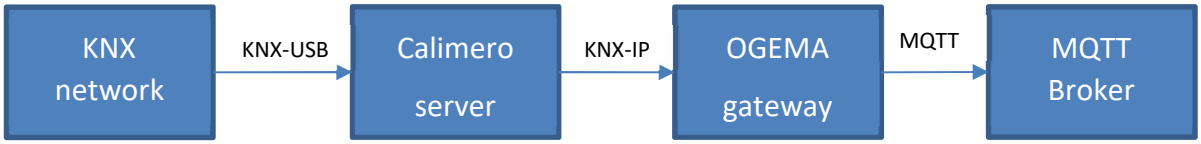

FIGURE 8: DATA EXCHANGE (OGEMA GATEWAY)

After openMUC was considered as an alternative in order to deal with the missing KNX datatypes, initial tests showed that some conflicts would happen if the Calimero server was deployed in the same device as the openMUC gateway. In order to solve this openMUC's KNX driver was modified enabling to connect to KNX-USB directly. This meant that the Calimero server was not needed with openMUC.

FIGURE 9: DATA EXCHANGE (OPENMUC GATEWAY)

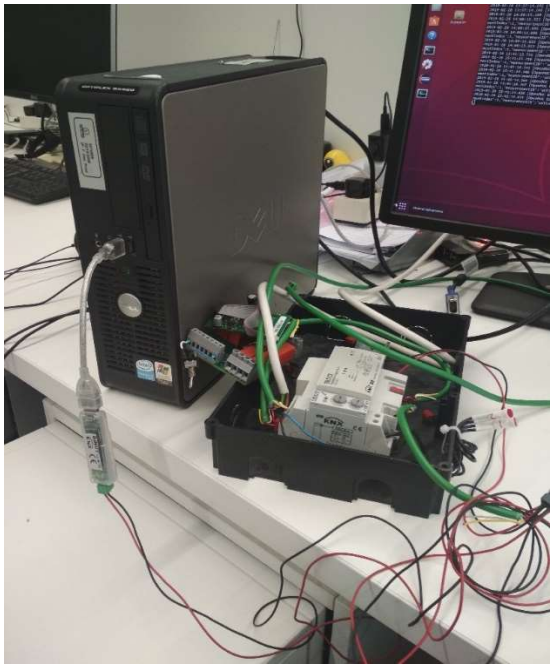

FIGURE 10: SMALL KNX NETWORK FOR TESTING

4.1.3.2 HARDWARE REQUIREMENTS

These are the requirements for deploying the energy gateway and its components:

- Internet connection.
- Enabled ports:
  - 8080/8443 HTTP/HTTPS fot OGEMA web interface.
  - 3671 Default KNXnet/IP
- Java 8 or higher (OGEMA/openMUC gateway)
- Java 11 (Calimero server)
- Maven
- Gradle

Both OGEMA and openMUC gateways use Maven and Gradle to make the deployment of the framework easier, as well as to help develop new components. Nevertheless, it is not strictly necessary to deploy them, and in this case, additional launching scripts would be needed to be defined instead.

Java allows deploying the solution in different operating systems. The gateways have been tested in Linux and Windows systems. One of the initial ideas was to deploy the gateway in a Raspberry PI or similar device. However, this solution is not possible with the OGEMA approach, as Calimero server needs Java 11, which as now is not compatible with Raspberry PI device.

Using the openMUC approach, the gateway deployment was tested on a Raspberry Pi. Although the gateway seemed to work correctly, more resilience tests were pending, and this deployment configuration was not used in the Madrid pilot.

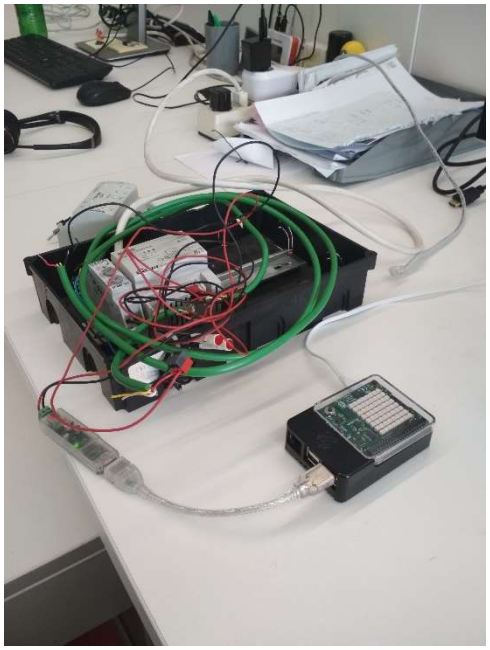

FIGURE 11: TEST SETUP

4.1.3.3 FINAL CONFIGURATION IN MADRID PILOT

The Madrid pilot site included a solar thermal installation that used the solar panels to pre-heat boiler water, reducing the energy costs. This installation is monitored by different sensors and counters connected to a KNX network through a Siemens RMS705B digital control unit.

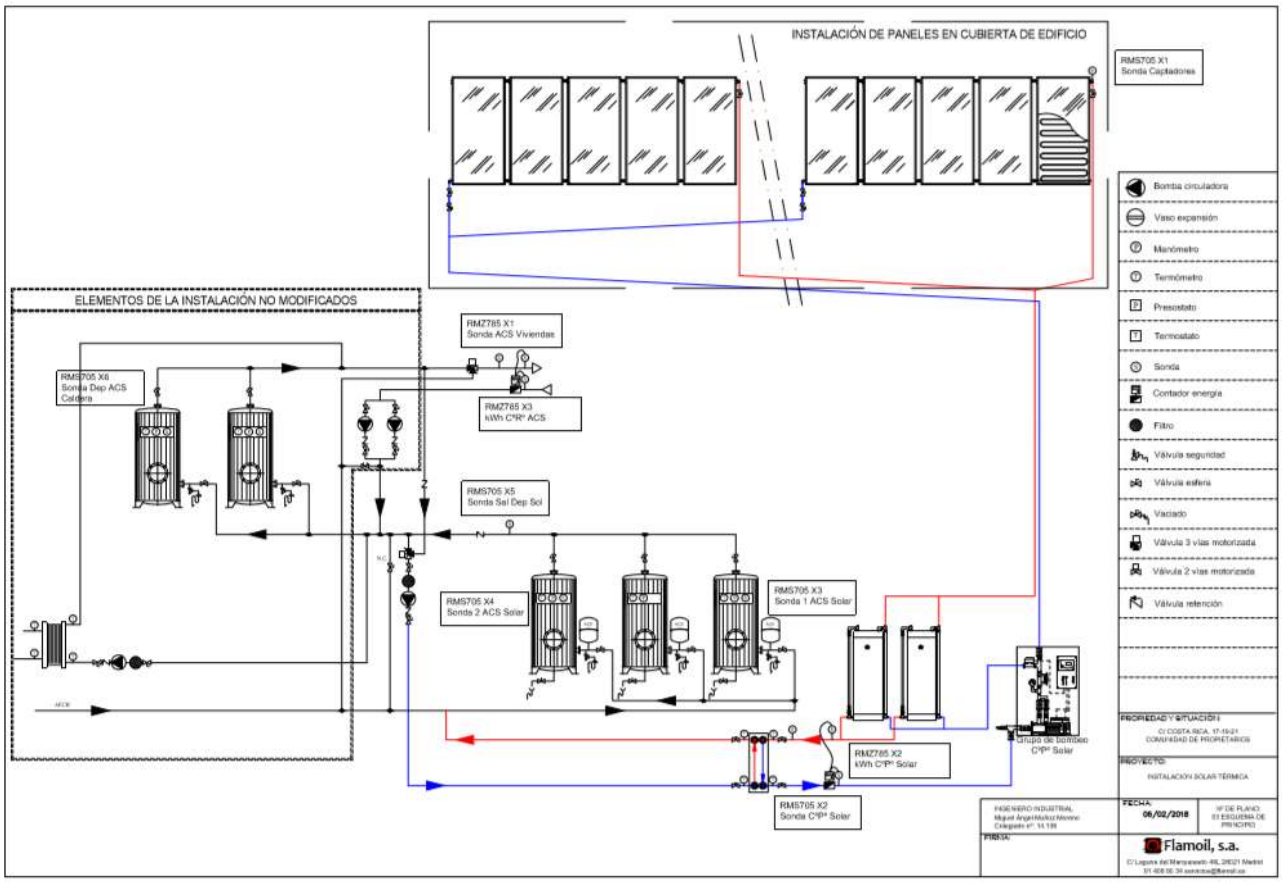

FIGURE 12: MADRID PILOT SITE CONFIGURATION

The configuration of the gateway used for deployment is as follows:

- PC with Linux Ubuntu 18.04
- Connected to KNX network with a Zennio KNX-USB bridge
- Connected to Internet through Ethernet connection to a router
- openMUC gateway configured to be launched on startup
- TeamViewer installed in order to provide remote support

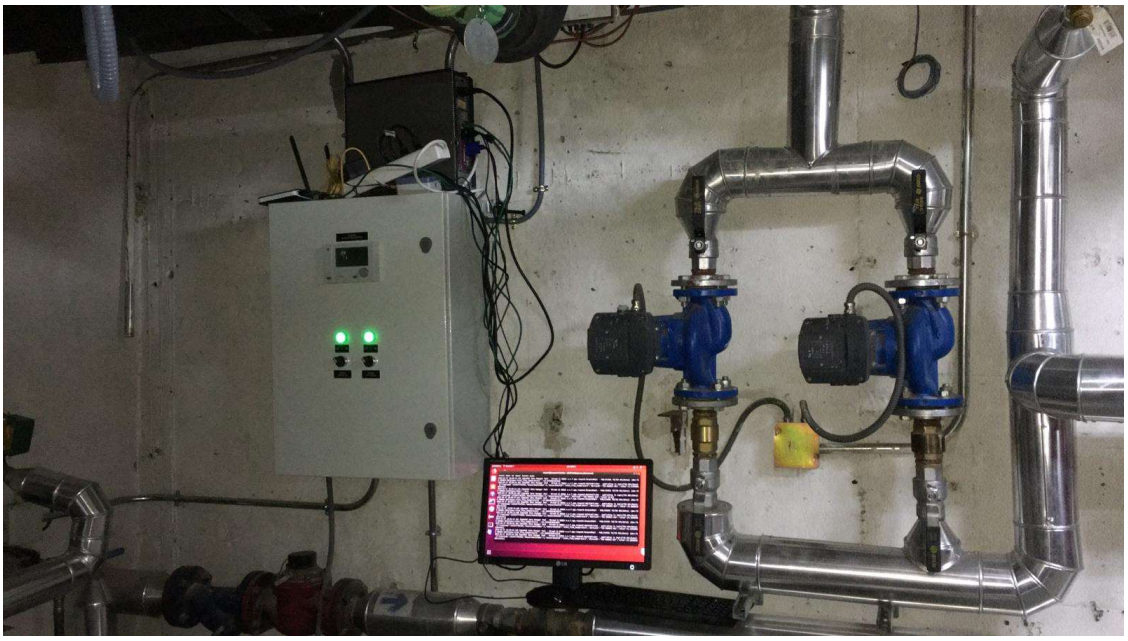

FIGURE 13: GATEWAY PC WITH KNX SWITCHBOX

During the deployment of the gateway, there were some problems due to the difference in the KNX configuration used by the Siemens Synco devices and the KNX driver from the gateway. Synco devices can be configured in two modes: LTE (easy mode) and S-Mode (system mode). LTE mode is used to quickly connect and use Synco devices to a KNX network and is the one that they use by default, while S-Mode is a more advanced configuration mode used when connecting to other KNX entities outside the Synco network, such as the gateway's KNX driver.

KNX devices were configured in Madrid with LTE, and during the deployment none of the presents had the expertise to change it to S-Mode. Other functionalities were tested such as the configuration of communication (ensuring the messages were correctly published in the main broker) and other resilience tests, such as disconnecting the Wi-Fi communications and sudden shutdowns due to power loss.

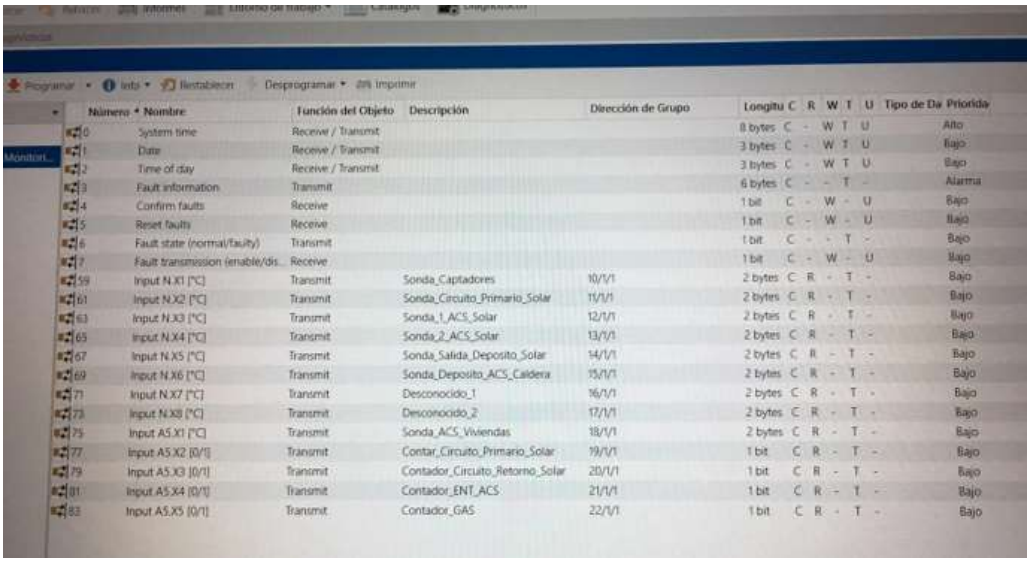

FIGURE 14: RMS705B CONFIGURATION IN ETS

Later on, the RMS705B was properly configured with ETS (Engineering Tool Software) to be used in S-Mode. This enabled the openMUC gateway to detect data changes, format them in CDM and send it to the RESPOND MQTT broker. Temperature probes send data in a decimal format, depicting the temperature in ºC. Counters send binary data, sending a message whenever they were activated. The next table shows the full list of the monitored elements:

TABLE 6: SOLAR SENSORS CONFIGURATION

DEVICE	RMS705B	KNX -DTP	CDM Topic	CDM DeviceID	CDM MeasurementID
Probe - Panels	N.X1	10/1/1 (9.001)	TEK-001	TEK-000001-001	inlet_flow_temperature
Probe – Primary Solar Circuit	N.X2	12/1/1 (9.001)	TEK-001	TEK-000001-002	inlet_flow_temperature
Probe - Solar ACS 1	N.X3	12/1/1 (9.001)	TEK-001	TEK-000001-003	inlet_flow_temperature
Probe - Solar ACS 2	N.X4	13/1/1 (9.001)	TEK-001	TEK-000001-004	inlet_flow_temperature
Probe - Solar Tank output	N.X5	14/1/1 (9.001)	TEK-001	TEK-000001-005	inlet_flow_temperature
Probe - Boiler ACS Tank	N.X6	15/1/1 (9.001)	TEK-001	TEK-000001-006	inlet_flow_temperature
Probe – Household ACS	A.X1	18/1/1 (9.001)	TEK-001	TEK-000001-007	inlet_flow_temperature
Counter – Primary Solar Circuit	A.X2	19/1/1 (1.001)	TEK-001	TEK-000001-008	heat_energy
Counter – Return Solar Circuit	A.X3	20/1/1 (1.001)	TEK-001	TEK-000001-009	heat_energy
Counter - ENT ACS	A.X4	21/1/1 (1.001)	TEK-001	TEK-000001-010	dhwc

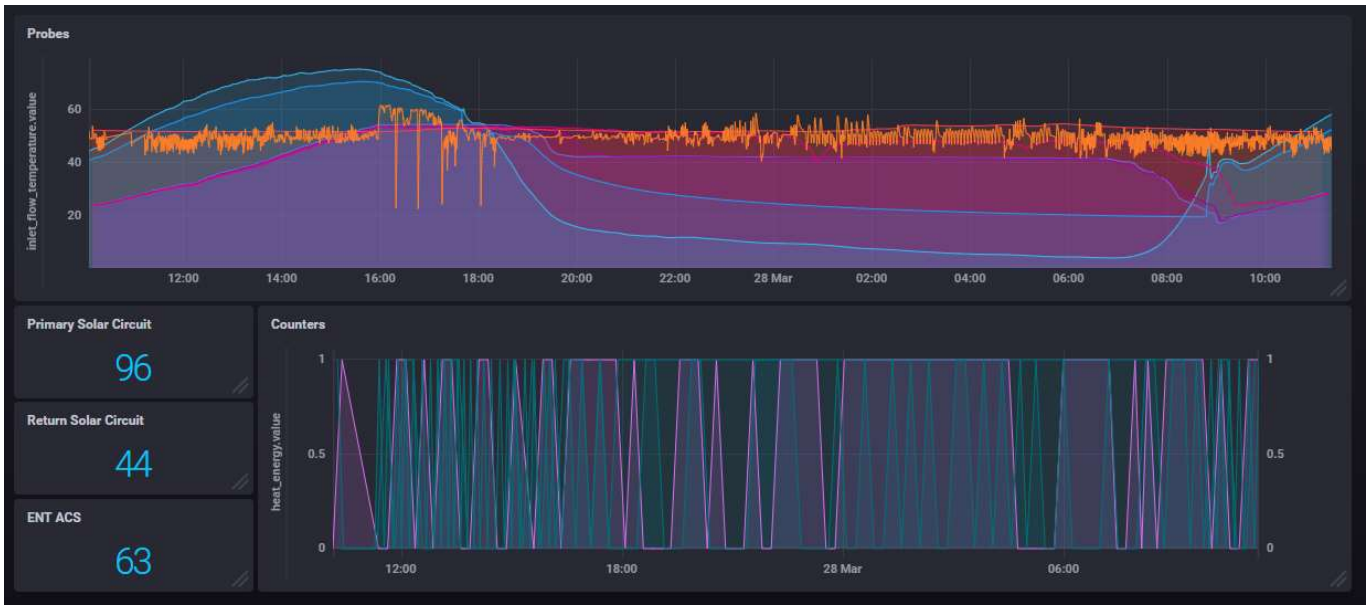

FIGURE 15: COLLECTED DATA DASHBOARD

4.2 DEVELCO CUSTOM FIRMWARE (DEV)

Figure 16 presents the devices available in Develco portfolio which encompasses a variety of smart metering equipment and corresponding wireless communication interfaces and is fully compliant with the RESPOND cloud-based platform. Develco's platform is able to measure the consumption of electricity, water, heat and gas thus enabling the RESPOND system to infer the User's consumption habits. In addition, it enables monitoring of key indoor environment parameters (temperature, relative humidity, etc.) and other phenomena (individual device consumption meter, occupancy, etc.) that will be used as an input for the RESPOND analytic services.

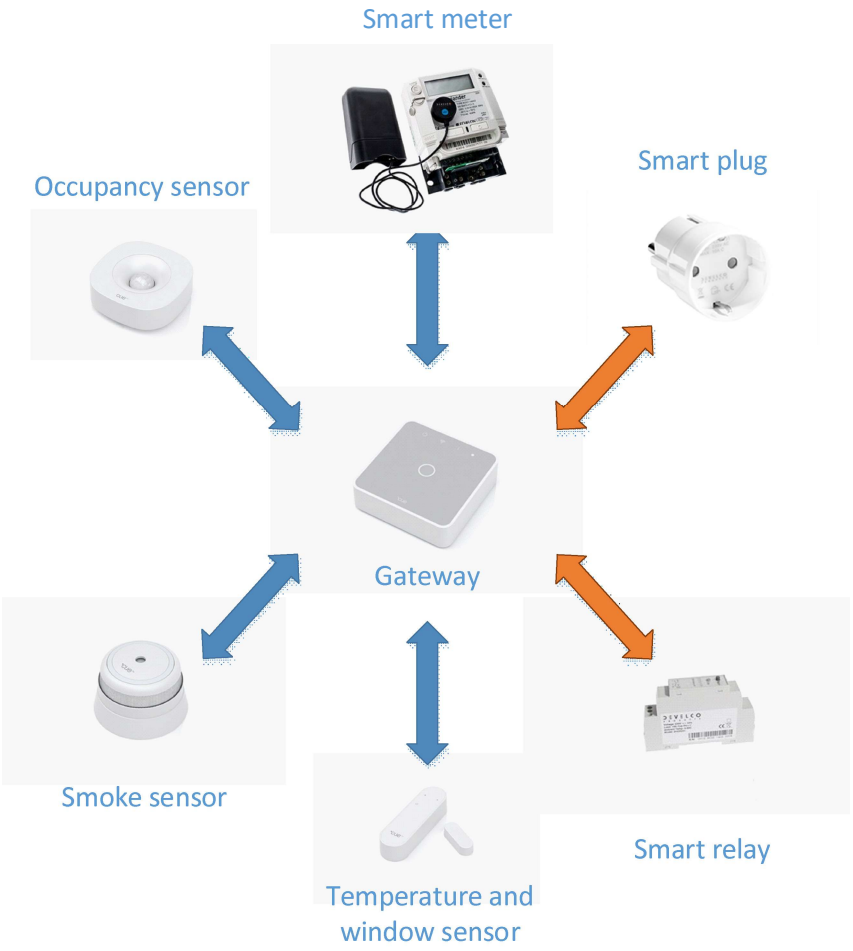

FIGURE 16: DEVELCO DEVICES

The Develco gatway uses publish/subscribe messaging pattern to send data to the RESPOND platform. It must be noted that Develco gateway communication protocol is well aligned with the RESPOND Canonical Data Model and the corresponding protocol, which required only a few customization tasks aimed at interoperability with the rest of RESPOND platform:

- Proprietary MQTT topic structure used with original Develco firmware was simplified from:

`Gateway_id/update/zb/dev/1/ldev/alarm/data/alarm` to `Gateway_id/data`.

- The structure of MQTT messages was changed to be aligned with the one proposed in CDM:

Original message	RESPOND CDM message

{
"key":"alarm",
"name":"Alarm",
"type":"boolean",
"value":false
}
	[{"timestamp":1551830400000,
	"value":false,
	"measurementIndex":1,
	"deviceID":"FEN-44481269",
	"measurementID":"alarm"}]

- The communication protocol used for control actions (e.g. turn on/off the smart plug), was adjusted to be in line with the proposed CDM

All the customizations were performed by issuing new firmware image, which was used to update the gateways.

4.3 ENERGOMONITOR CUSTOM ADAPTER

To understand how the Energomonitor custom adapter is built and integrated into the Energomonitor backend, it is first necessary to discuss the Energomonitor system in general.

4.3.1 THE ENERGOMONITOR SYSTEM

The Energomonitor system consists of integrated hardware and software parts, as shown in Figure 17.

4.3.1.1 ENERGOMONITOR HARDWARE

The hardware part of the Energomonitor system consists of sensors connected to a gateway. The sensors communicate with the gateway using a custom radio protocol. The gateway is connected to the internet using an Ethernet connection. The radio protocol protocol between the sensors and a gateway is packet-based and it allows two-way communication. There are several packet types defined, which enable:

- Receiving measured data from the sensor
- Sending commands to the sensor (reading and writing of configuration parameters and triggering actions such as turning a switch on/off or displaying values on a display)

The gateway communicates with the Energomonitor backend using a custom JSON-based data protocol built on top of MQTT. The gateway mostly just translates radio packets into JSON messages and back with minimal additional processing. The data protocol is optimized for ease of this translation and for minimal bandwidth consumption.

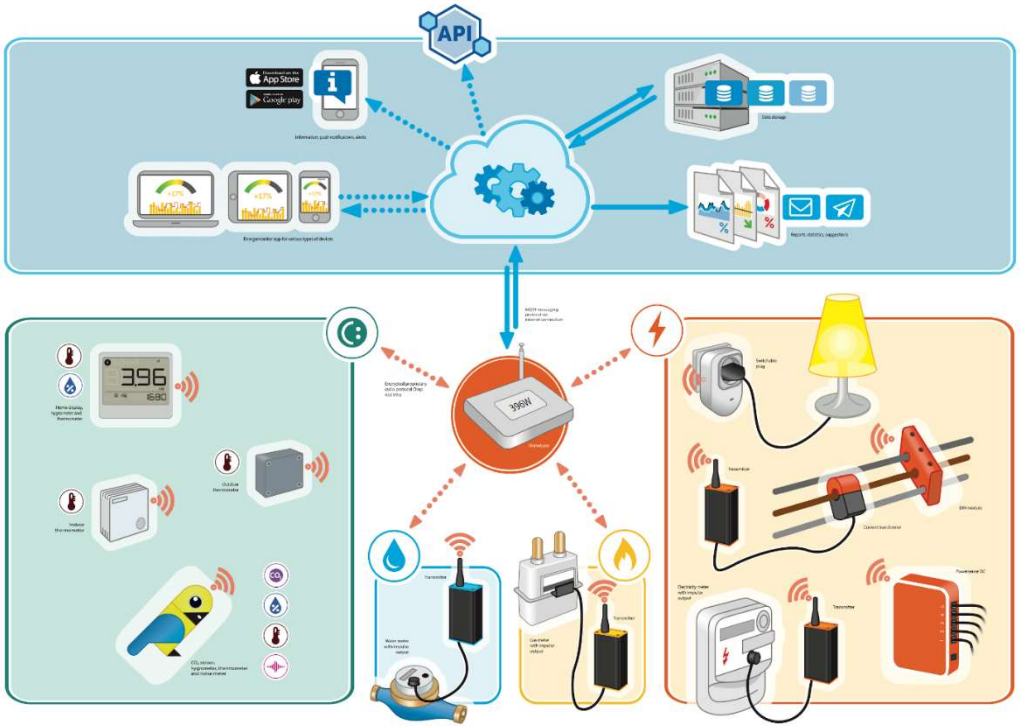

FIGURE 17: THE ENERGOMONITOR SYSTEM

4.3.1.2 ENERGOMONITOR BACKEND

The Energomonitor backend consists of multiple components. In this section, we describe only the ones relevant for later discussion. See Figure 18 for an overview.

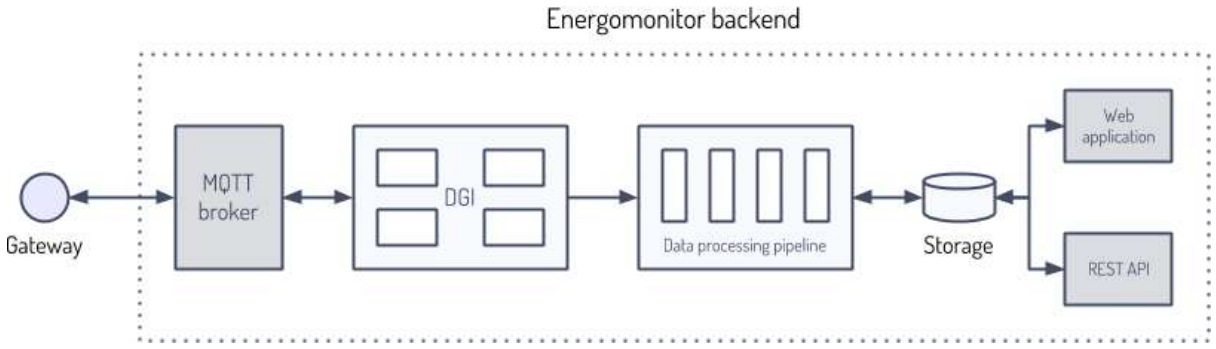

FIGURE 18: ENERGOMONITOR BACKEND (RELEVANT PARTS, SIMPLIFIED)

MQTT broker

The MQTT broker all gateways connect to.

Device gateway interface (DGI)

An abstraction built on top of the MQTT broker that presents a consistent interface for communicating with the devices to the rest of the system. There are several subcomponents:

- **Reader** – a streaming service that reads messages with data measured by sensors from the MQTT broker and writes them into a Kafka topic
- **Processing** – a streaming service that reads data measured by sensors from the Kafka topic where it was written by Reader, transforms it into format used by other components (mainly the data processing pipeline), and writes resulting data packets into another Kafka topic
- **Commander** – a HTTP service that handles sending commands to devices

Data processing pipeline

A pipeline that processes incoming data (filtering, aggregation, overall consumption computation, etc.).

Storage

A system for storing measured data. It is built using InfluxDB, Redis, S3, and PostgreSQL.

Web application

The main interface to the Energomonitor system for its users. It allows users to see measured data, send commands to devices, and manage them.

REST API

The main interface to the Energomonitor system for developers. It allows developers to download measured data, but it currently doesn't allow to send commands to devices and to manage them.

4.3.2 INTEGRATING ENERGOMONITOR DEVICES WITH THE RESPOND PLATFORM

The Energomonitor devices had to be integrated with the RESPOND platform so that they can send it measured data and can receive commands issued by it. The question was, how exactly this integration should be done.

Like the RESPOND platform, Energomonitor gateways use MQTT for all communication. However, Energomonitor's proprietary protocol is different from RESPOND CDM. There were three possible ways how to overcome this:

- Modify Energomonitor gateway firmware to implement RESPOND CDM
- Use Energomonitor REST API
- Develop a custom backend adapter

4.3.2.1 MODIFYING ENERGOMONITOR GATEWAY FIRMWARE TO IMPLEMENT THE RESPOND CDM

Modifying gateway firmware looked like the most straightforward solution. However, it has several significant drawbacks:

- The firmware is written in C, which makes it relatively hard and costly to modify.
- The modified firmware would be developed only for the RESPOND project. This would create overhead during device production and complicate device management. It would also make the gateways more complicated to replace in case of failures.
- The gateway has limited memory and processing power. Energomonitor's MQTT data protocol is designed with that in mind, but RESPOND CDM implementation would likely be more complex and it was possible it would hit one or both of these limits.

4.3.2.2 USING ENERGOMONITOR REST API

Using the REST API looked like another possible solution. Again, it had several significant drawbacks:

- The API had no means for device management.
- The API had no means for sending commands to devices.
- The API was not a streaming API, requiring periodic polling.
- The API would not produce data in the RESPOND CDM format, requiring development of an additional adapter.

4.3.2.3 DEVELOPING A CUSTOM BACKEND ADAPTER

The idea of a custom backend adapter was that it would integrate with Energomonitor's DGI on one side and RESPOND's MQTT broker on the other side. It would simply translate both measured data and issued commands between these two.

This sounds like an unnecessary complicated way, especially when compared with gateway firmware modification. But the analysis revealed that building and maintaining the backend adapter would be several times cheaper and faster than gateway firmware modification. It would also be more flexible solution, because changing backend code is much easier than changing firmware of deployed gateways. For these reasons, it was decided to go ahead with the adapter.

4.3.3 ADAPTER ARCHITECTURE

As described above, the adapter is a translation layer between Energomonitor's DGI on one side and RESPOND's MQTT broker on the other side. Note that it would have been technically possible to avoid DGI and implement the adapter as a direct MQTT-to-MQTT translator, but that would mean reimplementation of significant parts of Energomonitor's MQTT protocol and various DGI features.

It was decided to split the adapter into two separate services, each handling one aspect of the communication:

RESPOND Data Bridge

Receives data measured by the sensors from DGI and sends them to RESPOND's MQTT broker using the CDM format.

RESPOND Control Bridge

Receives commands from RESPOND's MQTT broker in the RESPOND CDM format, sends them to gateways through DGI, receives responses, and sends these back to RESPOND's MQTT broker in the RESPOND CDM format.

The reason for splitting the components is that both communication pathways are completely independent, both in the Energomonitor backend and in RESPOND CDM. In this situation, it was better to split the functionality into two services, each of which would have only one responsibility and be simpler as a result. The overall situation is shown in Figure 19.

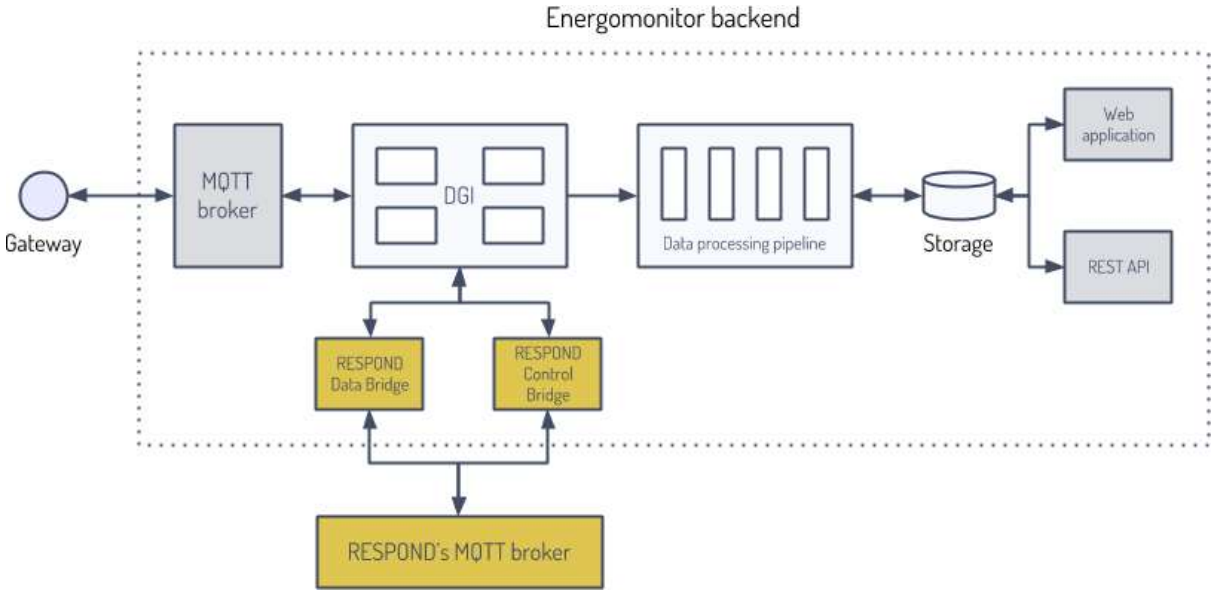

FIGURE 19: INTEGRATION OF THE ADAPTER INTO ENERGOMONITOR BACKEND

4.3.4 ADAPTER IMPLEMENTATION

At this point, only RESPOND Data Bridge is fully implemented and deployed. RESPOND Control Bridge is implemented only partially and it is not deployed yet. For simplicity, the following text is written as if both services were completed, with understanding that in case of RESPOND Control Bridge the text describes the intended final state.

Both services are implemented using Python 3.7 and the asyncio framework and deployed as Docker images on a Docker Swarm cluster. The main reason for these choices is technological compatibility with the rest of Energomonitor's backend infrastructure.

Both services are deployed in only one instance. In case of RESPOND Data Bridge, it is possible to deploy multiple instances for reliability or performance, but this wasn't needed so far. In case

of RESPOND Control Bridge, deploying multiple instances is not possible because they would conflict when subscribing to the RESPOND's MQTT broker (the Mosquitto broker used by RESPOND doesn't implement shared subscriptions, which would solve this problem).

The connection of both services to RESPOND's MQTT broker is secured using TLS.

4.3.4.1 RESPOND DATA BRIDGE IMPLEMENTATION

The RESPOND Data Bridge connects to a Kafka topic produced by DGI Processing and performs a loop that consists of the following steps:

1. Read a batch of data packets.
2. Filter out relevant packets based on a static list of gateway serial numbers.
3. Convert these packets to the RESPOND CDM format.
4. Publish resulting messages to a topic on RESPOND's MQTT broker
5. Perform a Kafka commit to confirm processing of the batch.

RESPOND Data Bridge is designed to be robust. When an MQTT publish fails, it is retried periodically until it succeeds. And when an unexpected error occurs, leading to the service being terminated, the service is restarted and processing continues from the last commited point. This means that occasionally data may be processed twice, but never lost (the at-least-once semantics).

4.3.4.2 RESPOND CONTROL BRIDGE IMPLEMENTATION

The RESPOND Data Bridge connects to RESPOND's MQTT broker, subscribes to a set of request topics using a pattern, and waits for requests. When a request arrives, it performs the following steps:

1. Check whether it should handle the request based on the vendor prefix and a static list of gateway serial numbers.
2. Convert the request into a HTTP request for DGI Commander and issue it.
3. Wait for a response.
4. Convert the response to the RESPOND CDM format
5. Publish resulting message to a topic on RESPOND's MQTT broker.

RESPOND Data Bridge is also designed to be robust. Most of this robustness comes from the robustness of DGI Commander, which tracks the status of all devices, handles the details of the communication, and returns appropriate error codes on errors.

4.4 WATER CONSUMPTION CUSTOM ADAPTER (FEN)

During the early deployment, it was found impossible to install Energomonitor's relay sense water at the Madrid pilot site due to the current installation situation. In order to be able to obtain water consumptions, Fenie Energia worked on an alternative solution.

Fenie contacted water meters company to see if it would be possible to get daily data through its webpage. Fortunately, the meters company publish in their web daily, both hot and cold, water consumption so it was agreed to develop an application to read this information and forward it to the RESPOND platform.

This solution is not perfect as the consumptions sometimes are published with several days of delay and the granularity is daily so it is necessary to profile the measures to hourly values. Nevertheless, it allowed the water consumptions to be been considered in the RESPOND solution as planned.

In Figure 20, we present an example of the excel file downloaded from the water meters company web page with the daily consumption of the households, both hot and cold water.

	Piso	Num Serie	06/03	05/03	04/03	03/03	02/03	01/03	28/02	27/02	26/02	25/02	24/02	23/02	22/02	21/02	20/02	19/02	18/02	17/02
2	1ºA	JU	24448126	174.82	174.75	174.711	174.68	174.625	174.547	174.469	174.445	174.367	174.336	174.312	174.219	174.133	174.055	174.016	173.938	173.914	173.859
3	1ºA	JU	24448074	52.906	52.875	52.836	52.805	52.75	52.688	52.648	52.617	52.586	52.555	52.516	52.469	52.43	52.391	52.359	52.32	52.273	52.234
4	1ºB	ME	24156527	639.719	639.719	639.719	639.719	639.719	639.719	639.719	639.719	639.719	639.68	639.68	639.68	639.68	639.68	639.68	639.68	639.68	639.68
5	1ºB	ME	24156510	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641	485.641
6	1ºC	M	24156529	523.172	523.008	522.828	522.648	522.492	522.258	522.078	521.867	521.664	521.445	521.266	520.977	520.781	520.57	520.203	520.031	519.719	519.539
7	1ºC	M	24448075	330.211	330.086	329.852	329.664	329.469	329.203	329.086	328.883	328.633	328.43	328.266	328.07	327.867	327.57	327.391	327.281	327.086	326.82
8	1ºD	M	24448158	254.102	254.039	253.969	253.883	253.82	253.711	253.625	253.508	253.352	253.211	253.141	253.078	253.008	252.945	252.859	252.781	252.625	252.562
9	1ºD	M	24448145	75.43	75.406	75.32	75.289	75.258	75.219	75.18	75.133	75.117	74.977	74.93	74.898	74.875	74.844	74.789	74.758	74.648	74.594
10	2ºA	JU	24448073	239.383	239.305	239.141	239.141	239.141	239.086	239.07	239.016	238.875	238.836	238.805	238.742	238.625	238.484	238.43	238.391	238.328	238.266
11	2ºA	JU	24450432	175.18	175.102	175.016	175.016	175.016	175.0	174.961	174.922	174.82	174.805	174.742	174.703	174.68	174.562	174.5	174.445	174.375	174.297
12	2ºB	JO	24448127	499.727	499.633	499.633	499.633	499.578	499.523	499.469	499.391	499.258	499.18	499.109	499.047	498.992	498.844	498.773	498.562	498.469	498.289
13	2ºB	JO	24448077	205.43	205.391	205.383	205.383	205.352	205.32	205.273	205.156	205.102	205.031	204.969	204.953	204.906	204.797	204.758	204.703	204.664	204.625
14	3ºA	JO	24156513	572.734	572.508	572.312	572.039	571.727	571.523	571.281	571.078	570.867	570.625	570.383	570.164	569.883	569.664	569.461	569.273	569.07	568.812
15	3ºA	JO	24450431	211.086	211.008	210.883	210.781	210.664	210.539	210.477	210.359	210.297	210.148	209.992	209.922	209.766	209.68	209.547	209.469	209.305	209.156
16	3ºB	RC	24156526	577.766	577.766	577.766	577.766	577.766	577.766	577.766	577.547	577.203	576.938	576.523	576.07	575.812	575.547	575.211	574.898	574.664	574.461
17	3ºB	RC	24156549	165.078	165.078	165.078	165.078	165.078	165.078	165.078	165.0	164.859	164.766	164.617	164.422	164.297	164.164	163.938	163.766	163.547	163.375
18	3ºC	PI	24450410	633.477	632.977	632.555	632.07	631.766	631.422	630.922	630.562	630.156	629.711	629.242	628.844	628.562	628.109	627.602	627.25	626.719	626.328
19	3ºC	PI	24448147	340.555	340.25	339.969	339.703	339.438	339.148	338.859	338.586	338.328	338.086	337.766	337.469	337.258	337.0	336.703	336.438	336.078	335.781
20	4ºA	AL	24448124	105.367	105.0	104.797	104.664	104.648	104.633	104.492	104.367	104.25	104.117	103.891	103.875	103.852	103.602	103.344	103.188	103.094	103.008
21	4ºA	AL	24448076	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055	31.055
22	4ºB	BE	24156546	270.312	270.07	269.859	269.656	269.469	269.297	269.086	268.789	268.594	268.414	268.188	268.039	267.859	267.68	267.477	267.242	267.055	266.789
23	4ºB	BE	24156530	575.055	574.578	574.172	573.672	573.359	573.023	572.57	572.258	571.867	571.562	571.148	571.008	570.805	570.359	569.969	569.57	569.156	568.758
24	5ºA	JU	24156512	200.734	200.711	200.508	200.414	200.336	200.25	200.211	200.211	200.211	200.016	199.938	199.883	199.805	199.781	199.734	199.68	199.336	199.234
25	5ºA	JU	24156509	121.477	121.477	121.398	121.328	121.328	121.266	121.258	121.258	121.258	121.141	121.141	121.094	121.031	120.969	120.906	120.844	120.609	120.539
26	5ºB	PI	24156525	351.961	351.812	351.68	351.461	351.273	351.18	351.047	350.852	350.68	350.516	350.266	350.094	350.031	349.906	349.742	349.594	349.422	349.242
27	5ºB	PI	24156548	149.938	149.836	149.758	149.703	149.617	149.531	149.492	149.414	149.344	149.266	149.211	149.148	149.141	149.039	148.906	148.859	148.758	148.68
28	5ºC	IG	24156531	653.133	652.93	652.547	652.234	652.188	652.188	651.867	651.508	651.273	650.852	650.547	650.211	649.961	649.773	649.484	649.148	648.922	648.523
29	5ºC	IG	24156511	442.891	442.68	442.594	442.289	442.281	442.281	442.133	441.75	441.57	441.273	441.125	440.828	440.508	440.367	440.156	440.008	439.727	439.438
30	6ºA	JO	24448125	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25	354.25

FIGURE 20: WATER CONSUMPTIONS EXCEL FILE EXAMPLE

The purpose of this adapter is to gather periodically daily water consumption measures from the Madrid pilot dwellings (both hot and cold water), profile them hourly and forward them as MQTT message to the MQTT RESPOND server.

The application is designed to be executed once a day within a Fenie Energia server and it has been coded using Microsoft .NET 4.7.1 framework in Visual Studio 2017 IDE. In addition, the following libraries have been required:

- Selenium WebDriver: Set of tools to automatize the use of a web browser (Google chrome in this case), necessary to download and excel file with consumption measures in a daily basis.
- Newtonsoft Json: Allows to serialize the messages in a quick and efficient way to adequate them to the CDM format for the delivery.
- M2MQTT: Necessary to configurate a MQTT client in FEN's server to manage communications in MQTT protocol.

Moreover, to read data from the excel file, it has been required to install Microsoft Access Database Engine 2007 in the server. This version has been chosen due to combability reasons.

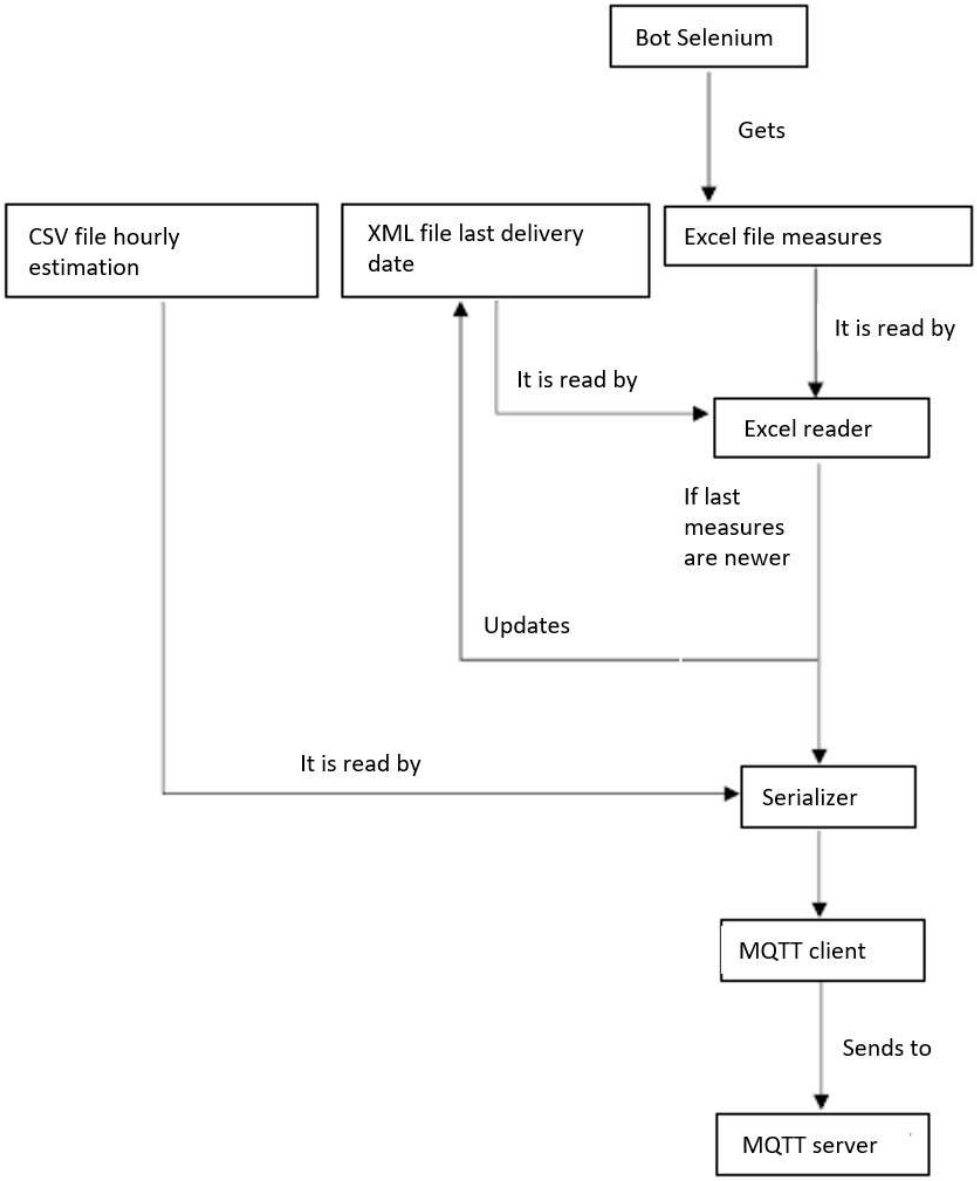

FIGURE 21: WATER CONSUMPTION ADAPTER DIAGRAM

The application encompasses five different parts:

- Automatic Bot to download measures: Water metering company web provides the measures through an excel file that must be downloaded and stored in the server to be parsed later.
- Excel files reader: It is in charge of collecting measures row by row in the excel file and creating one, or several, list of objects (according to the number of messages to be sent to the MQTT broker).

- Serializer: Compounds the messages accordingly the required format. In addition, it applies the hourly profile where the coefficients have been previously calculated and stored in a CSV file.
- MQTT client: Responsible for setting up the connexion with the MQTT server and sending the messages.
- Persistence: Due to the fact that the Water metering company web is not always updated with the last measures on a daily basis, during the parsing of the excel file, the last submitted measured date is checked against an ancillary XML file. In such a way, the application finds out whether it is necessary to deliver just one message with the last measures of the excel file, several messages if there are delayed measures, or no messages at all if the more recent available measures in the excel file have been already sent.

The diagram shown in Figure 21 provides the description of the aforementioned process carried out by the application and how the process components interacts among them:

4.5 EXTERNAL SYSTEM INTERFACES (DEX)

4.5.1 INTRODUCTION

In the present section, an explanation of how DEXMA's DEXCell solution is adapted to be integrated in RESPOND project is summarized. The explanation focusses on DEXCell data acquisition from the pilots, but it must be said that other tasks are also related to external systems that are going to be explained deeply in other deliverables, such as D5.4.
To understand the work carried out by DEXMA, the usual monitoring methodology is explained, so that later it can be compared with the new, which includes the Message Queuing Telemetry Transport (MQTT) ISO.

4.5.2 DEXCELL COMMON SOLUTION

DEXMA DEXCell Energy Manager is a software-based solution specialized in energy monitoring. The common installation for monitoring with DEXCell EM consists of a hardware to convert the information from the monitored system into data. Then, this data is gathered in a gateway manufactured by DEXMA, the DEXGate (DEXCell is a hardware neutral product and other gateways can be connected), so that later it can send the data through GSM or Ethernet connection. The data passes through the insertion API to be stored in the appropriate DB after been managed by DEXMA's data process engine. From here, each software uses the system data or metadata for its processes. From the installation point of view, there exist wide variety of configurations, as shown in Figure 22, where we can observe that DEXCell supports different sensors and devices.

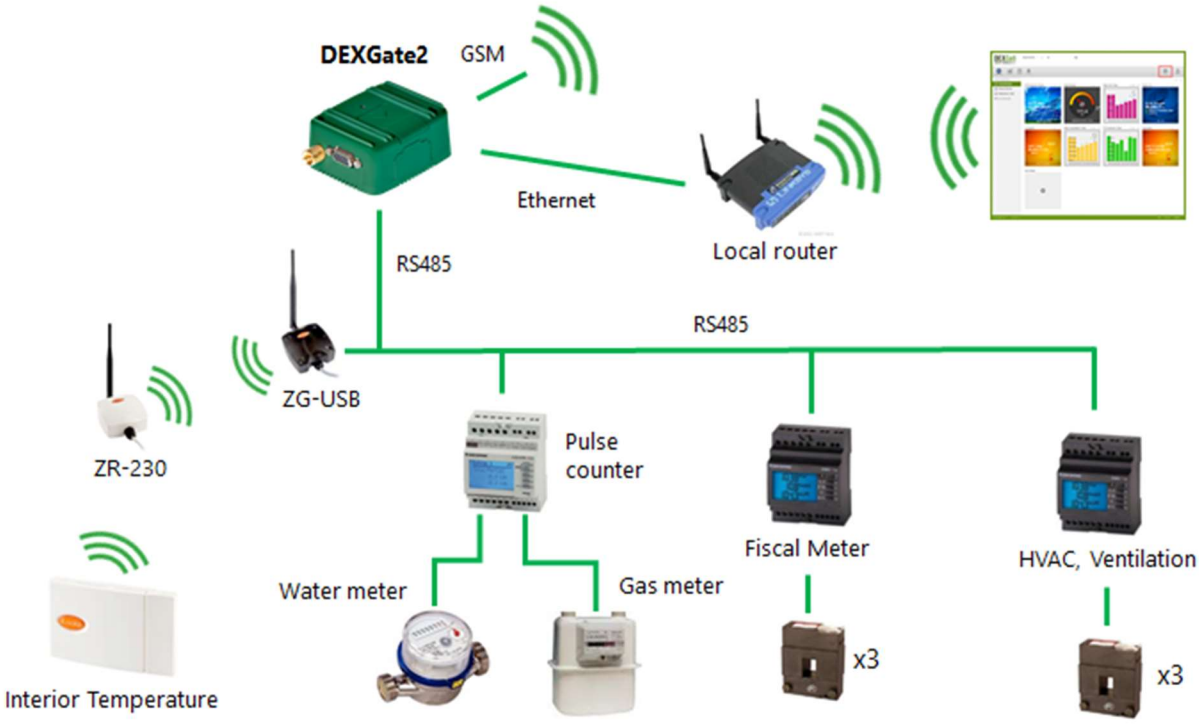

FIGURE 22: MONITORING INSTALLATION EXAMPLE

From DEXCell's point of view, Figure 23 shows the path followed by the data from the system to monitor to each specific service.

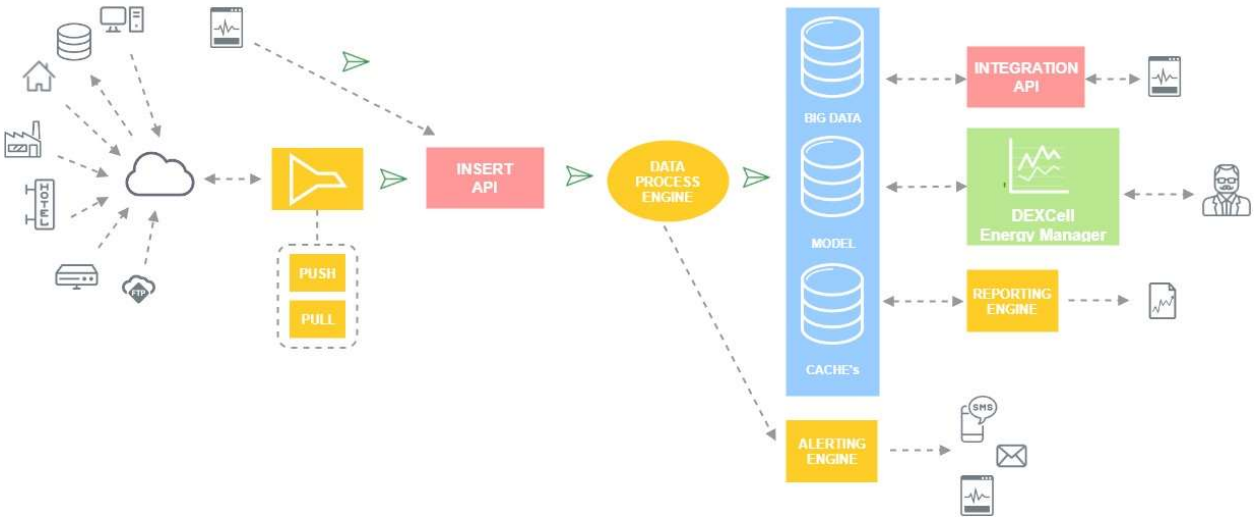

FIGURE 23: COMMON DEXCELL ARCHITECTURE

4.5.3 DESCRIPTION OF THE CHALLENGE TO BE SOLVED

For RESPOND project, the main challenge in terms of system monitoring for DEXMA is to adapt the current architecture and data gathering processes for tertiary sector, where the variety in the

systems to monitor is even bigger, due to the large number of monitoring sublocations. With the current process, explained above, it is possible to observe that the more participants are included in the system to monitor the bigger is the flexibility required for the monitoring system.
In RESPOND project the end users are residential inhabitants, the number of dwellings to monitor is large and the resolution for each one of them is high. Hence, adapting the current DEXCell's data gathering process to make it more flexible was a necessary step to further increase the success chances for a later commercial product. Therefore, in order to make the solution more flexible it was decided to implement MQTT publish-subscribe-based messaging protocol in the DEXCell's data gathering process.

4.5.4 DEXCELL MQTT INTEGRATION

In order to implement the solution, the following steps were followed:

- Needs definition
- Limitations and assumptions
- Design and develop the solution

4.5.4.1 NEEDS DEFINITION

In order to implement a new data acquisition process, the best option was to develop a service capable of communicating with the MQTT broker, using the standard communication protocol defined by PUPIN (the Common Data Model CDM) and DEXCell insertion system. From now on, this service is going to be called DEXCell Bridge (DBridge).
The list below describes the DBridge requirements:

- Read data from the RESPOND MQTT Broker - CDM language
- Manage and relate the income data with locations metadata
- Write data in DEXCell Insert API - DEXCell language

Also, according to the project boundaries there are some limitations that need to be considered, in particular, those related to the performance testing. Due to the stage of the project, load and stress test have not been performed yet.

4.5.5 SOLUTION DESIGN AND DEVELOPMENT

According to the exposed problem above, the solution is divided in two parts. The first one related to the DBridge development and the second one, to DEXCell platform.

4.5.5.1 DBRIDGE DEVELOPMENT

The tasks related to the development of the DBridge are:

- RESPOND MQTT Broker infrastructure integration
- Integrate the CDM in the code
- Develop the translation from CDM to DEXCell

In Figure 24, an architecture of the implemented solution is presented.

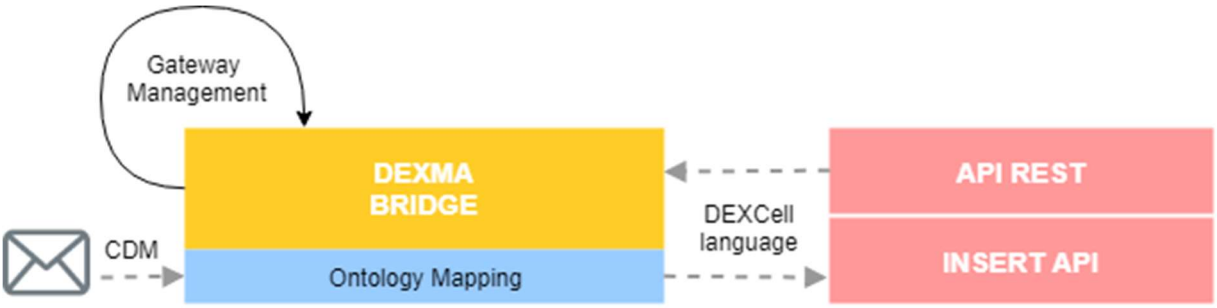

FIGURE 24: HOW DOES THE BRIDGE WORK?

4.5.5.2 DEXCELL PLATFORM CHANGES

In the case of the DEXCell platform, the changes were required for the gateway configuration in both front and back-end. Since for RESPOND project, the platform will be reading the gateway id according to the topology. Figure 25 and Figure 26 show the gateway configuration screens.

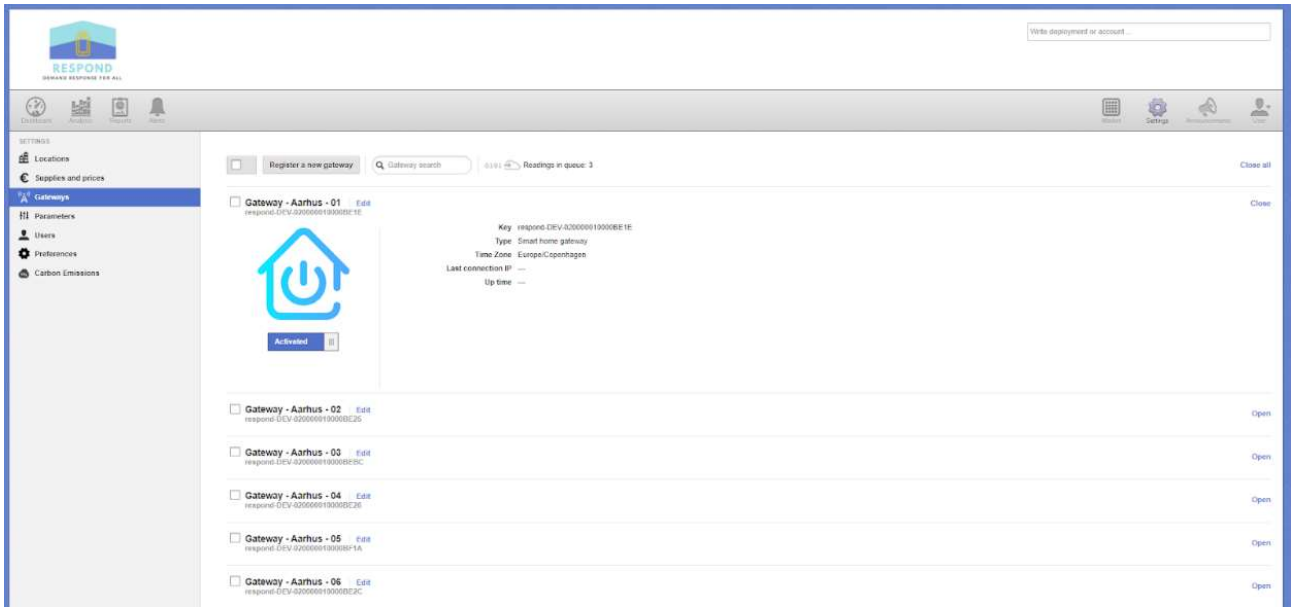

FIGURE 25: GATEWAY CONFIGURATION EXAMPLE

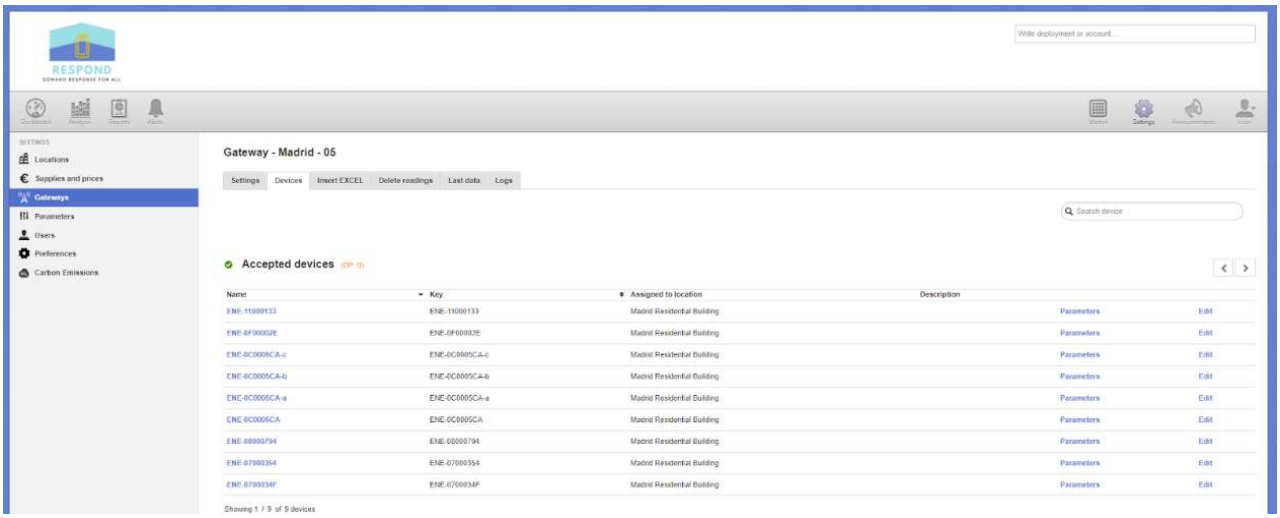

FIGURE 26: EXAMPLE OF DEVICES RELATED TO A GATEWAY

4.5.6 APPLIED SOLUTION

After the development carried out in RESPOND project framework, the new path for data to be stored in DEXCell DB from the dwellings can be observed in Figure 27. Now, the sensors and meters from different providers send data to their gateways (also from those providers). From there each one is processed in a different platform into CDM language, then published in the MQTT Broker. Finally, the DBridge will be subscribed to the topics depending on the topology specification in the DEXCell platform.

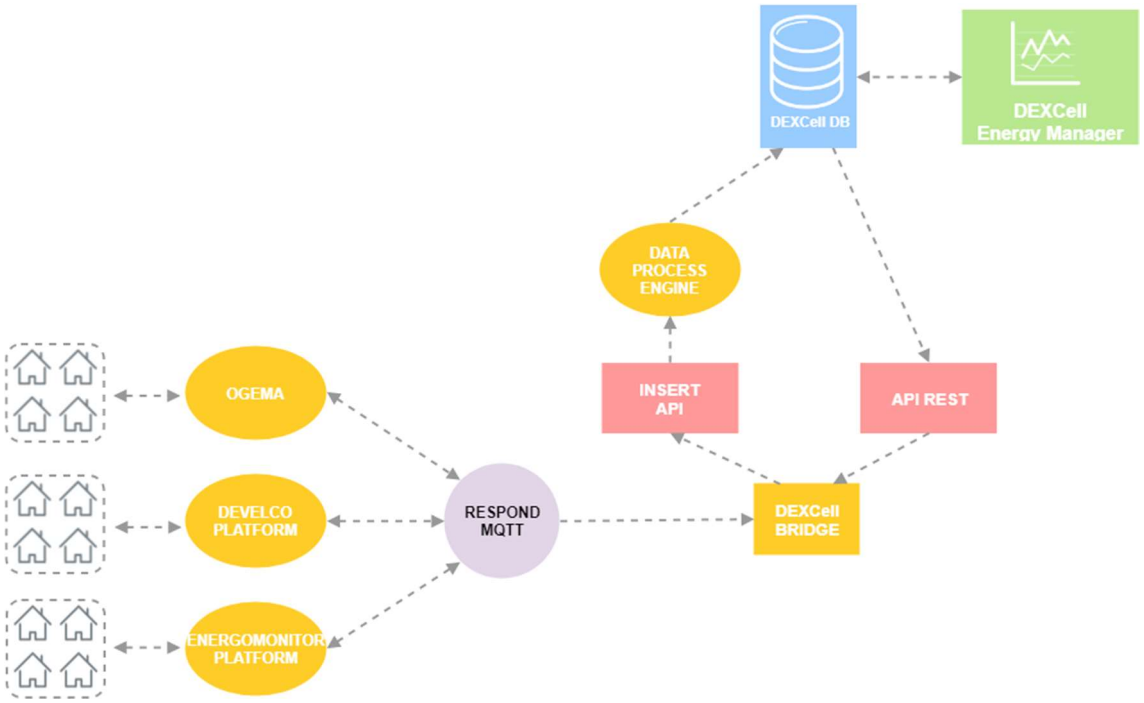

FIGURE 27: RESPOND DEXCELL MONITORING ARCHITECTURE

In Figure 28, a demonstration of data acquisition made through the implemented solution is shown. In the presented case, the data come from a dwelling in Madrid pilot site.

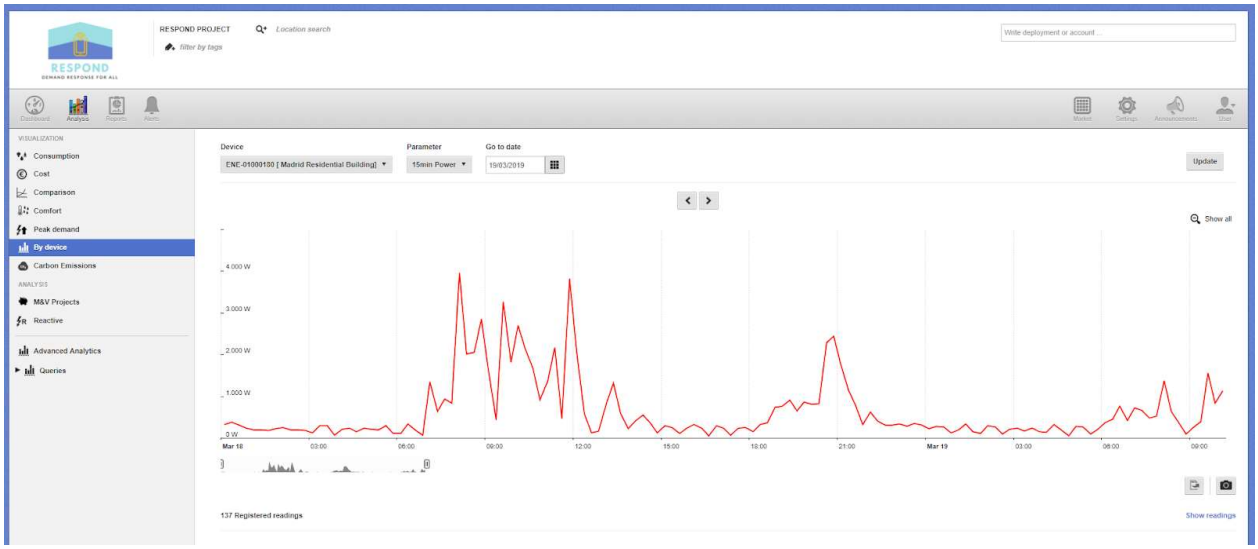

FIGURE 28: DATA AQUISITION THROUGH RESPOND MQTT BROKER

5. CONCLUSIONS

In this report, we have presented the work performed within Task 5.1, where the involved project partners were dealing with interoperability of the RESPOND platform with existing and newly deployed home automation and building management systems, metering equipment, energy resources, etc. All relevant home automation protocols will be supported as part of plug-in library. Besides, the goal is to enable plug-and-play registration of new devices that could be discovered automatically. The aim was to ensure integration of RESPOND with external hardware and software systems, which enables both acquisition of data and control actions to be performed in line with demand response strategy.

Firstly, we briefly revisited RESPOND platform architecture and main interoperability requirements. Next, we focused on RESPOND middleware and Canonical Data Model which is used for two-way data communication between the RESPOND platform and field level devices. Then, we presented the design and deployment of different RESPOND interfaces developed by project partners. In particular, we provided more information regarding OGEMA energy gateway used to integrate the system monitoring thermosolar system in Madrid pilot site. Furthermore, custom developments performed by Develco and Energomonitor regarding integration of their equipment with RESPOND were presented. Then, we provided more details on custom adapter developed by Fenie Energia for water consumption data collection. Finally, the details regarding integration of Dexcell Energy Manager platform with the rest of RESPOND system were presented.

REFERENCES

RESPOND DOCUMENTS

[1] D1.3 RESPOND strategy to support interoperability

[2] D2.2 Integration of key RESPOND technology tiers

[3] D2.1 RESPOND system reference architecture

EXTERNAL DOCUMENTS

[4]Nestle, D., & Strauß, P. (2009). Concept of the Open Gateway Energy Management Alliance.

[5]Feuerhahn, S. (2011). OpenMUC-Monitor & Control. Online. *http://www. openmuc. org.*