PHOENIX

WP4 – Development of an interoperable and secure PHOENIX Smartness Hub based on an ICT solution

Document Version:

D4.2 PHOENIX Smartness Hub implementation – Intermediate Version

2.0

Project Number:	**Project Acronym:**	**Project Title:**
893079	PHOENIX	Adapt-&-**P**lay **H**olistic c**O**st Effective and user-frie**N**dly **I**nnovations with high replicability to upgrade smartness of e**X**isting buildings with legacy equipment

Contractual Delivery Date:	**Actual Delivery Date:**	**Deliverable Type* - Security**:**
28/02/2022	28/02/2022	R - PU

* Type: P - Prototype, R - Report, D - Demonstrator, O - Other

** Security Class: PU- Public, PP - Restricted to other programme participants (including the Commission), RE - Restricted to a group defined by the consortium (including the Commission), CO - Confidential, only for members of the consortium (including the Commission)

Responsible and Editor/Author:	Organization:	Contributing WP:
Alfredo Quesada	OdinS	WP4

Authors (organizations):

OdinS, SAGOE, UMU, UBITECH, S5, MIWENERGIA, VERD

Abstract:

This document details the intermediate version of the Smartness Hub implementation. At this stage, the implementation of the Smartness Hub has been extended to the other pilots and, in an attempt to prepare the Knowledge Layer for providing a more rich representation of the information, the Knowledge Graph has been extended and more work on semantics has been done. A first SRI model has also been included.

User and grid services have also been updated since they were introduced in D4.1, as well as the Security & Privacy framework.

Keywords:

Smartness Hub; Knowledge Graph; SRI; Data analytics services; Security and Privacy.

Revision History

The following table describes the main changes done in the document since created.

Revision	Date	Description	Author (Organization)
0.1	20/01/2022	ToC Draft	Alfredo Quesada (OdinS)
0.2	03/02/2022	Security framework	Alfredo Quesada (OdinS)
0.3	09/02/2022	Data analytics for grid integration services	Valentina Tomat (UMU) Alfonso Ramallo (UMU)
0.4	10/02/2022	Knowledge graph	Josiane Parreira (SAGOE) Stefan Bischof (SAGOE)
0.5	16/02/2022	Data analytics for user-centric services, data analytics for grid integration services, updates on KG section, SAREF4ENER	Dimitra Georgakaki (Ubitech) Kostas Tsatsakis (S5) Josiane Parreira (SAGOE) Alfonso Ramallo (UMU)
1.0	17/02/2022	Updates on data analytics for grid integration services and security framework	Alfonso Ramallo (UMU)
2.0	22/02/2022	Final version including the comments received from the reviewers	Aristotelis Dafalias (VERD) Pablo Barrachina (MIWENERGIA) Alfredo Quesada (OdinS)

Executive Summary

This document is the second part of a set of deliverables that describes the development and implementation of the PHOENIX Smartness Hub.

After the first part of the set, which was focused on a basic representation of the information using Smart Data Models in the Context Broker, this document extends this representation taking the Knowledge Graph to a next level by adding a triplestore. Apache Jena Fuseki has been chosen for this purpose and will provide the semantics required by services from upper layers in what was introduced in D4.1 as the Knowledge Layer. A first version of the SRI model is also introduced using the RDF Data Cube Vocabulary.

From the data analytics services side, basic but operational versions of user-centric and grid-related services are included in this document with both using real data.

And in the security section, the list of risks identified in D4.1 has been analysed and there has been included a description of how the Security & Privacy framework deals with each one of them. In addition a fully functional version of the security components is described as well as some details of the deployment based on the requirements identified in the different pilot sites.

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 893079, but this document only reflects the consortium's view. The European Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

Table of Figures

Table of Tables

Acronyms

Abbreviation	Description
API	Application Programming Interface
BMS	Building Management System
CB	Context Broker
CP-ABE	Cyphertext-Policy Attribute-Based Encryption
DCapBAC	Distributed Capability-Based Access Control
DLC	Direct Load Control
DRE	Demand Response Event
EV	Electric Vehicle
HMM	Hidden Markov Model
HTTP	Hypertext Transfer Protocol
IDM	Identity Management/Manager
IoT	Internet of Things
ISP	Imbalance Settlement Period
JSON	JavaScript Object Notation
KG	Knowledge Graph
MQTT	Message Queuing Telemetry Transport
NGSI-LD	Next Generation Service Interfaces – Linked Data
NSGA	Nondominated Sorting Genetic Algorithm
OWL	Web Ontology Language
PAP	Policy Administration Point
PEP	Policy Enforcement Point
PoC	Proof of Concept
RDF	Resource Description Framework
REE	Red Eléctrica Española
RFC	Random Forest Classifier
REST	Representational Estate Transfer
SAREF	Smart Applications REFerence
SAREF4ENER	SAREF for Energy
S&P	Security and Privacy
SCADA	Supervisory Control and Data Acquisition
SOSA	Sensor, Observation, Sample and Actuator (Ontology)
SPARQL	Simple Protocol And RDF Query Language
STD	Standard Deviation
TLS	Transport Layer Security
SRI	Smart Readiness Indicator
USEF	Universal Smart Energy Framework
UTC	Universal Time Coordinated
VRF	Variable Refrigerant Flow
W3C	World Wide Web Consortium
WebUI	Web User Interface
XACML	eXtensible Access Control Markup Language

1 Introduction

1.1 Scope of the document

This document is the second part of a set of deliverables that describes the development and implementation of the PHOENIX Smartness Hub. This intermediate version includes updates in the design of the Knowledge Graph that is intended to be used for offering a high-level access to the data stored in the PHOENIX Smartness Hub.

Updated versions of data analytics services are also included in this document in their path to provide the information required by services from WP5 and WP6 now that devices are already operational and real data is available in the PHOENIX Smartness Hub.

Finally, the Security & Privacy framework has received an upgrade mostly in the security part after having completed and deployed a subset of the security components. The privacy-related security component (CP-ABE) will be included in D4.3.

1.2 Relevance to other deliverables

This deliverable is related to the rest of the set, namely Deliverables 4.1 and 4.3.

This Deliverable is also connected to D3.2 "*Technical upgrades and integration mechanism for legacy equipment – Intermediate version*" as the security features described in the current one have been introduced in D3.2, an integration-focused Deliverable. In addition, some dependencies arise regarding the provisioning of entities in the context of actuation based on the chosen data models.

This deliverable is also relevant to D7.2 "*Initial Pilots Deployment, Operation and Validation*". After having finished the PoC, there is a direct relation between the deployment of the pilots and the security requirements that must be fulfilled by the IoT gateways, BMS, etc. using the S&P framework.

1.3 Structure of the document

This document is divided in four main sections including this introduction. Section 2 includes a more refined version of the basic models defined in D4.1, including SRI, which provide some higher-level functionalities. Section 3 includes the updated revision of the data analytic services, separated in two categories, user-centric and grid related. Section 4 includes the updated version of the S&P framework as well as information about specific details of the current deployment and integrations from a security point of view.

2 Integrated Data Models, Knowledge Graphs and Automatic Semantics

2.1 Knowledge Graph

Integrated Data Models and Knowledge Graphs (KGs) provide means to represent and access heterogeneous data from multiple sources in a unified manner.

As already mentioned in D4.1, unified representation/access does not imply a centralized approach. PHOENIX approach is a combination of edge and cloud storages, thus offering a scalable solution. Semantic data representation ensures that data stored at the different locations can be easily combined and analysed. This section details PHOENIX's decentralized KG solution and describes the current status of its use for enabling the different analytics services.

In our project, the edge storages are enabled by a FIWARE Context Broker, while a triplestore is used as cloud storage solution.

Figure 1 illustrates the current PHOENIX architecture at the PoC site that is also being used for the other pilot sites.

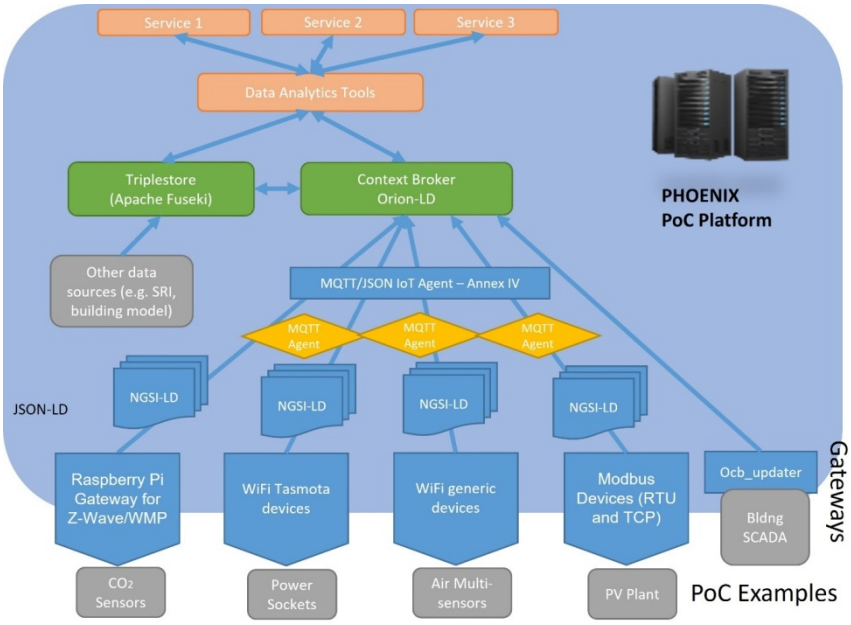

Figure 1 – Current Architecture (extended from the PoC-only version)

The difference with the previous version (D4.1) is the addition of a triplestore. In the first phase of the project, we focused on getting the different sensors and their data modelled and published at the Context Broker. For sensors, we made use of Smart Data Models representations [1], which are in compliance with FIWARE NGSI version 2 and NGSI-LD specifications, and therefore suitable for describing data within FIWARE Context Brokers. For representing the stream of data coming from the sensors, we use concepts from the IoT-Stream ontology [2] developed within the IoTCrawler project [3] which in turn extends the W3C's SOSA ontology [4] to describe sensor observations.

In the current phase of the project, we have extended the data at the Context Broker to include actuation entities (details below). Moreover, we have added a triplestore for higher level data representation coming from sensors (via the Context Broker) as well as from other data sources. A triplestore, or RDF store, is a purpose-built database for the storage and retrieval of triples through semantic queries. All data represented following the models described in D4.1 can be directly stored in such triplestores. This combination of semantic edge and cloud storage constitutes the edge-cloud Knowledge Graph solution, and is the backbone for the development of multiple analytical engines and services, which are geared towards building occupants and the grid. Moreover, data generated by the services is also captured and used to enhance the information at the Knowledge Graph.

For the triplestore we have opted for the Apache Jena Fuseki [5] implementation. Apache Fuseki is an active, free and open-source RDF triplestore implementation that is part of the Apache Jena RDF framework. Apache Fuseki is based on the Apache Jena RDF API and implements a SPARQL endpoint. It provides an implementation of the SPARQL 1.1 specifications: the SPARQL query language and the SPARQL Update language as well as the SPARQL graph store protocol. Via the Jena RDF API also OWL reasoning capabilities are available. Fuseki can run on several different backends and provides a graphical WebUI for administration access and a query editor.

To support the services being developed in PHOENIX, the Knowledge Graph is being populated with information coming from relevant sources.

At the edge level the data in the Context Broker has been extended to also include the actuation entities, as shown in Figure 2.

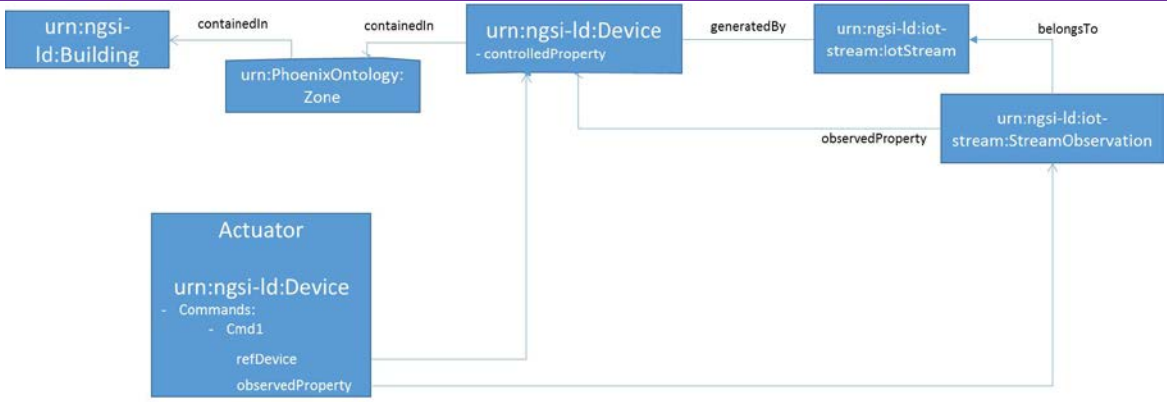

Figure 2 – Actuation model

The actuation data is going to be used mostly by services that need to act on devices with actuation capabilities, such as thermostats, as part of their normal operation. This is a visual representation of the extended hierarchy of entities created after D4.1 and D3.1, and a more detailed explanation of the format of actuation entities is available in the *Formats of new types of entities* section of D3.2.

2.2 SRI model

At the cloud level, the current focus is on the SRI service. The SRI service aims to provide recommendations for buildings' upgrade based on the current SRI assessment and additional information about the functionality levels of the different SRI services. For this service, we have created a semantic model for the relevant SRI concepts, such as domain, impact, services and functionality levels. The first version of the SRI model was described in D4.1. Since then, the model has been modified following the updates on the SRI calculation sheet.

Figure 3 shows a conceptual overview of the model.

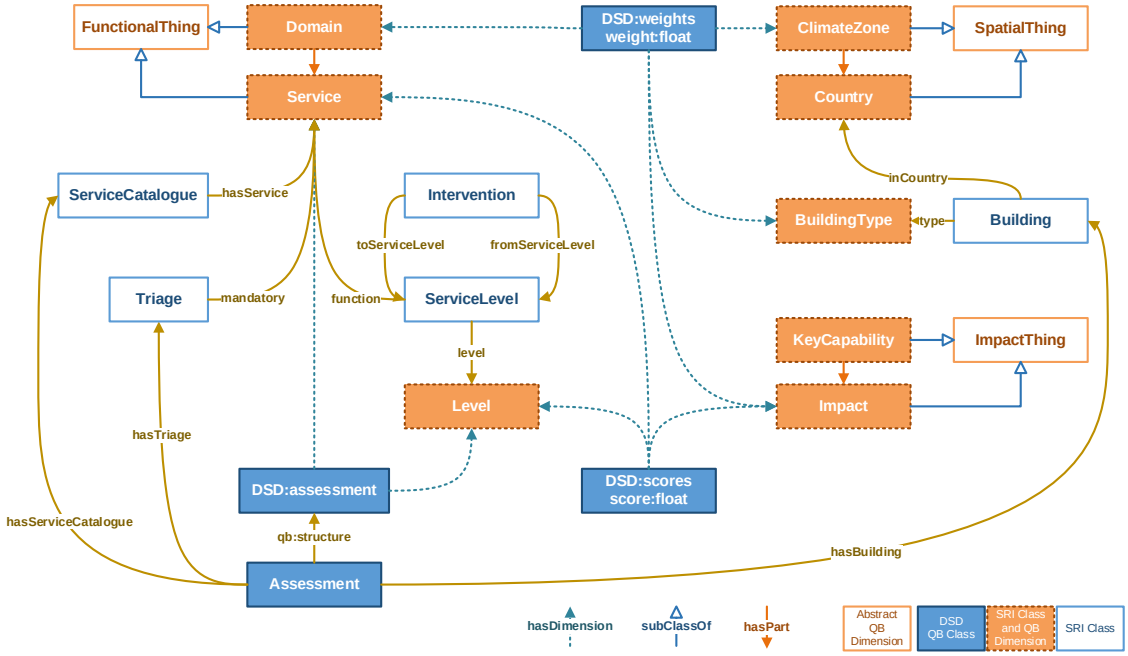

Figure 3 – Conceptual overview of the SRI model

The model is based on the dimensional modelling approach which is the standard approach for data modelling of (also relational) data warehouses. Specifically, we use the RDF Data Cube Vocabulary [6] to represent the data cube specifications (DataStructureDefinition (DSD)) as well as the SRI matrices.

Figure 3 shows the five possible dimensions: Domain/Service, ClimateZone/Country, BuildingType, KeyCapability/Impact and Level. The Assessment (qb:DataSet) is a logical anchor for assessment metadata. The model supports custom service catalogues and definition of the SRI triage process results. In addition the model has been extended to include information about building's upgrades that can lead to higher SRI scores (Interventions). The model currently allows to indicate, for consecutive pairs of services' functionality levels (e.g., between "H-1a Level 1" and "H-1a Level 2"), whether a software of a hardware intervention is required to move from the lowest to the highest level. For example, the SRI service H-2a "Heat generator control (all except heatpumps)" has the Level 0 functionality "Constant temperature control". Via a software intervention using external weather information, a building could be upgraded to Level 1 "Variable temperature control depending on outdoor temperature".

This can be represented in our model as:

```
[] a srii:Intervention ;
   srii:fromServiceLevel sri:h2alevel0 ;
   srii:toServiceLevel sri:h2alevel1 ;
   srii:hasInterventionType srii:software .

sri:h2alevel0 a sri:ServiceLevel ;
    rdfs:comment "Constant temperature control" ;
    sri:function sri:h2a ;
    sri:level sri:level0 .

sri:h2alevel1 a sri:ServiceLevel ;
    rdfs:comment "Variable temperature control depending on outdoor temperature" ;
    sri:function sri:h2a ;
    sri:level sri:level1 .
```

To populate the KG with SRI information we automatically map the SRI assessments to our SRI model. The SRI data is then stored in the triplestore and linked to other available building information (the building model if available). Our model is intended to be integrated with other existing building models such as SAREF's extension for building domain [8] and BRICK [9]. To that end we have defined a "Building" class which has equivalent classes in both aforementioned models.

2.3 Energy flexibility data uses a subset of SAREF

The extension of SAREF specification, called SAREF4ENER, has been used in PHOENIX to model the data related to demand response events in the context of flexibility management. SAREF4ENER aims to enable interoperability between solutions from different owners in the smart home domain with an extensive ontology focused on the demand response event scenario.

In particular, the flexibility services offered by PHOENIX are aligned with the scenario envisaged by SAREF4ENER in which devices react to a given flexibility requests with the aim of reducing load consumption by modifying their usage pattern.

Once the flexibility is agreed between the aggregator and PHOENIX, the demand response events to be performed on the devices involved have to be defined.

SAREF4ENER proposes a way to model these demand response events, as Direct Load Control (DLC).

Specifically, SAREF4ENER defines a series of classes that allow this scenario to be modelled:

- LoadControlEventData: used to represent information about the event on a device, indicating the period of actuation and the specific device controlled.
- LoadControlEventAction: represents the type of control that is performed on the device, indicating whether it is a pause, reduction or increase in consumption. The list of control types is as follows: emergency, increase, normal, pause, reduce, resume. Is associated to LoadControlEventData.
- LoadControlStateData: models the information regarding the status of the demand response event. It is associated to LoadControlEventAction.
- LoadControlEventState: is a class associated to LoadControlStateData that represents the possible states of the demand response event. The possible status values are: eventAccepted, eventStarted, eventStopped, eventRejected, eventCancelled, or eventError.

In addition, some classes related to the next step, the evaluation of flexibility results, are being modelled for integration into PHOENIX with a dedicated entity that will be created ad hoc for this project.

2.4 Automatic Semantics

One of the goals of the project is to help seamless (Adapt & Play) integration of devices. This is already in practice in the project, by means of reusing data models and exploring context brokers for registering devices and their data. In the next phase of the work, we will explore the data collected by the framework to further support devices' registration.

To that end we are creating "data profiles" that characterise different types of devices (e.g., indoor temperature, humidity, energy consumption, etc.). The data profiles are obtained following a time-series analysis of the data provided by the devices, with the purpose of providing compact, yet meaningful information about the devices. Data profiles will be added to the PHOENIX' KG, thus enhancing the knowledge base on buildings. A direct use of the extended KG will be to support automatic semantic labelling of devices. For the automatic semantic labelling, we will explore similarities among the data profiles to suggest semantic annotations when registering a device to the platform, based on the data coming from that device. By suggestion semantic labels, the PHOENIX platform not only ease the registration process for the users, but it also fosters model reuse, which consequently decreases the data integration efforts. The outcome of this work will be described in the final deliverable for WP4.

3 Data analytics services

3.1 Data analytics for user-centric services

3.1.1 Occupancy nowcasting and forecasting for building occupants

The nowcasting occupancy prediction, as well as its short-term prediction (between 15 minutes and 3 hours ahead), are of crucial importance for the best interaction of the building occupant with his surroundings and the preservation of his comfort. More specifically when the PHOENIX system needs to make decisions on how to act on some devices e.g., lights, it needs to be aware of the user presence because the automatic turning off of lights could disrupt the user if of course the room is occupied.

For this purpose, we have developed a 3-step algorithm that is described as follows:

Initially, we have the unsupervised clustering phase triggered by a scheduled cron job once per week. In this phase, a 2-week historic dataset regarding temperature, relative humidity, CO2 and luminance measurements is considered, per zone (only the residential zones are taken into consideration, e.g., parking is excluded). These historic data are stored in our local Elastic Search instance and are enriched each day from the historic sync cron job between the Historical Data component and Elastic that runs every night at 03:00 UTC.

Before transforming the data into a useful Pandas dataframe for our clustering algorithm, some pre-processing steps are required such as reindexing, matching the timestamps, handling missing values, accounting for holidays, splitting weekdays and weekends, standardization and dimensionality reduction.

Then Kmeans clustering is then invoked and tries to give the labels occupied and non-occupied to the input dataset. So, the resulting dataset is now labelled, and the problem has been transformed to a supervised one. Note here that the dbscan method has also been tested but with not as good results as the ones of the Kmeans.

At the second phase, the supervised one, the Random Forest Classifier (RFC) is invoked. The input dataset (from the previous clustering phase) is split into a training (70% of the total) and a test set (30% of the total). The classifier is trained on the 70% and tested on the remaining 30% where the corresponding accuracy and recall are calculated.

This resulting occupancy model is stored in Elastic Search in the following document form:

```
occup_model = {
    "model": gANjc2tsZWFybi5lbnNlbWJsZS5fZm9yZXN0ClJhbmRvbUZvcmVzdENsYXNzaWZpZXIKcQ……,
    "trained_at": date,
    "features_used": light, temperature, relativeHumidity, co2
    "accuracy_score": 98.68,
    "recall_score": 97.17,
    "missing_timeseries_perc": 8.85
  }
```

The field “model” represents a base64 binary form of the trained model and the field “missing_timeseries_perc” is the % percentage of missing values in the dataset. The accuracy range is between [98.68 - 99.5] and the recall range lies between [97.17 – 100].

At the third phase, the predict phase, the RFC is called again to predict the occupancy state per residential zone only for the latest time frame. The latest time frame is considered as valid if the given timestamp of a measurement falls within an accepted time range of the last 30 minutes. This compromise is because there is usually a delay in the retrieval of measurements from the Orion Broker and because not all sensors are synchronized to produce measurements at the same timestamp. If this time frame is considered valid, then a new dataframe is constructed with the latest values of the features used (`light, temperature, relativeHumidity, co2`).

The new dataframe is fed to the RFC, the predicted label Occupied or Non-occupied is returned as a result from the RFC and the corresponding value is also shown in the Building Occupant Dashboard along with the zone id.

For the short-term (15 minutes – 3 hours ahead) occupancy prediction, another approach has been selected. Two prediction requests to the Algorithms Engine are made, regarding the zone energy consumption and the zone CO_2 level and the execution results are made available in the Orion Broker as two different entities. We need both entities because there are some zones where Energy Power Meters are available, but CO_2 meters are not, and vice versa, so the algorithm chooses according to the meter availability.

The occupancy labels that derived from the previously described clustering phase can be used as historic labels of a training dataset that will be used for training an HMM model. In HMM, a sequence is modelled as an output of a discrete stochastic process [7], which progresses through a series of states that are ‘hidden’ from the observer and is widely used in problems of estimating occupancy.

In more detail, for each occupancy state, the state of distributions is calculated. Then the transition matrix for the Hidden Markov Model is generated according to the following form:

prob(STATE_0 -> STATE_0), probability for the zone to remain empty
prob(STATE_0 -> STATE_1)], probability for the zone to get occupied
prob(STATE_1 -> STATE_0), probability for the zone to get empty
prob(STATE_1 -> STATE_1), probability for the zone to remain occupied

In the next step, the occupancy predictor algorithm calculates the starting state probabilities for the occupied and not occupied state. The HMM model is then trained and fitted using Baum-Welch algorithm taking into account all previous information. The test set (predicted consumption or CO2 level) is used to derive the prediction of next occupancy states using the Viterbi algorithm.

3.1.2 Default Comfort calculation for building occupants

Another crucial aspect of the automated control actions that will improve the thermal, visual and air quality comfort of the building occupant is the knowledge of the users' default comfort profiles as well as their relevant short-term predictions. Here we will explain the default user comfort profile and in D4.3 we will account for the short-term predictions.

Initially the algorithm needs some configurable metrics as input such as the thermal vector range, the visual vector range, the air vector range and the humidity vector range as well as their respective vector steps (these are taken from literature). This can easily be explained with the example of a thermal vector of $T_{min} = 20$ and $T_{max} = 30$ with a time step dt = 1.

Then the algorithm considering a specified building zone, links the respective vector to the associated sensors of the zone and collects their values. If the sensor values cannot be retrieved or if the timestamp validation fails (like we previously explained for the occupancy algorithm), a corresponding error message is returned in the internal application and the frontend part demonstrates a N/A to the user at the specified metric.

If there are sensor data available at the time of the request, the calculation of a user's default comfort based on their survey responses is triggered. We remind here that upon a user's first login, he is being redirected to fill in a survey.

A part of it corresponds to preferences regarding internal temperature, lighting and humidity, and is shown in the next example:

```
"I feel comfortable in very warm places",
   "I feel comfortable in warm places",
   "I feel comfortable in a bit colder places",
"I feel comfortable in very bright places",
   "I feel comfortable in bright places",
   "I feel comfortable in places with normal brightness",
   "I feel comfortable in darker places",
"I feel comfortable when the air in my home is dry",
   "I feel comfortable when the air in my home is moist ",
   "I feel comfortable when the air in my home is neither too dry nor too moist"
```

These answers are being translated into vector values when appropriate mappings are made in the algorithm taking into account the vector's min and max limits and the respective time step. So, as a result in each vector a sub-area that denotes comfort is determined.

Then a comparison is made with the real-time sensor values and if the value lies in the previously specified comfort area, then for this calculation time step the user is being identified as comfortable, else uncomfortable. For the non-residential zones, comfort is not calculated.

3.1.3 Descriptive analytics for the delivery of basic metrics

Following the requirements of D2.1 the following metrics have been calculated and made available in the building occupant dashboard:

Table 1 – Calculations per KPI

Metric number	Metric name	Calculation
M_1	Real Time Average Room Temperature (Celsius)	Average sensor measurements (if multiple sensors are present) per building zone for each time step
M_2	Real Time Average Relative Humidity (%)	Average sensor measurements (if multiple sensors are present) per building zone for each time step
M_3	Real Time Average CO_2 level (ppm)	Average sensor measurements (if multiple sensors are present) per building zone for each time step
M_4	Real Time Average	Average sensor measurements (if multiple sensors

	Luminance level (lux)	are present) per building zone for each time step
M_5	Average Room Temperature (Celsius)	Average value for the last 15 days per building zone
M_6	Average CO_2 level (ppm)	Average value for the last 15 days per building zone
M_7	Average Luminance level (lux)	Average value for the last 15 days per building zone
M_8	Real Time Thermal Comfort Status	Identification of comfort status or not per building zone for each time step
M_9	Real Time Air Quality Comfort Status	Identification of comfort status or not per building zone for each time step
M_10	Real Time Visual Comfort Status	Identification of comfort status or not per building zone for each time step
M_11	Real Time Occupancy Status	Identification of occupied status or not per building zone for each time step

In the final version of PHOENIX Smartness Hub implementation (D4.3) the full list of descriptive KPIs will be made available in the dashboard both for the building occupants as well as the building managers. Another improvement that can be made for the near real time measurements is the possible subscription of the dashboard to the Context Broker so as to minimize the amount of GET requests per user click in every sensor update.

3.2 Data analytics for grid integration services

Grid-oriented services are a fundamental part of PHOENIX, thus requiring intelligent interaction with the different grid agents (e.g. aggregator). To perform these services, the development of algorithms and data analysis is necessary. This section details the implemented algorithms and their connection to the PHOENIX architecture in terms of interaction with the grid.

3.2.1 Context of algorithms for grid integration

The interaction between PHOENIX and the grid is a fundamental part of the project. For this, it is essential to use and develop algorithms and analytics techniques that allow PHOENIX to behave intelligently with the grid. A key issue in grid-oriented services is Demand Response Events.

Demand response events are performed by the Flexibility Engine. First, Flexibility Engine is in charge of the negotiation between PHOENIX and the aggregator, a step in which the feasibility of the required flexibility is evaluated by examining the energy availability of each of the devices that may be involved in the flexibility operation. Aggregator requests PHOENIX about a decrease of the load on its connection points for a specific period and PHOENIX offers different flexibility offers after the evaluation of the available energy on its appliances including the demand response events to be performed. Finally, the aggregator accepts one of the offers and instructs the Flexibility Engine to initiate the negotiated demand response events for the specific period, to achieve the agreed flexibility.

Data analytics techniques play an important role in the first phase of flexibility negotiation, which is responsible for assessing the feasibility of the required flexibility. For this purpose, the Flexibility Engine establishes a communication with the Algorithms Engine, which will require the resolution of different analytics techniques to make an estimation of the available flexibility that can be offered to the aggregator.

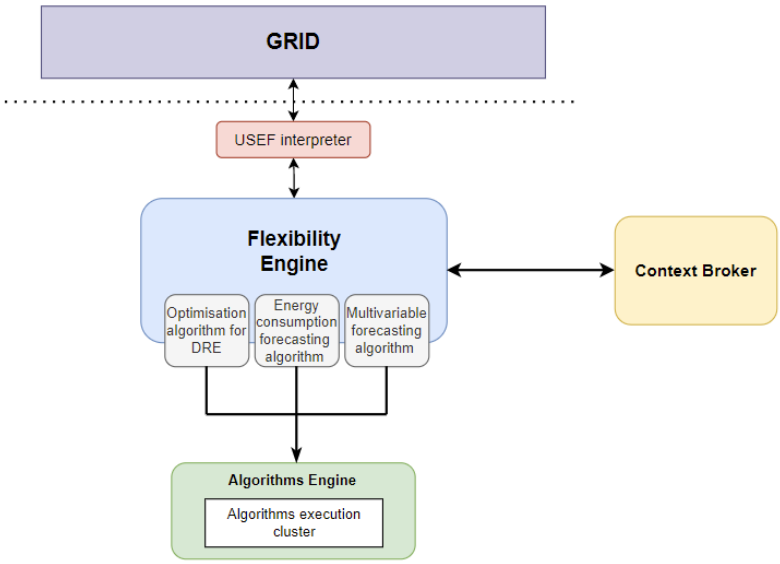

Figure 4 – Architecture and components for flexibility

Figure 4 shows the architecture designed to implement the flexibility services by the Flexibility Engine. This includes the connection with the rest of the PHOENIX components such as Context Broker, Algorithms Engine or USEF interpreter that are essential to perform the demand response events. The creation of the Algorithms Engine allows decoupling the execution of data analysis algorithms from the consumer, offering a more scalable architecture since many services require the execution of these algorithms, which are indispensable for their tasks. In this case, the consumer is the Flexibility Engine.

Figure 5 shows the communication flow to perform an execution of an algorithm in the Algorithm Engine. This is done by creating dedicated entities which detail the request for the algorithm to be executed together with the data streams and some required parameters.

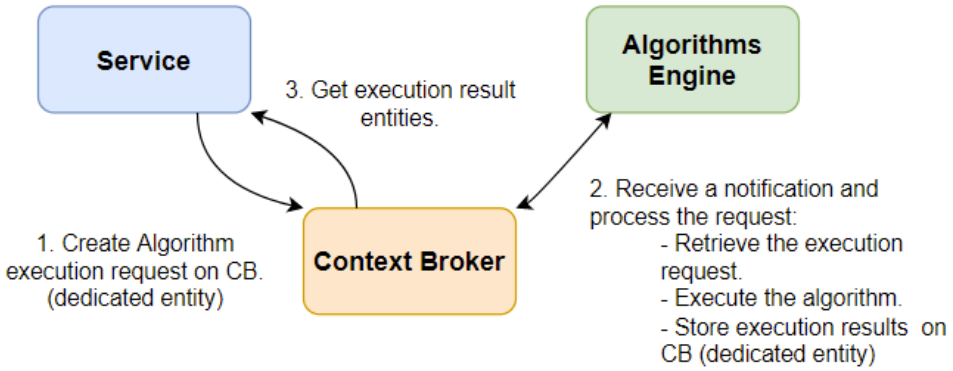

Figure 5 – Communication flow with Algorithms Engine

The algorithm engine, previously subscribed to these request entities, receives a notification of the new algorithm request to be executed. It then performs the execution of the required algorithm and stores the results in the Context Broker. Finally, the consumer/service (e.g., Flexibility Engine) can consult the results in the Context Broker to continue with its tasks.

3.2.2 Algorithms for forecasting, now-casting and benchmarking

For the design and evaluation of flexibility it is necessary to use data analysis to estimate the amount of energy that will be consumed by the devices and the load profile based on the timestamps during the period in which flexibility is requested. These algorithms could be configured for different time scales.

For this purpose, a time series forecasting algorithm has been developed, whose execution is requested by the Flexibility Engine to estimate the load profile of each of the devices that may be involved in demand response events. This process consists of analysing time series energy consumption data from previous periods and predicting the load profile for the flexibility period. This analysis allows to know both the estimated energy load and the operating period of the device.

Finally, a multivariate generic forecasting algorithm has been developed for forecasting considering external data from official sources such as REE, the API of the Spanish electricity operator. Some parts of algorithm development code are contained in Annex I, including also the code related to the extraction of stream data from the Context Broker and, after the execution of the algorithm, the storage of the results in the dedicated entity in the Context Broker.

This algorithm supports the inclusion of additional data such as electricity price or current energy generation, thus being able to offer smarter flexibility offers by increasing their feasibility and adding monetary information. For example, Figure 6 shows the forecast of hydro generation for the first 4 months of the year 2021, after analysing the data of the previous full year.

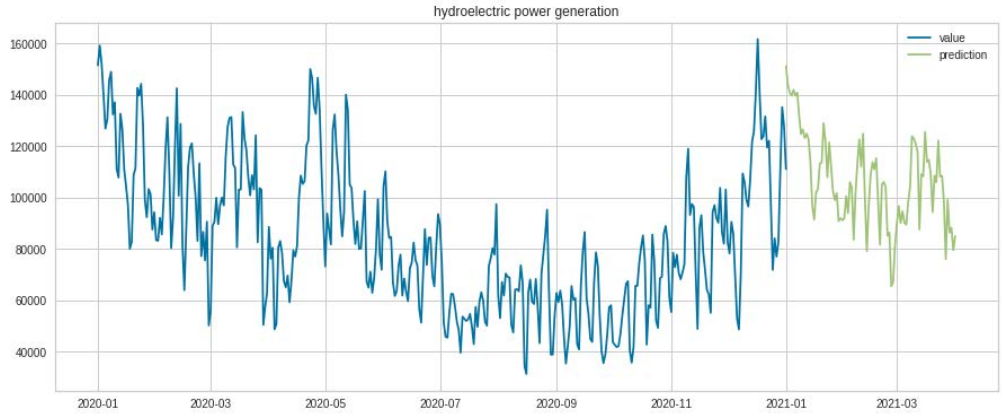

Figure 6 – Hydropower generation forecast

3.2.3 Algorithms set for optimal operations for grid integration services

One of the necessary functionalities of PHOENIX, specifically on the actions towards grid integration, is the capability of modifying the demand. On our preliminary approach this took the form of re-scheduling devices on an optimal way. Hence, one of the algorithms developed was an optimisation algorithm that reschedules the operation of devices in intervals of 15 minutes representing the ISPs.

The so-called shiftable devices allow planning a re-schedule that can lead to a consistent reduction of the power peaks, creating a more distributed curve of power usage. The re-scheduling could be hard to accept from an end-user's point of view since the occupants' habits need to be changed. In order to increase the occupants' acceptance of this change, one of the variables of the optimisation problem will be maintaining this shifting in time as little as possible.

Table 2 – Types of devices based on their shifting capabilities

	Shiftable devices	Controllable devices	Non-shiftable
Description	Their entire operation can be moved to another time	Their operation can be modified so they use less power without moving it to another time	Their operation neither can be moved to another time nor it can be reduced
Example	EV charging	HVAC (thermostat)	Computer /TV

Considering the preferences of a given PHOENIX user, we can consider that we are facing a multi-objective optimisation problem, defined by three objective functions: (1) minimisation of the maximum power peak (desired by the utility), (2) minimisation of the overall cost per user (desired by the user), and (3) minimisation of the distortion to the user (desired for the user). Maximum peak power, cost and distortion are calculated through the solutions vectors as follows:

1. Peak(T', t) = max(Power(T',t)) / n
2. Cost(T',t) = sum(Power(T',t) . C(t))
3. Distortion(T') = sum(|T – T'|) / n

Where n is the number of consumers, C is the price of power over a day, T and T' are the power-on timestamps, respectively referring to before the optimisation and after it. Hence, T-T' is the variation of the starting time of use of the appliances.

To solve this kind of optimisation problem in machine learning, a widely used solution is recurring to a search-based algorithm called genetic algorithm. In this way, it is possible to obtain a set of points that solve each objective without being conflictive for the others. These points will form the so-called *Pareto front*, which represents the solution to the multi-objective optimisation problem. Those solutions can be later sent to the Flexibility Engine, so the right actuation orders can be sent to the devices.

To help visualising how the method described can be applied, an example of how to set the optimisation problem is presented in this subsection. First of all, it is necessary to define the agents of the problem. In this case, realistic user profiles can be used, in order to obtain the appliance data to construct the appliance timestamp vector (consumption sources). Then, the decision space has to be set, that is the range of choice. In this case, the decision space is defined by the number of homes, multiplied by the variation in time of the appliances power-on. This displacement can fluctuate in an interval of +/- 24 hours, i.e. the action can be moved to any hour of the day. The algorithm used for the multi-objective optimisation is defined by setting its parameters. In this case, a possible solution is given by the NSGA II algorithm that uses the mechanics of natural selection: similarly, to the theory of biological evolution, individuals face a 'survival selection', based on crowding distance. Hence, the algorithm will test and evaluate modification to the timestamp T (the one chosen by the user before the optimisation) in search for the minimum values for the three functions described in the previous subsection. The results are improved by iterating the process until, at the end of the execution, the Pareto front of the optimised solutions is obtained as a set of 3-dimensional vectors.

Parameters can be assigned as follows:

Table 3 – Description of the algorithm used for the multi-objective optimisation

Library of the algorithm	pymoo.algorithms.moo.nsga2
Crossover mechanism	integer simulated binary crossover
Mutation Mechanism	integer probability mutation
Crossover parameter	0.9
Mutation parameter	0.01
Population size	300
Termination criteria	tolerance
Termination criteria parameter	0.05

The proposed multi-objective optimisation algorithm for shiftable loads is used by the Flexibility Engine. Some parts of optimisation algorithm development code are contained in Appendix A. When the Flexibility Engine receives a flexibility request from the grid, which indicates a power peak for a specific period, it starts the task of processing this flexibility request in which it evaluates the feasibility of the request. To do so, it makes use of the proposed optimisation algorithm to decrease the load in a specific period by re-scheduling the shiftable devices to smooth power peaks while minimising the cost of electricity and the distortion of the operating time of the devices involved. The results obtained lead to the design of demand response events to solve these power peaks.

3.2.4 Investigation of pilot's real data for algorithms testing

Strategies and tools described in the previous sections will be applied to the offices that form part of the UMU pilot. The next step of the trial will be planning a flexibility strategy for the next months, starting from February 2022. The data analytics needed for this aim mainly concern the schedules of use of the HVAC system and the setpoint temperature chosen. The analysis is based on the period with an active heating system, i.e. months from November to February are considered. Real historical data have been analysed and a qualitative summary is presented in this section. Searching for a pattern of use, three different years have been compared. In particular, we used data from winter 2018/2019 and winter 2019/2020, while data from winter 2020/2021 were useless since the offices were not used due to the pandemic.

The raw data are obtained from the Context Broker, and they are referring to the terminal units of the HVAC system of the four main offices that compose the UMU pilot (namely, Office 14, Office 15, Office 16 and Office 17). In particular, the terminal units are Toshiba VRF and the data is collected from the internal drive of each terminal. The granularity of the raw data is 10 minutes. The data has been filtered in order to consider only the timesteps in which the devices were turned on.

For each office and each month, Table 4 summarises:

- the number of timesteps considered (timesteps in which the HVAC is turned on)
- the mean value
- the maximum value
- the minimum value
- the standard deviation

Table 4 – Analysis of the setpoint temperature set for the four main offices of the UMU pilot

Office	Month	Year	Mean setpoint Temperature (ON)	Max setpoint Temperature (ON)	Min setpoint Temperature (ON)	Count	STD
14	11	2018	25.13	29	23	538	2.76
14	11	2019	24.87	29	0	146	6.11
14	11	2021	26.24	29	23	479	2.27
14	12	2018	24.05	26	23	382	0.98
14	12	2019	26.65	29	22	353	1.58
14	12	2021	25.00	25	25	5	0.00
14	1	2019	24.91	26	22	708	1.05
14	1	2020	24.77	29	24	803	1.12
14	1	2022	24.77	26	23	378	0.73
14	2	2019	22.61	24	22	1395	0.92
14	2	2020	24.49	266	23	504	1.06
15	11	2018	24.70	25	22	10	0.95
15	11	2019	22.53	24	22	386	0.51
15	11	2021	22.26	24	22	809	0.49
15	12	2018	22.00	22	22	163	0.00
15	12	2019	NaN	NaN	NaN	0	NaN
15	12	2021	22.36	23	22	604	0.48
15	1	2019	22.00	22	22	784	0.00
15	1	2020	22.78	23	22	658	0.41
15	1	2022	22.80	23	22	282	0.40
15	2	2019	22.00	22	22	560	0.00
15	2	2020	22.32	24	22	326	0.56
16	11	2018	24.30	28	22	108	1.79
16	11	2019	22.85	24	21	142	0.62
16	11	2021	22.80	27	22	56	1.12

16	12	2018	23.39	25	22	558	0.73
16	12	2019	23.38	27	23	427	1.02
16	12	2021	24.39	25	23	66	0.93
16	1	2019	23.30	25	22	983	0.92
16	1	2020	24.77	28	24	274	0.73
16	1	2022	22.53	23	22	347	0.50
16	2	2019	23.65	25	23	478	0.80
16	2	2020	23.51	25	22	149	1.49
17	11	2018	23.72	25	21	586	1.01
17	11	2019	22.02	26	22	198	0.28
17	11	2021	23.11	26	21	1118	1.43
17	12	2018	21.65	23	21	861	0.67
17	12	2019	NaN	NaN	NaN	0	NaN
17	12	2021	23.96	25	23	880	0.56
17	1	2019	22.34	24	21	1485	1.05
17	1	2020	23.46	24	23	982	0.50
17	1	2022	25.04	28	25	918	0.34
17	2	2019	21.74	23	21	1278	0.88
17	2	2020	22.78	24	21	415	0.68

With respect to the analysis of the schedule, the time series of the three working weeks of January (one week is excluded due to holidays) are depicted in figures from Figure 7 to Figure 10, and it can be observed that in some cases it is not easy to find a pattern of use.

Trying to summarise, the schedules can be set to:

- Office 14: from 9:00 to 16:00
- Office 15: from 8:30 to 15:00
- Office 16: from 8:30 to 18:00
- Office 17: from 8:30 to 18:30

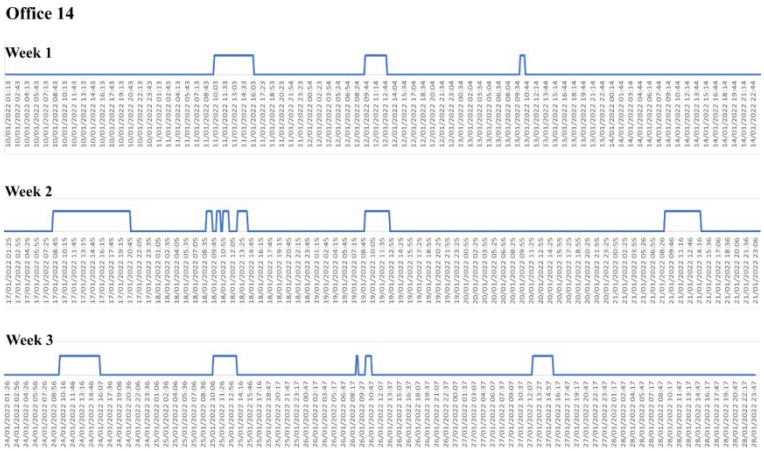

Figure 7 – Time series of the HVAC use (ON/OFF representation) (Office 14)

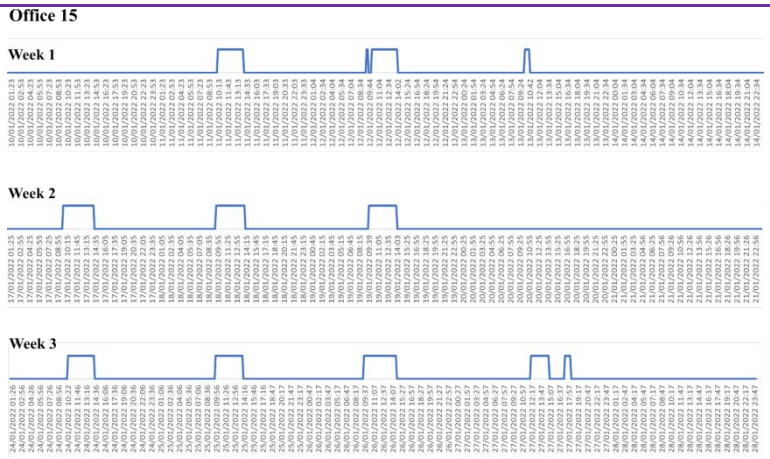

Figure 8 – Time series of the HVAC use (ON/OFF representation) (Office 15)

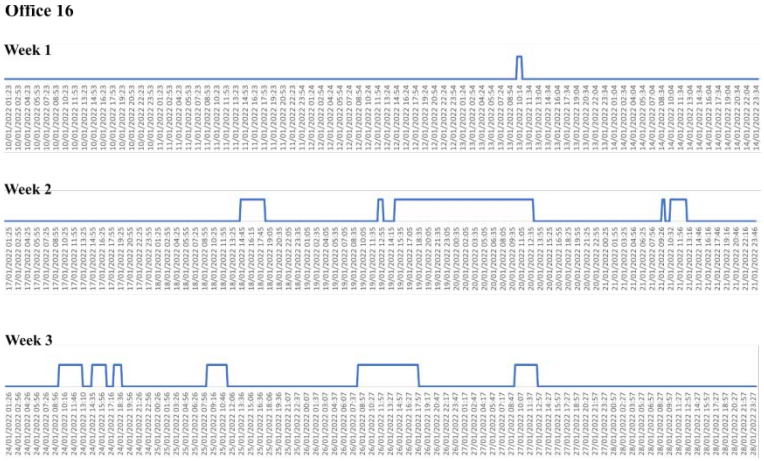

Figure 9 – Time series of the HVAC use (ON/OFF representation) (Office 16)

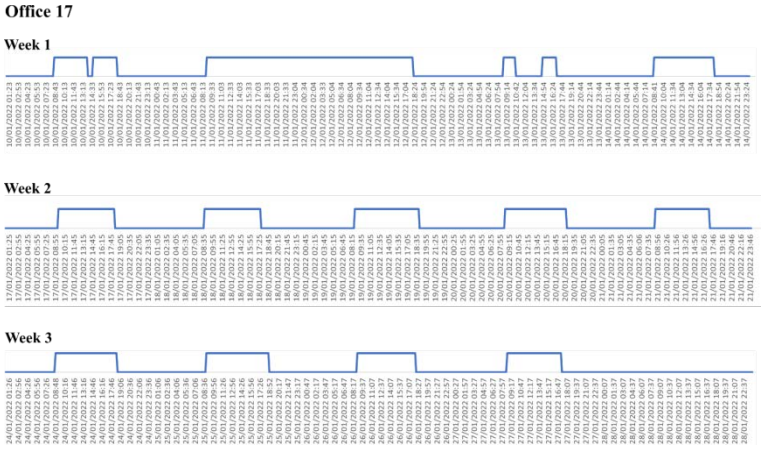

Figure 10 – Time series of the HVAC use (ON/OFF representation) (Office 17)

In the context of the pilot, a forecasting of the internal temperature of several rooms measured by the HVAC has been performed. Figure 11 shows the result of a forecasting algorithm performed in one of the rooms, using data from the full year 2021 to estimate the temperature in the first four months of 2022. This data brings great value to the design of Demand Response Events as it allows a more accurate assessment of HVAC control during a DRE, optimising the energy consumption and reducing the power peaks.

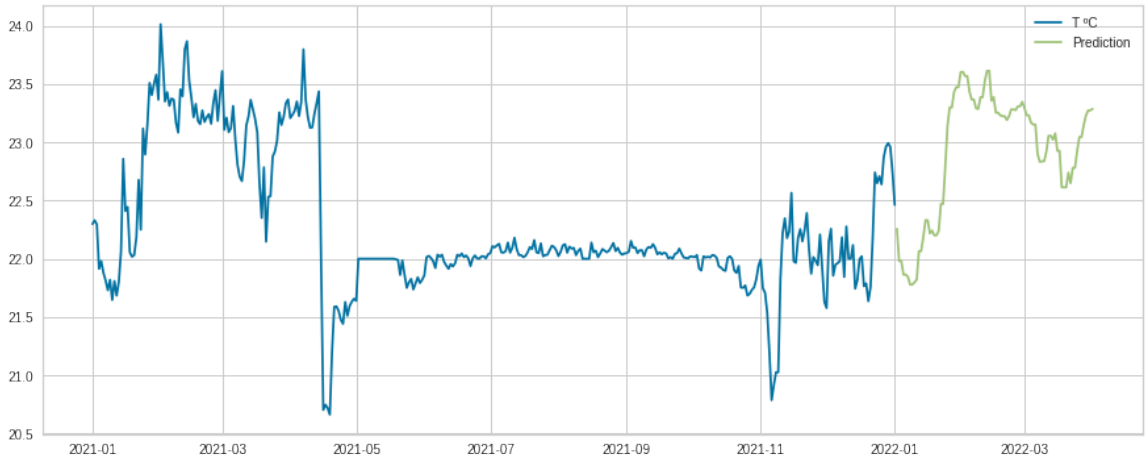

Figure 11 – Forecasting of room temperature measured by HVAC

4 Privacy and Security

4.1 Overview

At this point of the project, an amount of efforts have been made on offering mechanisms that cover the security requirements of the PHOENIX solution.

As stated in D4.1, the S&P framework will deal with both security and privacy, and so far the work done in the context of this deliverable has been mostly focused on security in order to guarantee that the integration of devices from pilots is fully operational for both reading and actuation.

4.2 Risk analysis associated to privacy/security features

This section describes what has been done in order to deal with the risks identified in D4.1 in the context of privacy and security.

4.2.1 Non-functional risks

The next table shows an updated version of the same table coming from D4.1 that includes the implemented solution to deal with each non-functional risk identified.

Table 5 – Non-functional risks in PHOENIX

ID	Name/Description + Solution implemented	Priority
NF_RI_1	**Accessibility** – S&P framework will provide the data access according to security and privacy policies. The data is available as long as the Service/Device fulfils the security requirements.	**HIGH**
NF_RI_2	**Availability** – S&P framework will be available 24/7. The security components are deployed as docker containers, are configured to be started automatically in case of server reset and are monitored to guarantee the containers are operational. If they're not, they are reset.	**HIGH**
NF_RI_3	**Backup** – S&P will include back-up procedures for storage facilities of all relevant data (e.g. security and privacy configurations, sensing data, etc.). A backup copy of the security and privacy configurations is done every time there is a change using GitLab. Other than that, the backup mechanism is executed at a server level.	**MEDIUM**
NF_RI_4	**Capacity** – S&P framework will manage a minimal group of <N> devices (to be defined at pilot level). This capacity has yet to be identified as some virtual devices (with their own entities in the Context Broker) are still being created related to the internal operation of some services. However the framework is managing all the devices developed in all the pilots.	**HIGH**
NF_RI_5	**Privacy** – S&P framework will provide privacy-preserving techniques to be compliant with the GDPR data protection regulation. The framework definition ensures that only those who have the right credentials are allowed to access to the stored data in its original deciphered version based on the policies used by the producer	**HIGH**
NF_RI_6	**Security** – S&P framework will include security-by-design mechanisms (i.e. authentication, authorization, channel protection, etc.) to ensure data access for the allowed entities (i.e. devices and services) according to security policies. This is guaranteed per-design by the security components' ecosystem	**HIGH**

	and by the use of secure (https) connections with automatic renovation of certificates.	
NF_RI_7	**Configurability** – S&P framework will provide mechanisms to configure security policies, privacy policies, entities, attributes, etc.	**HIGH**
	A configuration interface is available through the PAP.	
NF_RI_8	**Effectiveness** – S&P framework will be able to provide secure, private and trust exchanging of sensing/actuation data among different entities (i.e. devices and services).	**HIGH**
	The data can only be modified if the right policies owned by the corresponding user, service or device.	
NF_RI_9	**Extensibility** – S&P framework will be a modular system enabling to include new features and customizations.	**HIGH**
	The framework itself is designed to be deployed in a fully distributed scenario.	
NF_RI_10	**Interoperability** – S&P framework will include standards of Application Programming Interfaces (API) to facilitate the exchange with other entities (i.e. devices and services).	**HIGH**
	API documentation is provided to allow integration with devices and services.	
NF_RI_11	**Performance** – S&P framework will provide a good response time to interact with other entities in real time.	**HIGH**
	All the components of the framework respond in real time.	
NF_RI_12	**Scalability** – S&P framework will be able to guarantee the communication with a high number of devices and services.	**HIGH**
	The framework has been tested to handle requests from all the devices and services that are already operational	
NF_RI_13	**Reporting** – S&P framework will maintain a log of the operations performed.	**LOW**
	The components do have an internal logging system	

4.2.2 Functional risks

In this case, the next table shows an updated version of the table related to the functional risks coming from D4.1 and also includes the implemented solution to deal with each risk identified.

Table 6 – Functional risks in PHOENIX.

ID	Name/Description + Solution offered	Priority
F_RI_1	S&P framework will include operations to enable management (i.e. creation, modification and deleting) of security/privacy policies by the system administrator.	**HIGH**
	The management of policies is available using the PAP interface.	
F_RI_2	S&P framework will include operations to manage the identity and attributes of the involved entities (i.e. devices and services).	**HIGH**
	These attributes can be managed using the Keyrock interface.	
F_RI_3	S&P framework will include storage for real-time sensing/actuation data.	**HIGH**
	The real-time data is stored in the core components of the Smartness Hub.	
F_RI_4	S&P framework will provide storage for security/privacy policies.	**HIGH**
	The framework stores the policies in a persistent storage.	
F_RI_5	S&P framework will include operations to check security/privacy policies when data is requested by any entity.	**HIGH**
	The policies are validated in a distributed way every time a request to access any entity is received.	
F_RI_6	S&P framework will support context-aware access policies.	**HIGH**
	A context can be defined inside the framework based on a set of attributes of users and services.	
F_RI_7	S&P framework will provide capabilities to control the access of data according to the defined security and privacy policies.	**HIGH**
	Each Capability Token includes the set of capabilities that indicate which operation can be executed on the related resources/entities.	
F_RI_8	S&P framework will guarantee that security/privacy policies are not accessible publicly.	**HIGH**
	The policies can only be accessed using the administration interface.	
F_RI_9	S&P framework will incorporate standardized interfaces to communicate with third-parties entities (i.e. devices and services).	**HIGH**
	The framework provides an open API to use them and offers an endpoint for devices and services to access the data.	

F_RI_10	S&P framework will support the authentication of the devices and services before allowing the access to any data. The data can't be accessed unless the device or service is correctly authenticated (Identity Manager).	**HIGH**
F_RI_11	S&P framework will support the authorization of the devices and services before allowing the access to any data. Devices and services must be authorized to access the data (Capability Token + PEP Proxy)	**HIGH**
F_RI_12	S&P framework will provide channel protection (i.e. confidentiality, integrity and availability) to communicate with third-parties entities (i.e. devices and services). The data is always exchanged using secure channels (https) and the components are monitored to guarantee that they are restarted if a malfunction is detected at some point.	**HIGH**
F_RI_13	S&P framework will include the encryption of sensors/actuation data by public-key cryptography techniques. The data will be stored directly encrypted when CP-ABE is being used once its development is finished (privacy component, not included in this deliverable).	**HIGH**

4.3 Security Components

This section shows how the security components have been integrated in the PHOENIX project. The implemented technologies that will be detailed along the section are:

- Keyrock (FIWARE GE) for Identity Management (IDM) and authentication purposes.
- Distributed Capability-Based Access Control (DCapBAC) for authorisation purposes (access control).

The security components can be deployed using either Docker or Kubernetes technologies. In the PHOENIX Project, Docker has been selected. The containers can be launched both in the same environment as the rest of the PHOENIX components or out of it due to their distributed nature.

The next subsections will detail the following aspects related to the security components:

- Definition of the global architecture.
- Description of the main aspects (API, configuration requirements, etc.) of each component.

- Explanation of the interactions between the components and how to access a resource using the S&P framework.
- Details of the deployment of some components in the Smartness Hub.

4.3.1 Architecture

This section shows a global view of the security architecture and how it is connected with the PHOENIX components that make use of it, i.e. the Context Broker, the Z-Wace/WMP Actuation Agent and the Historical Data component.

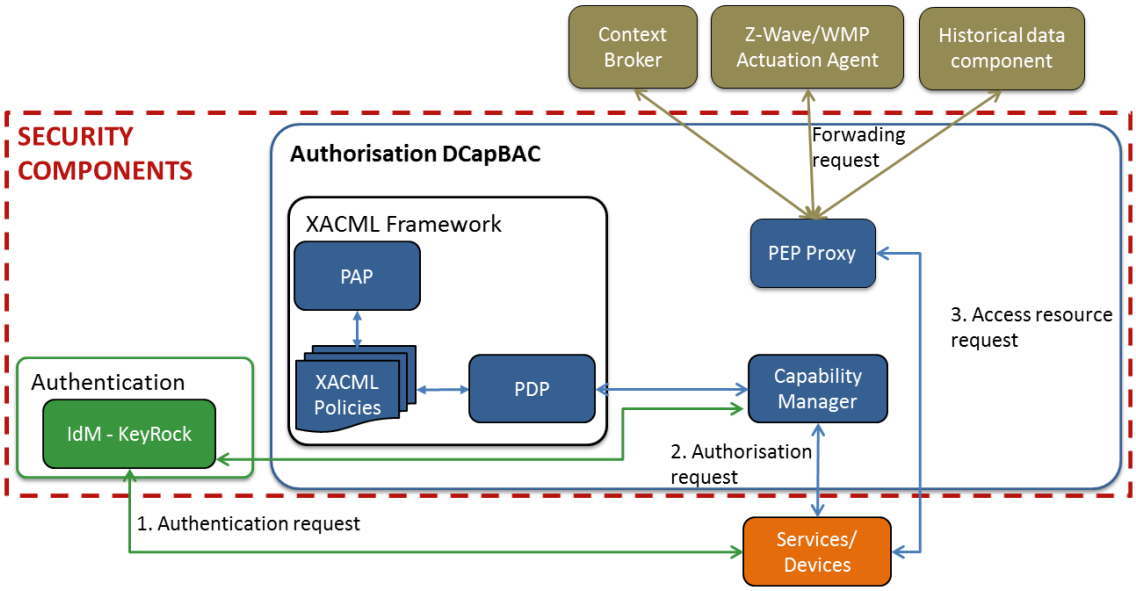

Figure 12 – Security components / Architecture

As shown in Figure 12, the security components are in front of the PHOENIX core components, which means the resources offered by them must be accessed through the security layer. From a security perspective, they offer both authentication and authorisation.

The component in charge of the authentication part is Keyrock (FIWARE Generic Enabler) while for authorisation, the Distributed Capability-Based Access Control technology (DCapBAC) is the one in control. DCapBAC itself includes a set of components, namely the XACML-Framework, Capability Manager and PEP Proxy.

Three requests will be required to get access to any resource, one for the authentication phase (Keyrock) and two for the authorisation access control phase (the authorisation request itself and the access to the resource). Any Service/Device must be authenticated (first request) before requesting authorisation (second request) and accessing the requested resource (third request).

4.3.2 Components

4.3.2.1 Keyrock

Keyrock (FIWARE GE Keyrock: https://fiware-idm.readthedocs.io/en/latest/index.html) is responsible for the Identity Management. It implements an OAuth2 REST API for managing identities in its repository as well as for authentication purposes. Further information can be found in the Keyrock Apiary (https://keyrock.docs.apiary.io).

This component allows creating users, organizations, applications, roles, assigning the users to organizations or application roles, etc. The Keyrock information is one of the pieces that the XACML framework uses to define access-control policies because the XACML policies are related to the entities defined by Keyrock (*subject* concept in the policy).

Keyrock entities can be managed not only through the OAuth2 REST API, but also using a web frontend.

Figure 13 shows the section of the Keyrock frontend with the list of the existing users.

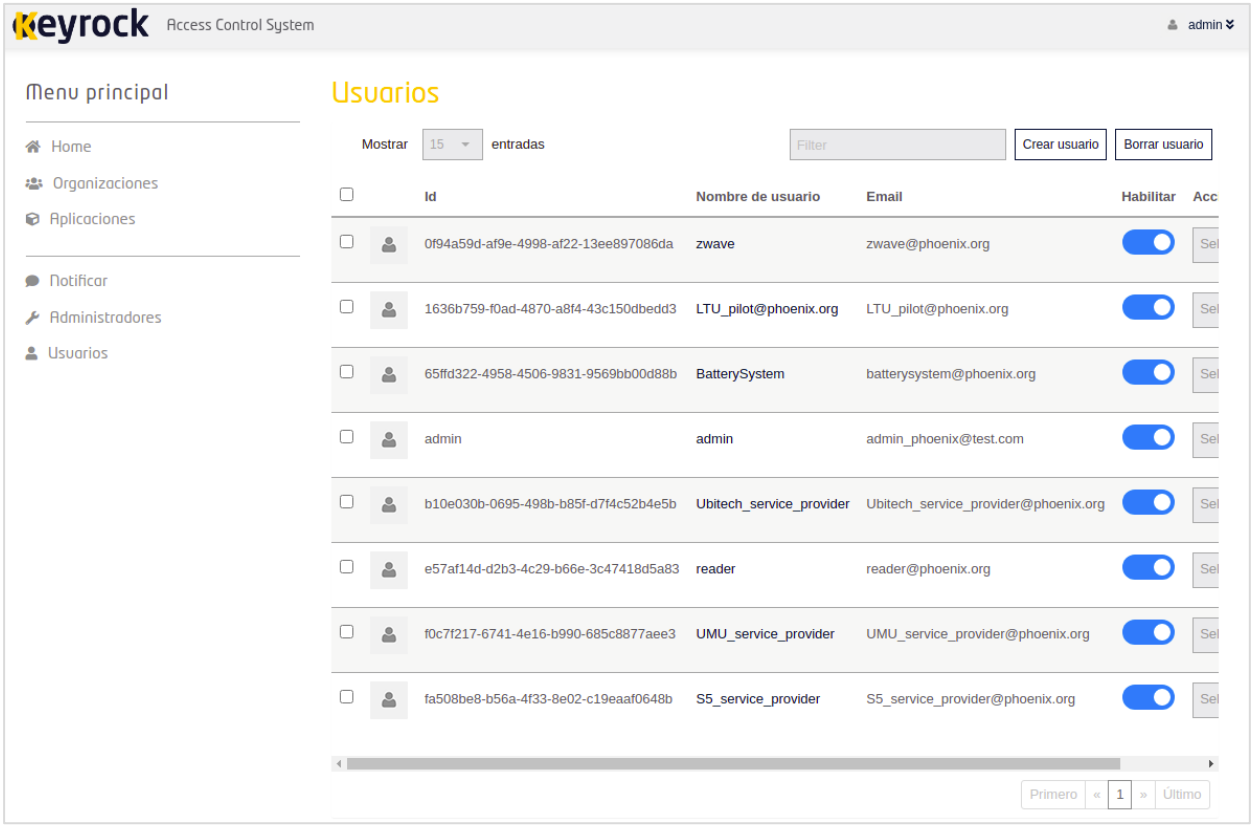

Figure 13 – Keyrock / User list

Once a Service/Device of PHOENIX is registered as a user in Keyrock, it can send an authentication request with the following format.

POST https://phoenix.odins.es:5443/v1/auth/tokens
Content-Type: application/json

```
{
  "name": "userEmail",
  "password": "userPassword"
}
```

The json payload includes two properties:

- name: the email of the registered user is used as identifier. For instance: LTU_pilot@phoenix.org
- password: the corresponding password.

In the response, if the credentials are wrong, a **401–"Unauthorized"** code response will be sent. Otherwise, if the authentication is successful, a **201-"Created"** code response will be returned and an IdM-Keyrock authentication token will be received in the **X-Subject-Token** http header of the response.

The following boxes show the formats of the different responses.

RESPONSE: HTTPS/1.1 201 Created
X-Subject-Token: 35ecf5d5-f9f5-417f-abe7-dc91cc13262f

```
{
   "token": {
      "methods": [
         "password"
      ],
      "expires_at": "2022-02-13T12:02:17.676Z"
   },
   "idm_authorization_config": {
      "level": "basic",
      "authzforce": false
   }
}
```

RESPONSE: HTTPS/1.1 401 Unauthorized

```
{
   "error": {
      "message": "Invalid email or password",
      "code": 401,
      "title": "Unauthorized"
   }
}
```

4.3.2.2 XACML/PDP + PAP

The XACML framework includes:

- The Policies file witch stores the XACML policies defined in the system.
- The Policy Administration Point (PAP), which allows the management of the XACML authorisation policies and can be accessed through a web frontend. These XACML policies are defined by a triplet (subject, resource, action). The PAP stores them in the XACML policies file. In this sense, this element is a configuration-only component and therefore doesn't interfere in the authorisation process.
- The Policy Decision Point (PDP), responsible for matching the authorisation requests with the XACML policies defined by the PAP, and issuing positive/negative verdicts accordingly.

Figure 14 illustrates the *Main view* of the PAP frontend.

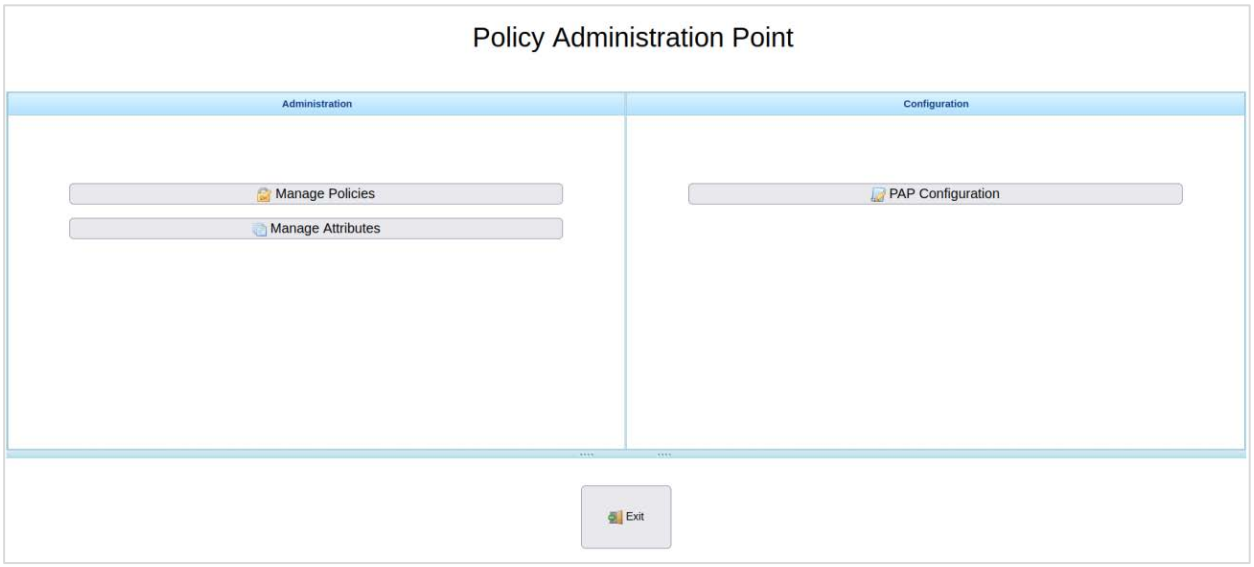

Figure 14 – PAP / Main view

The list of attributes required to create the XACML policies is available in the section *Manage Attributes* of the interface. These attributes must be previously defined in Keyrock.

Figure 15 shows the current attributes defined in the project.

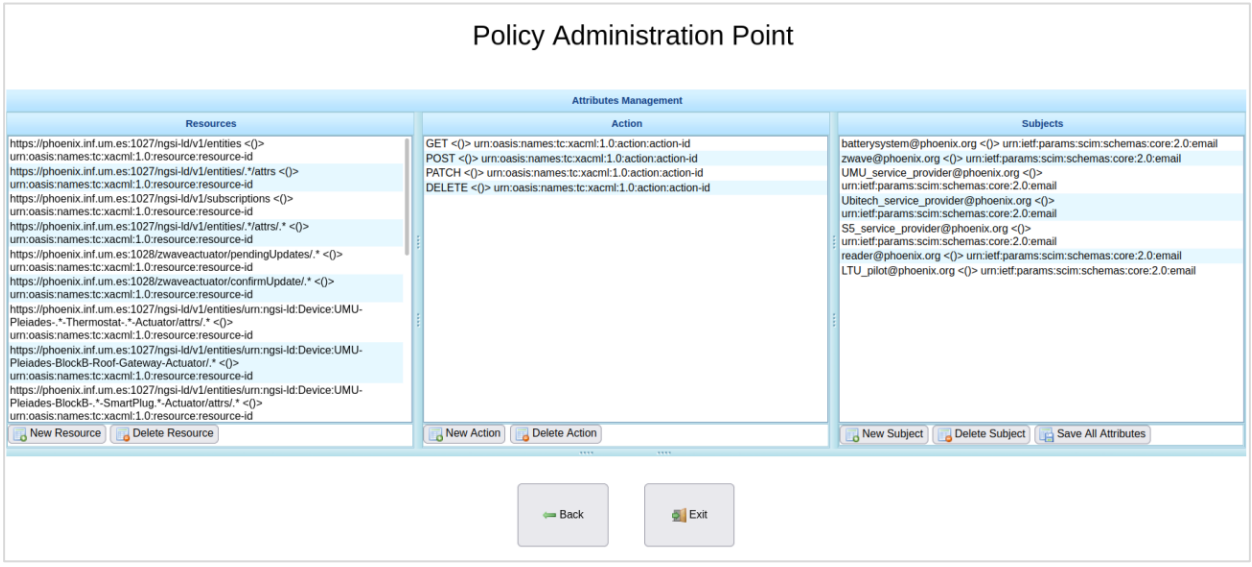

Figure 15 – PAP / Management of Attributes

Once the attributes have been defined and saved, and having gone again to the *Main view*, the policies can be managed in the *Manage Policies* section.

Figure 16 shows the current set of policies.

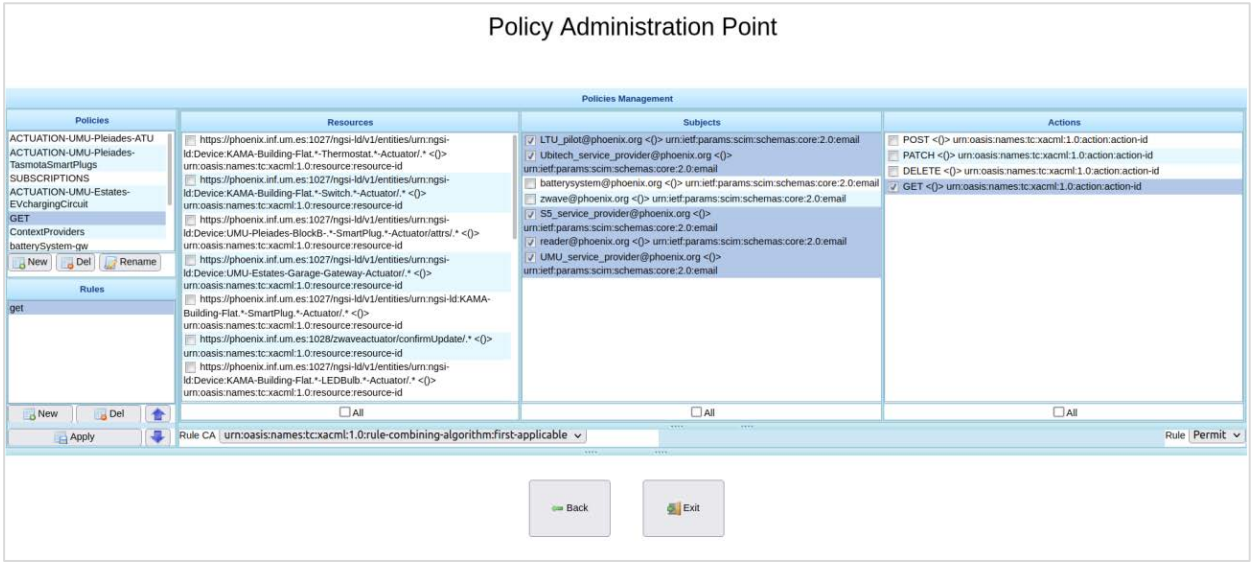

Figure 16 – PAP / Management of Policies

In this page Policies must be created first (top left) and then one or more rules can be created for each policy (left). Each rule can link resources, actions and subjects and establish whether this combination is *Permit* or *Deny* (bottom right).

Figure 16 shows an example of XACML policy defined where some users (*Subjects*) are allowed (*Rule=Permit*) to send *GET* requests (*Actions*) to some endpoints of the API of Context Broker (*Resources*).

Regarding the PDP, it offers an endpoint that returns the verdict to an authorisation request. It will try to match the request with the XACML policies to issue the verdict. The authorisation request follows this format:

POST http://phoenix.odins.es:8080/XACMLServletPDP/

Content-Type: text/plain

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
   <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
      <Attribute AttributeId=subjectType DataType="http://www.w3.org/2001/XMLSchema#string">
         <AttributeValue>subject</AttributeValue>
      </Attribute>
   </Subject>
   <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#string">
         <AttributeValue>resource</AttributeValue>
      </Attribute>
   </Resource>
   <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
       DataType="http://www.w3.org/2001/XMLSchema#string">
         <AttributeValue>action</AttributeValue>
      </Attribute>
   </Action>
   <Environment />
</Request>
```

Including:

- The **subject** and **subject's type** of the resource's request. These fields reference the information stored in Keyrock. The subject can be a user attribute (username, email, etc.), a specific organization, etc. although the email is being used right now in PHOENIX.
- The **resource**: endpoint (protocol + IP + PORT) + path of the resource's request.
- The **action**: method of the resource's request ("POST", "GET", "PATCH", "DELETE").

In the response, there are three possible verdicts: **Permit**, **NotApplicable** and **Deny**. All of them include **200 – "OK"** as response code.

The following boxes show the format of the different cases.

RESPONSE: HTTP/1.1 200 OK; Verdict = Permit

```
<Response>
   <Result ResourceID="resource">
      <Decision>Permit</Decision>
      <Status>
         <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
```

```
      </Status>
      <Obligations>
        <Obligation ObligationId="liveTime" FulfillOn="Permit">
        </Obligation>
      </Obligations>
   </Result>
</Response>
```

RESPONSE: HTTP/1.1 200 OK; Verdict = NotApplicable

```
<Response>
   <Result ResourceID="resource">
      <Decision>NotApplicable</Decision>
      <Status>
         <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
      </Status>
   </Result>
</Response>
```

RESPONSE: HTTP/1.1 200 OK; Verdict = Deny

```
<Response>
   <Result ResourceID="resource">
      <Decision>Deny</Decision>
      <Status>
         <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
      </Status>
   </Result>
</Response>
```

4.3.2.3 Capability Manager

In the DCapBAC scenario, the Capability Manager handles the first request of the authorisation access-control phase (the authorisation request).

It provides a REST API for receiving authorisation queries. These requests must include the authentication token already issued by Keyrock. When an authorisation query is received, the Capability Manager sends an authentication token validation to Keyrock and once the token has been validated, it creates an XACML authorisation request and sends it to the XACML-PDP engine. If a positive verdict has been received, it issues the Capability Token (authorisation token) which will be later used in the second authorisation request of the access-control process (access to the resource).

The next box shows the format of a sample request.

POST https://phoenix.odins.es:3030

Content-Type: application/json

```
{
"token": "authTokenReceivedFromKeyrock",
"de": "http://phoenix.odins.es:1030",
"ac": "GET",
"re": "/ngsi-ld/v1/entities/.*"
}
```

There are three possible responses shown in the following boxes, the first one with a valid answer (the token is received in the payload, as a json object in String format), the second one indicating a problem with the authentication token and the third one with the component indicating that the token couldn't be created (unknown reason).

RESPONSE: HTTPS/1.1 200 OK

"{Capability token}"

RESPONSE: HTTPS/1.1 401 Unauthorized

{"error": {"message": "Auth Token has expired", "code":401, "
title":"Unauthorized"}}

RESPONSE: HTTPS/1.1 500 Internal Server Error

"Can't generate capability token"

A sample request could include this information:

- token = valid Keyrock token (for instance: **35ecf5d5-f9f5-417f-abe7-dc91cc13262f**).
- device = https://phoenix.odins.es:1030
- action = **GET**
- resource = **/ngsi-ld/v1/entities/.***

Assuming a Keyrock token had been submitted to user **LTU_pilot@phoenix.org**, a possible response could be this:

```
{
    "id": "va6t2r5qet9p85tt3sf73ihcje",
    "ii": 1624449195,
    "is": "capabilitymanager@odins.es",
    "su": "LTU_pilot@phoenix.org",
    "de": "https://phoenix.odins.es:1030",
    "si":
"MEYCIQC7vXKpBaLd3N0jw5Sn1BLvVDGtLfeZQn0db3Ub9ZSInQIhAPY7CTDNpZVf8kLOOU7tRGEuFjXNKsxpWLvCs
1NG4mO+",
    "ar": [
```

```
        {
            "ac": "GET",
            "re": "/ngsi-ld/v1/entities/.*"
        }
    ],
    "nb": 1624450195,
    "na": 1624460195
}
```


In addition to other internal information, the json also includes two relevant attributes:

- **nb** (not before). Seconds UTC after the epoch to set the beginning of the validity of the token.
- **na** (not after). Seconds UTC after the epoch to set the end of the validity of the token.

4.3.2.4 PEP Proxy

The PEP Proxy is responsible for enforcing the authorisation when an access request to a resource is received (third and final request to access the data) directed to one of the PHOENIX components (Context Broker, Z-Wace/WMP Actuation Agent and Historical Data component). These queries must contain a *Capability Token* (sent as the value of a custom **x-auth-token** http header in the request) that will be validated by the PEP Proxy and, if the evaluation is positive, it will be forwarded to the corresponding API. The response will be forwarded back to the requester.

Based on this design, and considering that the only information analysed is an http header, the PEP Proxy supports any REST API.

4.3.3 Interactions

This section shows the complete flow that a Service/Device will have to carry out to gain access to the API offered by the PHOENIX components protected by the S&P framework. It represents a full valid exchange, assuming that valid credentials are being used and after having received valid and not expired tokens.

At this point there are a number of configuration steps that should be ready:

- The Services/Devices have been registered in the system using the credentials of an existing user of Keyrock.
- The XACML policies have been configured.

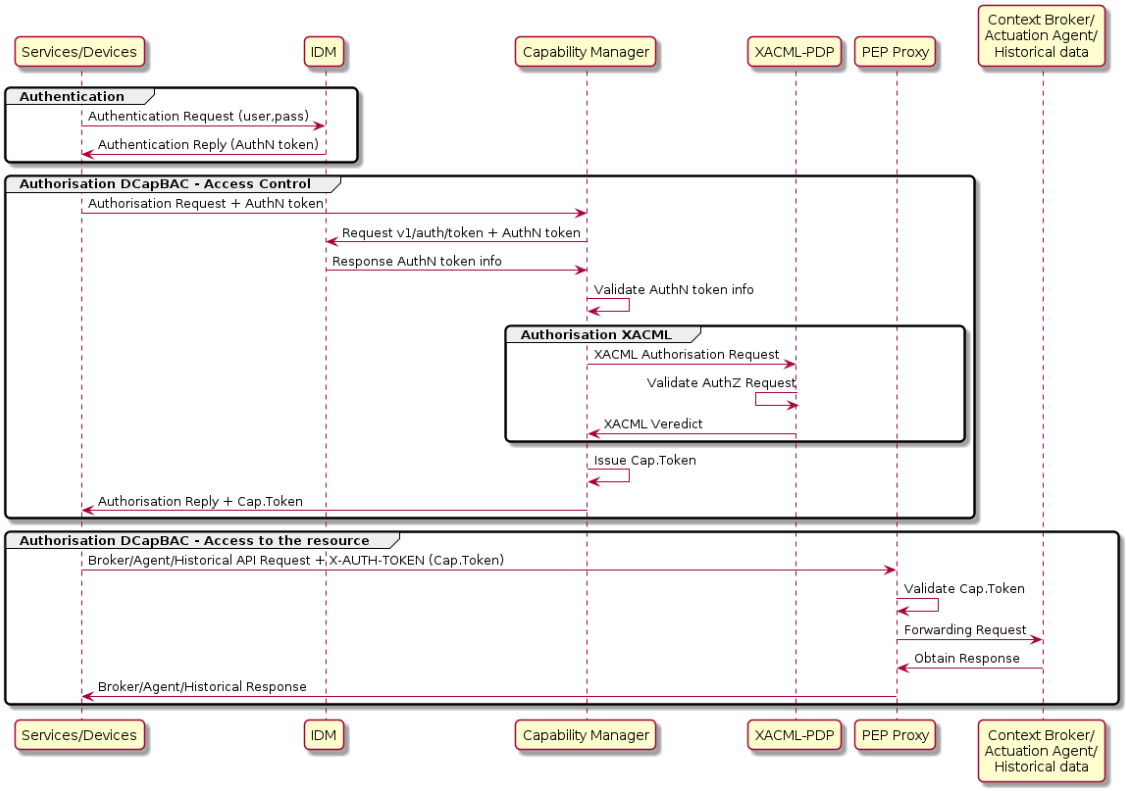

Figure 17 – Sequence diagram of a full secure access to a resource using the S&P framework

Following the previous diagram, the first request corresponds with the authentication phase. The Service/Device must send a POST request to **/v1/auth/tokens** (Keyrock) with its user credentials to obtain an authentication token (AuthN token). The format of the request was described in the Keyrock section. The Service/Device will receive this token in the **X-Subject-Token** header in the response.

The second request must be sent to the Capability Manager component of DCapBAC, which checks whether the Service/Device has access to the resource. As a result, an authorisation token (Capability Token) will be issued and will be returned as the payload of the response.

When a Service/Device sends a POST request to / (Capability Manager), it will extract the next information from it (as was described in the Capability Manager section):

- The authentication token (AuthN token).
- The base part of the request (protocol + IP + PORT). There will be a PEP Proxy listening in this address.
- The method of the request (“POST”, “GET”, “PATCH”, “DELETE”, “PUT”).
- The url of the request.

With this information the Capability Manager will:

- Access Keyrock to validate the authentication token. This access will validate if the token exists and whether it is expired or not.
- Access the XACML framework for validating authorisation requests (through PDP) and obtain the resource (verdict).
- Issue and return an authorisation token called *Capability Token* which is a signed json object that contains all the required information for the authorisation phase such as the resource to be accessed, the action to be performed and the validity of the token.

When an access resource request is received by PEP Proxy, it will:

- Extract the *Capability Token* from the **x-auth-token** header.
- Validate the token.
- Forward the request to the right component (Context Broker, etc.).
- Forward the response received to the requester.

4.3.4 Proxies deployed

Based on the distributed design of the security components and how the validation is done at the PEP Proxy using the *Capability Token*, each proxy has its own identity that must match the **de** attribute of the received token. This means one proxy can only protect one component and therefore it will be necessary to deploy as many proxies as components need to be protected.

In PHOENIX there are 3 components protected by their own instance of PEP Proxy.

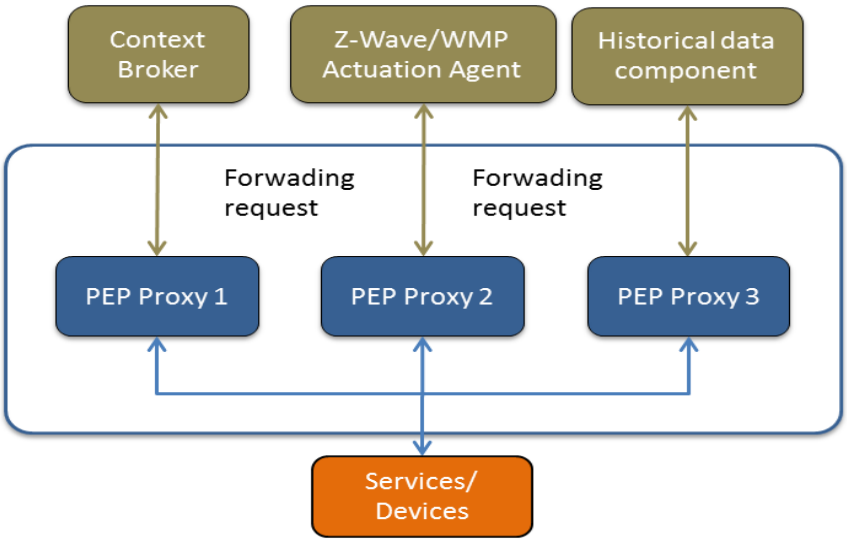

Figure 18 – Proxies deployed in the Smartness Hub

The base address for each PEP Proxy is detailed in the next table.

Table 7 – Deployed instances of PEP Proxy

Component	Base address of the PEP Proxy
Context Broker	http://phoenix.odins.es:1030
Z-Wave/WMP Actuation Agent	http://phoenix.odins.es:1028
Historical Data component	http://phoenix.odins.es:1029

4.4 Integration of different technologies requiring security and/or privacy

There are different devices that have their own requirements in terms of security and in this section a more detailed description of the special features of each integration is described.

4.4.1 Z-Wave / WMP

The integration of Z-Wave and Intesis WMP devices is divided in two parts, the Actuation Agent at the server and the client, executed at the IoT-Gateways using node-RED flows, which use different techniques for managing security for both actuation and sending sensor values.

4.4.1.1 Actuation

From a security perspective, the Actuation Agent must be accessed using the S&P framework in the way described in the previous sections.

As documented in D3.2, two resources are offered to the IoT Gateways:

- **/zwaveactuator/pendingUpdates** for getting the pending updates of the gateway (GET).
- **/zwaveactuator/confirmUpdate** for confirming that an update has been consumed by the gateway and can be removed from the list of pending updates from the Agent (POST).

First of all an authentication token must be requested to Keyrock. This process is repeated periodically.

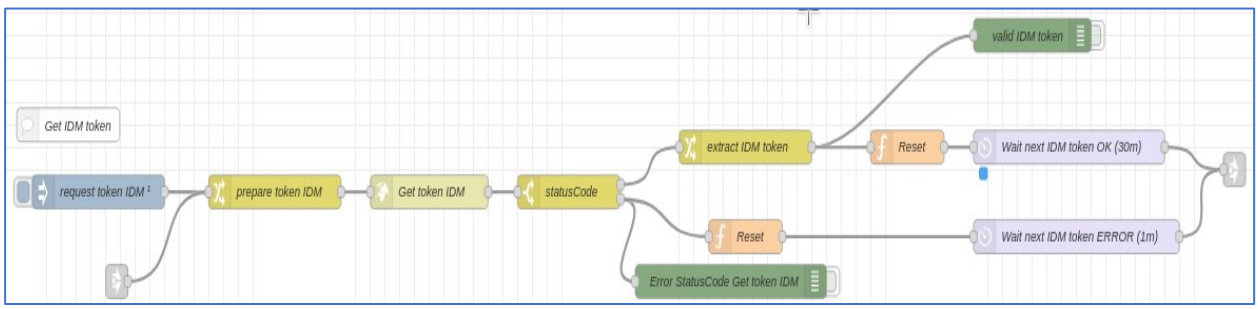

Figure 19 – Get authentication token

After that, also periodically, authorisation tokens must be obtained in order to get access to the resources provided by the Agent.

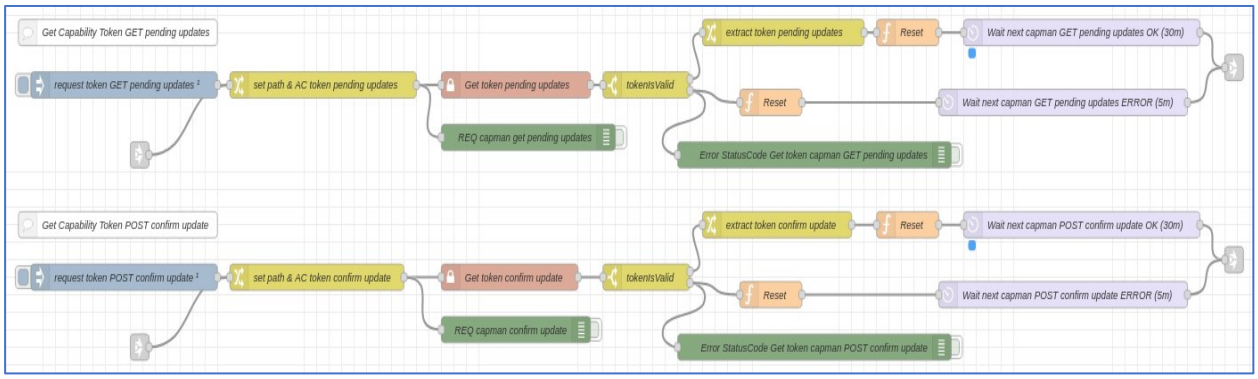

Figure 20 – Get authorisation tokens for actuation

Having valid tokens, the actuation process will be executed periodically. Here is where both authorisation tokens are used for communicating with the Agent through the PEP Proxy.

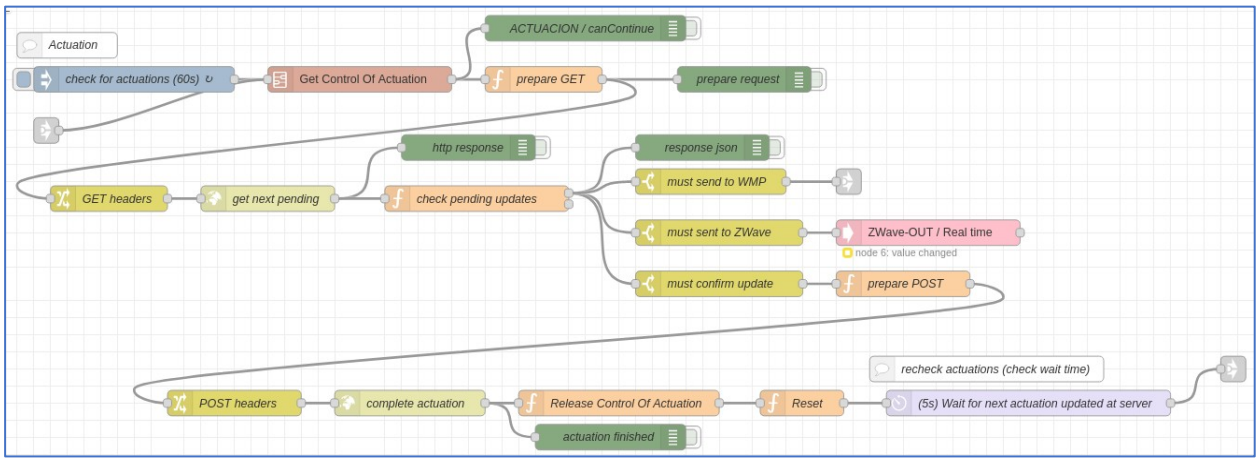

Figure 21 – Using the tokens (GET and POST) in the Actuation flow

4.4.1.2 Reading and publishing values

In the reading flow, a different authorisation token for publishing the values directly to the Context Broker must be obtained. In this case the PEP Proxy used for the communication is different, but the authentication token used is the same that was obtained following the instructions of the previous section as long as it's still valid.

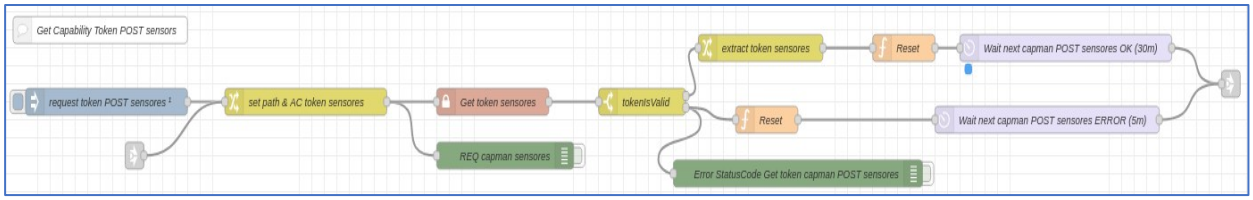

Figure 22 – Get authorisation tokens for publishing readings

Once a valid token has been received and there are readings pending to send, it is used for sending the values.

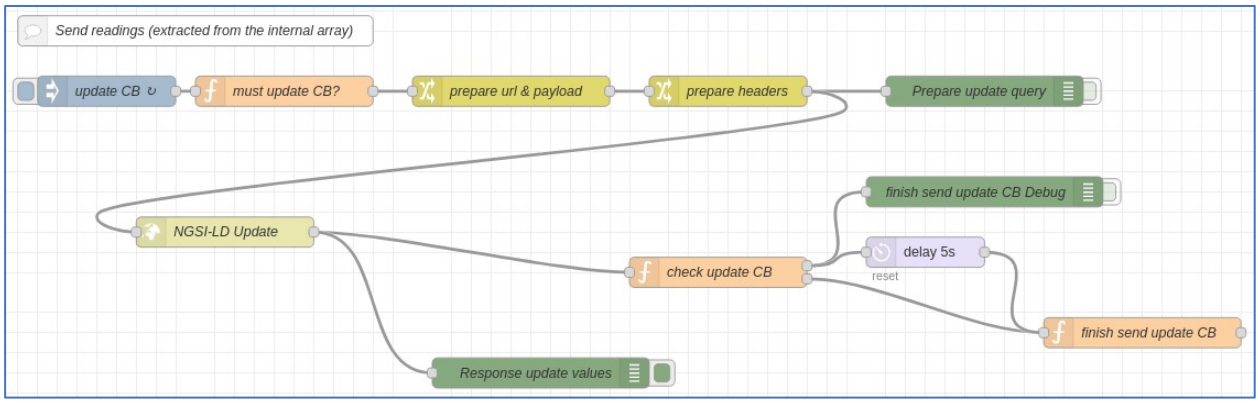

Figure 23 – Using the token to send readings (POST)

4.4.2 Platform to Platform (LTU pilot)

The only pilot that includes a direct integration with the Context Broker is the Swedish one.

The LTU platform uses another FIWARE Context Broker and an Agent is acting as a forwarded of information towards the PHOENIX Smartness Hub using NGSI-LD. Again this Agent must follow the set of steps required by the S&P framework to get access to the resources.

A sample publication is shown in the next box.

POST https://phoenix.odins.es:1030/ngsi-ld/v1/entityOperations/upsert/?options=update
Content-Type: application/json
fiware-service: phoenix
fiware-servicepath: /

```
[{
   "id": "urn:ngsi-ld:Observation:EntityId",
   "type": "http://purl.org/iot/ontology/iot-stream#StreamObservation",
   "http://www.w3.org/ns/sosa/hasResult": {
      "type": "Property",
      "value": "1782.171",
      "observedAt": "2021-09-22T10:00:00.000Z"
   }
}]
```

4.4.3 MQTT

The security aspects covered in MQTT integrations are:

- Connections are cyphered (TLS). This way privacy and integrity are guaranteed.
- Devices must be authenticated using their own credentials (login + password).

- Authorization is handled by the Access-Control list (ACL) mechanism provided by the MQTT broker, which offers a way to have a fine-grained control of the topics allowed for each device, IoT gateway or BMS.

This traditional security system has also been supported by the PHOENIX Smartness Hub to cover alternative scenarios where the devices are preconfigured or there is no way to integrate them with the S&P framework. Although it's not the preferred solution, this is accepted as long as all the privacy and security requirements are fulfilled.

4.5 Security Policies

The next tables show the list of security Policies defined so far in the framework (all of them grant permission to access the resource included in the Policy). There is one table per component that contains groups formed by one or more rows in grey background colour with the action and the resource of a Policy followed by 1 or more rows in white background colour with the subjects that have access to this combination. This is the base to create the set of XACML policies that will be used by the PDP.

Table 8 – Policies for the Context Broker

Context Broker (Device = https://phoenix.odins.es:1030)	
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:UMU-Pleiades-BlockB-Roof-Gateway-Actuator/.*
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:UMU-Pleiades-BlockB-.*-Thermostat-.*-Actuator/attrs/.*
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:UMU-Pleiades-BlockB-.*-SmartPlug.*-Actuator/attrs/.*
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:UMU-Estates-Garage-Gateway-Actuator/.*
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:KAMA-Building-Flat.*-SmartPlug.*-Actuator/.*
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:KAMA-Building-Flat.*-LEDBulb.*-Actuator/.*
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:KAMA-Building-Flat.*-Thermostat.*-Actuator/.*
PATCH	/ngsi-ld/v1/entities/urn:ngsi-ld:Device:KAMA-Building-Flat.*-Switch.*-Actuator/.*
PATCH	/ngsi-ld/v1/entities/MIW-.*-Actuator
PATCH	/ngsi-ld/v1/entities/SKE-.*-Actuator

UMU_service_provider@phoenix.org	
Ubitech_service_provider@phoenix.org	
S5_service_provider@phoenix.org	
POST	/ngsi-ld/v1/entities/.*/attrs
zwave@phoenix.org	
POST	/ngsi-ld/v1/entityOperations/upsert/.*
LTU_pilot@phoenix.org	
GET	/ngsi-ld/v1/entities
GET	/ngsi-ld/v1/entities/.*
GET	/ngsi-ld/v1/entities/.*/attrs
GET	/ngsi-ld/v1/entities/.*/attrs/.*
UMU_service_provider@phoenix.org	
Ubitech_service_provider@phoenix.org	
S5_service_provider@phoenix.org	
LTU_pilot@phoenix.org	
reader@phoenix.org	

Table 9 – Policies for the Actuation Agent

Actuation Agent (Device = https://phoenix.odins.es:1028)	
GET	/zwaveactuator/pendingUpdates/.*
POST	/zwaveactuator/confirmUpdate/.*
zwave@phoenix.org	

Table 10 – Policies for the Historical Data component

Historical Data component (Device = https://phoenix.odins.es:1029)	
GET	/ngsi-ld/v1/temporal/entities.*
UMU_service_provider@phoenix.org	
Ubitech_service_provider@phoenix.org	
S5_service_provider@phoenix.org	
LTU_pilot@phoenix.org	
reader@phoenix.org	

5 Conclusions

This document has described the work performed within WP4 in order to provide a first working version of the PHOENIX Smartness Hub.

At this stage, with the NGSI-LD interfaces operational and the components already making use of the Smart Data Models, the efforts on the modelling part have been done on extending the KG with the addition of a triplestore and the definition of a model for the SRI service.

At the functional layer, both user-centric and grid services are now working with real data once the pilots are ready and hence improvements can be obtained by analysing their outputs and by using them as relevant feedback for the implemented algorithms.

And finally, the Security & Privacy framework has received a significant upgrade that allows users, services and devices to communicate and have access to information in a secure way, always respecting the security policies.

6 References

[1] Smart Data Models https://smartdatamodels.org/

[2] Purl Ontology http://purl.org/iot/ontology/iot-stream

[3] IotCrawler https://iotcrawler.eu/

[4] Semantic Sensor Network Ontology https://www.w3.org/TR/vocab-ssn/

[5] Apache Jena Fuseki https://jena.apache.org/index.html

[6] The RDF Data Cube Vocabulary https://www.w3.org/TR/vocab-data-cube/

[7] Stochastic Processes https://www.sciencedirect.com/topics/neuroscience/stochastic-processes

[8] SAREF's extension for building domain https://saref.etsi.org/extensions.html#SAREF4BLDG

[9] BRICK's schema https://brickschema.org/

[10] https://www.iotacommunications.com/blog/how-can-you-improve-indoor-air-quality/

[11] https://www.meazurem.com/blog/how-indoor-humidity-and-temperature-affects-your-health/

[12] Usman Habib and Gerhard Zucker, Automatic occupancy prediction using unsupervised learning in buildings data, 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)

[13] Candanedo et al, A methodology based on Hidden Markov Models for occupancy detection and a case study in a low energy residential building, Energy and Buildings Volume 148, 1 August 2017, Pages 327-341

7 Annex I

A portion of the code that allows the connection to the Context Broker for the extraction and flattening of the data streams involved in the analysis process is shown below.

```
(…)

def retrieve_data_from_CB(Type, property_observed,ids_streams_list,dateStart, dateEnd):
    dfs = []
    for device in ids_streams_list:
                                                                                    urlAux
='{}?id={}&type={}&attr={}&timerel={}&time={}&endTime={}'.format(urlTIAMAT,device,Type,property_observed,'
between',dateStart,dateEnd)
        resp = requests.request("GET", urlAux, headers=cb_headers, data={})
        df=pd.DataFrame(resp[property_observed])[["value","observedAt"]]
        dfs.append(df)
    return dfs

def extract_flat_data(dfs):
    F = dfs[:]
    df_aggregated = pd.concat(F)
    return df_aggregated

(…)
```

The code shown below includes part of the connection to the Context Broker for the storage of the entities related to the algorithm execution results.

```
def post_cb(entity_payload):

    payload = json.dumps(entity_payload)
    response = requests.request("POST", urlCB, headers=cb_headers, data=payload)

    return response.status_code
```

Below is part of the code related to the multivariable generic forecasting algorithm developed.

```
def forecast_multivariable(data,all_dates):

    (...)

    j = 0
    for i in tqdm(all_ts):
        j += 1
        df_subset = data[data['time_series'] == i]
        # initialize setup from pycaret.regression
        s = setup(df_subset, target = 'value', train_size = 0.95,
                  data_split_shuffle = False, fold_strategy = 'timeseries', fold = 3,
                  ignore_features = ['observedAt', 'time_series'],
                  numeric_features = ['day_of_year', 'year',"hour","minute"],
                  categorical_features = ['month', 'day_of_week'],
                  silent = True, verbose = False, session_id = 123)

        # compare all models and select best one based on MAE
        best_model = compare_models(sort = 'MAE', verbose=False)
        # capture the compare result grid and store best model in list
        p = pull().iloc[0:1]
        p['time_series'] = str(i)
        all_results.append(p)
        # finalize model i.e. fit on entire data including test set
        f = finalize_model(best_model)
        # attach final model to a dictionary
        final_model[i] = f
        # save transformation pipeline and model as pickle file
```

```
        save_model(f, model_name='trained_models/' + str(j), verbose=False)

    (…)

    j = 0
    for i in tqdm(data['time_series'].unique()):
        j +=1
        l = load_model('trained_models/' + str(j), verbose=False)
        p = predict_model(l, data=score_df)
        p['time_series'] = i
        all_score_df.append(p)

    concat_df = pd.concat(all_score_df, axis=0)
    data["observedAt"] = pd.to_datetime(data["observedAt"].dt.strftime('%Y-%m-%d %H:%M' ))
    concat_df["observedAt"] = pd.to_datetime(concat_df["observedAt"].dt.strftime('%Y-%m-%d %H:%M' ))
     final_df = pd.merge(concat_df, data, how = 'left', left_on=['observedAt', 'time_series'], right_on =
['observedAt', 'time_series'])

    return final_df,concat_results
```

Finally, a fragment of proposed multi-objective optimisation algorithm is shown below.

```
X,Y = [],[]
n = 30
tie = np.array(user[0:n]).reshape(3*n)
n_var = len(tie)
lims = 96


xl = np.repeat(-lims,n_var)
xu = np.repeat(lims,n_var)
problem = FunctionalProblem(n_var,
                            PROB,
                            xl=xl,
                            xu=xu
                            )


algorithm = NSGA2(pop_size=300,
                  sampling=get_sampling("int_random"),
                  crossover=get_crossover("int_sbx"),
                  mutation=get_mutation("int_pm"),
                  eliminate_duplicates=True)
termination = MultiObjectiveSpaceToleranceTermination(tol=0.05,
                                                      n_last=100000,
                                                      nth_gen=5,
                                                      n_max_gen=1e20,
                                                      n_max_evals=None)
res2 = minimize(problem,
                algorithm,
                termination=termination,
                seed=1,
                save_history=True)
```

8 Annex II

The code snippets below refer to the default user comfort calculations:

```
def get_full_comfort_info(self, zone_id: str, user_id: str, metrics: list):
    """
    Build a json formatted dictionary with all relevant information for a list of comfort
    metrics, containing: current comfort state of the list of metrics, current measured values
    of related sensors, average values of related sensor values for the last 15 days
    :param `str` zone_id: The id of the zone/area to calculate comfort and get sensor values
    :param `str` user_id: The id of the user to calculate comfort status
    :param `list` metrics: The list of comfort metrics for which to calculate status and retrieve
        related sensor values
    :return `Response obj`: The django rest framework Response filled with data, or error
        message and the error status code
    """
    # fetch area's latest measurements
    response = get_comfort_related_observations_of_zone( zone_id, metrics)
    if response.status >= 500:
        error_message = response.error
        status_code = response.status
        logger.error('No sensor values could be retrieved for this area: "{}". \
            \nGot error message: {} with status code {}'.format(
            zone_id, error_message, status_code))
        # TODO: here return a not available label
        return Response(data=error_message, status=status_code)


    response_data = response.data
```

```
        # TODO: check first if the timestamp of current data is valid and then proceed to mapping,
        # otherwise return a not available lable
        current_sensor_values = map_properties_to_values(response_data)
        comfort_state = self.get_user_comfort_for_area(zone_id, user_id, metrics, current_sensor_values)
        # update the confort state dict with the current sensor values retrieved earlier
        comfort_state.update(current_sensor_values)
        # properties, i.e. "co2", mapped to observation ids
        mapped_property_ids = map_properties_to_ids(response_data)
        # aggregated values for each the properties
        aggregated_values = self.get_aggregated_values_for_metrics(
            mapped_property_ids, metrics)
        comfort_state.update(aggregated_values)
        return Response(data=comfort_state)
```

The code snippet below refers to the unsupervised clustering to acquire labels of occupancy for past data:

```
def occupancy_states_job():
    """
    This job will run once a week and will create a dataset of measurements of the last two weeks,
    label them as samples that denote an occupied (or not) area using clustering and then train a Random
Forest
    Classifier to be used for predictions in the upcoming week.
    """
    elastic_search_config = apps.get_app_config('elastic_search')
    elastic_status = elastic_search_config.elastic_status
    if elastic_status == 'down':
```

```
        logger.error("ElasticSearch not available. Cronjob will not run.")
        return
    else:
        elastic_connector = elastic_search_config.elastic_connector
    for demo in demo_pilots:
        build_response = get_demo_buildings(demo, filter_ids=True)
        if build_response.status >= 500:
            logger.error(
                f"Couldn't retrieve buildings of demo {demo}. An error occured: {build_response.error}")
            # continue to next demo
            continue
        building_ids = build_response.data
        index_env_name = demo_indices[demo]['zones_index']
        logger.debug(f"Index env variable name '{index_env_name}'")
        index = get_config(index_env_name)
        logger.debug(f"Index to be used: '{index}'")
        failed_buildings = []
        failures_per_building = {}
        for building in building_ids:
            zone_response = get_building_residential_zones(building, True)
            if zone_response.status >= 500:
                logger.error(
                    f"Couldn't retrieve zones of building {building}. An error occured:
{zone_response.error}")
                failed_buildings.append(building)
                # continue to next building
                continue
```

```
            zone_lst = zone_response.data
            logger.debug(f"Residential zones: {zone_lst}")
            if len(zone_lst) == 0:
                # the specific building has no residential zones
                continue
            # only if the demo building contains residential zones there is a point for checking the
            # respective index existence and creating it if not available
            if not elastic_connector.check_index_existence(index):
                logger.info(
                    f"Index {index} was not found. It will be created.")
                mapping = zones_index_mapping
                elastic_connector.create_index(index, mapping)
            failed_zones = []
            for zone_id in zone_lst:
                # TODO: we could find a way to choose which types of observations will be used
                metrics = ['thermal', 'humidity', 'visual', 'air']
                # metrics = ['thermal', 'visual', 'air']
                observ_response = get_comfort_related_observations_of_zone(
                    zone_id, metrics)
                if observ_response.status >= 500:
                    logger.error(
                        f"Couldn't retrieve observations of zone {zone_id}. An error occured:
{observ_response.error}")
                    failed_zones.append(zone_id)
                    # continue to next zone
                    continue
```

```
observations = observ_response.data

# get the observation ids for the respective properties

property_id_map = map_properties_to_ids(

    observations, key_value=True)

dates = DatesHelper.calculate_dates_list(

    num_of_days=14,

    fmt='%Y-%m-%d',

    last_date=DatesHelper.get_previous_day_datetime()

)



# get the list of the of the dataframes of each observation

observations_dfs = _load_dataframes(property_id_map, dates)

# reindex df and in parallel complete the timestamps missing

# reindexed_dfs = \

#     [_reindex_dataframe(df) for df in observations_dfs]

reindexed_dfs = _reindex_dataframes(observations_dfs)

# combine all dfs into one by matching their timestamps

merged_dfs = _merge_dataframes(reindexed_dfs)



counted_nas = _calculate_nas(merged_dfs)

# missing timestamps percentage

total_measurements = len(

    merged_dfs.index)*len(merged_dfs.columns)

missing_ts_perc = round(

    (sum(counted_nas)/total_measurements)*100, 2)

logger.debug(
```

```
                    f"NAs percentage in df {missing_ts_perc}%")

                # preprocess the data and keep the first to be enriched later with labels and

                # the second only to be used for the clustering procedure

                full_df, processed_df = _df_preprocessing(merged_dfs, demo)

                # clustering of timeseries samples

                clustering_job = KMeansClustering(cluster_num=2)

                clustering_job.run_clustering(processed_df)

                final_df = _define_df_labels(full_df, clustering_job)

                # _make_plots(final_df, len(final_df.columns), zone_id)

                # classification training

                classification_job = RFClassification()

                classification_job.train_classifier(final_df, y_label='state')

                accuracy, recall = classification_job.get_scores()

                logger.info(

                    f"Test scores after training: Accuracy={accuracy}, Recall={recall}")



                # load zone document

                zone_data = get_zone_data_by_id(zone_id)

                if len(zone_data) == 0:

                    # zone doc wasn't found, it will be initialized

                    zone_data = _init_zone_document(building)

                # fill doc with occupancy data

                zone_data['occupancy_model'] = _fill_occupancy_model_data(

                    classification_job.get_rfc_model_base64(),              list(property_id_map.keys()),
round(accuracy*100, 2), round(recall*100, 2), missing_ts_perc)

                # store back the zone document

                elastic_connector.write_to_index(index, zone_id, zone_data)
```

```
            # log and add any zone failures

            if len(failed_zones) != 0:

                logger.error(f'Demo {demo} --- Modeling failed for {len(failed_zones)}/{len(zone_lst)}
residential zones of building {building}.\n'

                             f'Failed zones: {failed_zones}')

                failures_per_building.update({building: failed_zones})

        # final logs for demo

        if len(failed_buildings)+len(failures_per_building) == 0:

            logger.info(

                f'Demo {demo} --- Modeling completed for all residential zones and all buildings.')

        else:

            if len(failed_buildings) != 0:

                logger.error(

                    f"Demo {demo} --- Failed to fetch zones (residential or not) for some buildings of
demo: {failed_buildings}")

            if len(failures_per_building) != 0:

                logger.error(

                    f'Demo {demo} --- Modeling failed for some zones of buildings:
{failures_per_building}')
```