WP6 – Synergic grid interaction and automatic energy savings

Document Version:

D6.2 Refinements in services for energy utilities and the grid - Initial Version

0.7

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**:**
31/03/2022	30/03/2022	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:
Dimitra Georgakaki	Ubitech	WP6

Authors (organizations):

UMU, UBITECH, ODINS

Abstract:

This document details the work achieved since D6.1 regarding the development, integration and testing of various grid services and their integration in the PHOENIX ecosystem. Significant progress has been made in the development of services for grid flexibility in combination with self-consumption as well as the testing framework for the API testing of these services.

Keywords:

Grid communication, grid flexibility, self-consumption, integration, testing

Revision History

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

Revision	Date	Description	Author (Organization)
0.1	03/03/2022	ToC Draft	Dimitra Georgakaki (Ubitech)
0.2	10/03/2022	Section 5	Alfredo Quesada (OdinS)
0.3	11/03/2022	Sections 2, 3	Alfonso Ramallo (UMU)
0.4	13/03/2022	Section 4	Dimitra Georgakaki (Ubitech)
0.5	15/03/2022	Sections 2, 3	Alfonso Ramallo (UMU)
0.6	24/03/2022	Compiled version	Dimitra Georgakaki (Ubitech)
0.7	30/03/2022	Final version	Dimitra Georgakaki (Ubitech)

Executive Summary

This document is the second part of the set of deliverables regarding grid services in the PHOENIX Smartness Hub or the PHOENIX platform. The main beneficiaries of these kind of services are the Aggregators, the ESCOs and the building managers (and of course indirectly the consumers/prosumers).

The communication of the various stakeholders with the grid Demand Response requests is simulated with the help of USEF framework interacting with PHOENIX, serving as a support tool for the aggregator in terms of flexibility, providing information from active consumers/prosumers and their devices able to adjust their demand to alleviate grid congestion peaks.

The evaluation and the execution of the available demand adjustment in the ecosystem of controlled devices is performed by the Demand Flexibility Management Engine, which works in conjunction with the Self-Consumption Optimization Engine, the latter having as primary objective to shift battery and EV usage in timeframes of maximum building energy generation.

In case of generated energy that has not been consumed by a battery or an EV, the excess of energy is ingested directly into the Demand Flexibility Management Engine and instructs one or more load shifts (if possible) in order for the building not to lose this generated energy surplus.

All the aforementioned energy services by interacting with the Energy Stakeholders Visualization Dashboard, provide valuable insights in the form of charts, tables and metrics, aiming at helping all the beneficiaries acquire knowledge about their consumers/prosumers energy behaviour and how this can be further improved to minimize more energy costs.

Finally, all the integration and testing activities of these services are carefully designed and implemented with the aim to provide an end-to-end state of the whole solution.

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

1 Introduction 11

1.1 Scope of the document 11

1.2 Relevance to other deliverables 11

1.3 Structure of the document 12

2 Interface for communication with the grid and smart contracts 13

2.1 Introduction 13

2.2 Grid communication definition activities 13

2.3 Grid communication development activities 17

2.4 Smart contracts integration 20

3 Services for grid flexibility oriented to demand adjustment 22

3.1 Introduction 22

3.2 First version of Demand Flexibility Management Engine 22

3.3 Demand Flexibility Management Engine Integration 24

3.4 Demand Flexibility Management Engine Workflow 30

4 Services for self-generation and energy storage 33

4.1 Introduction 33

4.2 Optimization problem 34

4.3 Technical implementation and deployment of the solution 35

4.4 Visualization of Results in the Energy Stakeholders Dashboard 37

5 Integration, testing and refinement 339

5.1 Integration 311

5.2 Testing 311

5.2.1 Description of a test 311

5.2.2 Definition of a stage 40

6 Conclusions 43

7 References 44

8 Annex I – Testing with Jenkins 45

8.1 Main functionality 45

8.2 Jenkinsfile 47

8.3 Test implementation 411

8.4 Test deployment 411

Table of Figures

Figure 1 PHOENIX role in the USEF Flexibility Value Chain for explicit Distributed Flexibility

Figure 2 USEF market-based coordination mechanism (MCM)

Figure 3 FlexRequest content

Figure 4 FlexOffer content

Figure 5 USEF Interpreter Integration

Figure 6 FlexRequest Translation

Figure 7 FlexOffer Translation

Figure 8 Smart Contracts design flow

Figure 9 Demand Flexibility Engine Overview

Figure 10 – PHOENIX role in the USEF Flexibility Value Chain for explicit Distributed Flexibility

Figure 11 Demand Flexibility Management Engine and Algorithms Engine Communication

Figure 12 FlexConfiguration entity

Figure 13 Demand Flexibility Configuration Panel Mock-up

Figure 14 Current Demand Flexibility Configuration Panel

Figure 15 Flexibility Negotiation Workflow

Figure 16 FlexOrder: flexibility execution workflow

Figure 17 Interactions of self-consumption optimization engine with other components

Figure 18 Historic Energy consumption of UMU-Pleiades

Figure 19 Historic Energy generation from the PV installed in UMU-Pool-Parking

Figure 20 Charging patterns of EVs in UMU

Figure 21 Jenkins frontend

Figure 22 Creating a job

Figure 23 Configuration of a job

Table of Tables

Table 1 Type of devices for flexibility

Table 2 Definition of a test

Table 3 Definition of a stage

Acronyms

Abbreviation	Description
AGR	Aggregator
BRP	Balance Responsible Party
DER	Distributed Energy Resource
DoA	Description of Action
DSO	Distribution System Operator
ISP	Imbalance Settlement Period
MCM	Market-based Coordination Mechanism
TSO	Transmission System Operator
UFTP	UDP-based File Transfer Protocol
USEF	Universal Smart Energy Framework
XML	Extensible Markup Language
WP	Work Package

1 Introduction

1.1 Scope of the document

This deliverable (D6.2) is the second version of the services aimed for energy utilities and the grid and continues the work done in D6.1. It mainly addresses developments on the following objectives of WP6:

- **Design of a communication platform for different stakeholders in the energy sector** (covered in section 2 of the document).
- **Implementation of mechanism for the implementation of demand response programs** (covered in section 3 of the document).
- **Development of services for energy production and storage in buildings** (covered in section 4 of the document).
- **Visualization results of historic, nowcast and forecast energy consumption, production and storage in buildings** (covered in sections 3-4 of the document).

In a nutshell, the document tries to present a more holistic view on the problems of self-consumption and participation in explicit demand response programs. Prosumers that participate in explicit flexibility but do not have a battery or EV installed in their premises, can declare a set of flexible loads and the corresponding optimization algorithm shifts them accordingly in order to serve DR events coming from the USEF interface that simulates the grid requests. On the other hand, prosumers with a battery or EV installed in their premises, prioritize the self-consumption and in case production exceeds charging needs, the excess amount of energy can be directed to load shifting.

1.2 Relevance to other deliverables

D6.2 is directly linked to WP3, WP4, WP6 and WP8. More specifically, all the sensor, meter and device data resulting from the work carried out in WP3, serve as input for the for the implementation of the business energy services. In D4.2, data analytics methods for grid integration services have been documented, with a special emphasis on the algorithms engine which is the core component of the forecasting algorithms for generation and consumption.

D6.1 was the first version of the services aimed for energy utilities and the grid, so this deliverable, D6.2, also demonstrates the progress of work done in that aspect. Finally, the outcomes of D6.2 will serve as a direct input for D8.2 in the definition of the novel business models that the Phoenix project will offer to the energy stakeholders.

1.3 Structure of the document

The document follows the structure of WP6. Section 2 is the outcome of Task 6.1 where the integration of USEF interface in the Phoenix ecosystem is presented. The communication between the grid and the Phoenix platform is of crucial importance for the delivery and execution of DR requests. Section 3 represents Task 6.2 in the DoA and mainly describes the development and integration of the flexibility engine and the algorithms engine, which together act on the load shifting of certain flexible devices, introduced in the flexibility registry table of the building managers dashboard. Section 4 represents Task 6.3, where the self-consumption maximization problem is analysed taking into account in principle energy generation, energy storage and energy demand from electric vehicles. An interaction with the flexibility engine is also described in cases of produced energy that is not self-consumed from DERs. Section 5 is relevant to Task 6.4, where all the aforementioned services are integrated together and exchange information if needed via the Orion Context Broker. Finally, section 6 reports the main conclusions of the deliverable and section 7 the references used.

2 Interface for communication with the grid and smart contracts

2.1 Introduction

As reflected in T6.1, the scope for this task is the design of a communication interface with the grid for the integration of the different grid agents such as aggregators, or DSOs in the PHOENIX platform and the definition and development of services related to smart contracts and rate plans.

The design of the communication interface goes from the definition of the communication mechanism and protocol to the deployment of the necessary components for the integration of the grid agents in the platform. The components and mechanisms necessary to facilitate grid interoperability have already been developed for this version of the document.

On the other hand, the services that enable the creation and proposal of smart contracts and rate plans to building occupants are under design at this time of the project and a brief description of their current conceptualisation is described on this document.

2.2 Grid communication definition activities

The evolution towards the new electricity market paradigm requires a series of innovations at all levels of the electricity supply chain, from the TSO to the prosumer. As stated in D6.1, USEF [1] proposes an innovative role model that contemplates this new paradigm in which the consumer becomes a prosumer and is able to offer flexibility to the grid. This new role of end-users allows the establishment of a new business model based on incentives, promoting new tariffs and contracts in which demand adjustment is contemplated.

Flexibility is therefore the main key to the new interactions between the different grid agents. The main figure in this role model is the Aggregator, whose role is to accumulate the flexibility of the active consumers and sell it to the BRP, the DSO or the TSO. Therefore, PHOENIX serves as a support tool for the aggregator in terms of flexibility, providing information from active consumers and their devices able to adjust their demand to alleviate congestion peaks reported by the grid. PHOENIX does not interfere in the market interactions between Aggregator, DSO, TSO or supplier, but provides the necessary tool for the aggregator to assess the available flexibility of its prosumers and execute such flexibility orders.

Figure 1 shows the flexibility chain offered by USEF and where the PHOENIX framework is located.

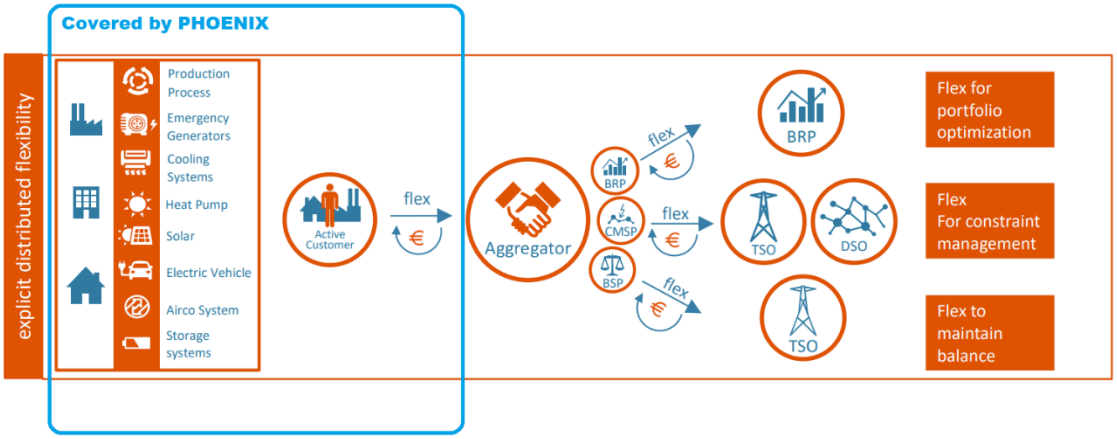

Figure 1 PHOENIX role in the USEF Flexibility Value Chain for explicit Distributed Flexibility

Since PHOENIX covers consumer energy management, this helps to overcome some of the barriers defined in D6.1. One of the barriers to grid flexibility was the delivery of data to the grid about the operating and consumption pattern of end-consumer appliances, compromising the privacy of active consumers.

In the proposed infrastructure, PHOENIX sits between active consumer and Aggregator, keeping all consumer-related information within the PHOENIX framework and providing only the necessary information to the aggregator in an anonymised form.

The aggregator operates under the USEF framework, it makes use of the different standards, structures and tools offered by this framework where PHOENIX is not involved. Instead, to facilitate interoperability in the flexibility negotiation and execution process in which PHOENIX participates as a support for the aggregator, it is necessary to use common mechanisms that facilitate the task of each of the elements of the flexibility supply chain. For this purpose, the definition of the communication messages between PHOENIX and the aggregator are based on those defined by USEF.

USEF proposes an interaction mechanism between AGR and DSO for flexibility trading. This mechanism is called the USEF market-based coordination mechanism (MCM) and is based on distinguishable phases. Figure 2 shows a diagram with the phases corresponding to the USEF system.

Figure 2 USEF market-based coordination mechanism (MCM)

As mentioned above, PHOENIX does not participate in the flexibility trading process with the DSO but provides support in assessing the available flexibility and in executing the agreed flexibility through demand response events within the PHOENIX framework. Therefore, PHOENIX participates as a tool of the aggregator in the Validate and Operate phases.

The trading process in these phases is as follows:

1. DSO estimates a peak load at a congestion point and creates a flexibility request indicating the magnitude (amount of excess power) and the time period (ISP) in which congestion is expected. This is handled by a FlexRequest message, which is sent to the aggregator.
2. The aggregator receives the flexibility request via FlexRequest and forwards it to PHOENIX as it is the consumers energy management.
3. PHOENIX evaluates the feasibility of the received request and proposes flexibility offers indicating the amount of flexibility that is able to achieve by changing the load/generation profile. This is handled by the FlexOffer message.
4. The aggregator receives the FlexOffers and forwards them to the DSO.
5. DSO evaluates the received FlexOffers and issues a FlexOrder for those that fit its needs. The content of the FlexOrder message is the same as the previous FlexOffer, as it is a confirmation.
6. The aggregator receives the FlexOrder and forwards it to PHOENIX, which is in charge of performing the flexibility process through Demand Response Events on the active consumer's appliances.

Specifically, these communication mechanisms between Aggregator and DSO are defined by the USEF Flex Trading Protocol [2]. This protocol can be used for flexibility forecasting, offering, ordering, and settlement processes.

In terms of time granularity, USEF makes use of the Imbalance Settlement Period (ISP). ISP is the time unit for which the imbalance of the balance responsible parties is calculated. It indicates an interval of 15 minutes, with 96 ISPs per day. Flexibility requests, offers and orders are based on ISPs [2]. ISP is therefore the unit of time used in the context of flexibility and grid interaction.

Figure 3 shows the content of the FlexRequest message. In these examples, the first 6 ISPs of the day are shown for simplicity. However, the implementation requires to indicate the 96 ISPs relative to the whole day (15 minutes per ISP) on our engine as this facilitates the implementation and makes the module compatible with the self-generation engine. FlexRequest includes ISPs with a request to reduce power consumption. In addition, FlexRequest may include ISPs with the available space to deviate from the power consumption. In this way, load reduction on the requested ISPs can be performed by a demand response event in which the consumption of the users' appliances is controlled or even divert the load in time to the available ISPs. In the example shown, a load reduction of 5kW is required for ISP number 5. In addition, some of this load relief can be shifted to the other available ISPs.

Figure 3 FlexRequest content

Figure 4 shows the content of a FlexOffer message in response to the previous FlexRequest.

It shows how the 5kW required in ISP 5 is feasible. In addition, it provides information on the load that has been diverted to other ISPs. The example shows that 3kW has been diverted to ISP 6, as the previous FlexRequest indicated that there were 3 kW at ISP 6 available for use.

As for the FlexOrder message, the content is the same as for the FlexOffer, as it is a confirmation/acceptance of the flexibility offer.

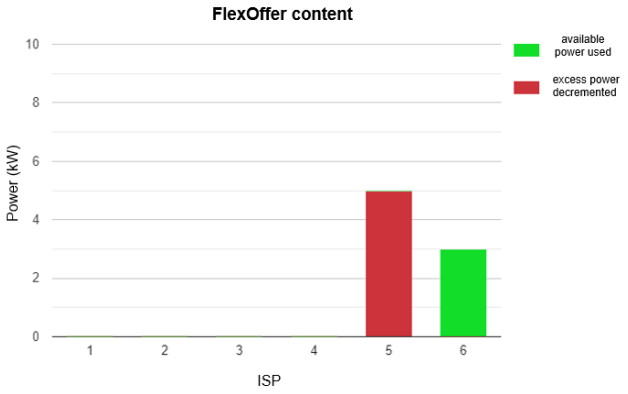

Figure 4 FlexOffer content

2.3 Grid communication development activities

Externally to PHOENIX, the aggregator uses the USEF framework to drive towards the smart energy future. This framework offers a common standard on which to build energy smarts services and products in the energy domain, as well as tools that allow the establishment of guidelines for the harmonisation and development of distributed flexibility mechanisms, benefiting all stakeholders in the new energy market.

As detailed in the previous section, the communication between aggregator and DSO is performed externally to PHOENIX. On the other hand, PHOENIX acts as a support tool for the aggregator and must allow the integration of the aggregator into the platform. The communication between the aggregator and the grid is done through the set of protocols and tools proposed by USEF, therefore, PHOENIX must fulfil the standard messaging format between the different actors of the smart grid.

In order to increase interoperability between PHOENIX and the different grid actors, an interpreter for USEF has been developed. As indicated in the grid communication requirements defined in D2.3, PHOENIX must support the standard format of the grid agents (such as Aggregator) to facilitate their integration into the platform and allow access to the services offered. This is solved by developing a USEF interpreter that translates the USEF language (XML) into an appropriate language in PHOENIX (NGSI-LD).

Figure 5 shows the integration of the interpreter developed in the PHOENIX platform and its connection with the flexibility engine.

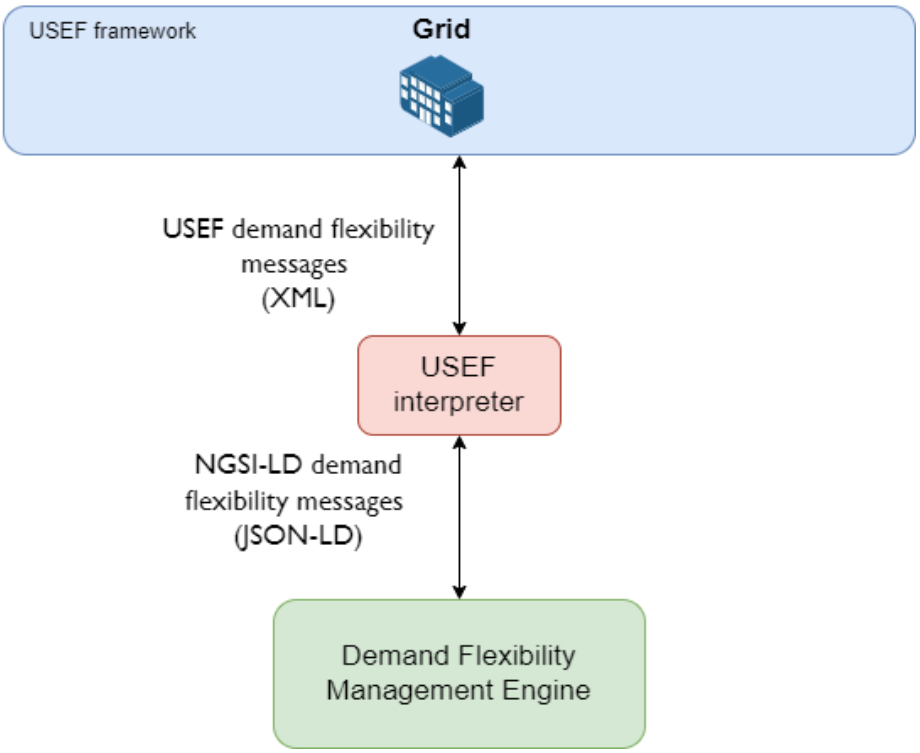

Figure 5 USEF Interpreter Integration

The main messages involved in the flexibility trading process are FlexRequest, FlexOffer and FlexOrder. Therefore, the first version of the USEF interpreter translates this set of messages for flexibility trading and operation, promoting the interoperability offered in PHOENIX, being compatible and allowing the integration in PHOENIX of grid agents established in an external framework such as USEF.

XML is the language proposed by USEF for the serialisation of these messages defined by the UFTP protocol. The USEF interpreter provides a REST API that allows messages in XML to be received from the grid (such as the aggregator) and translated into JSON-LD, a compliant format for the NGSI-LD API used throughout the PHOENIX framework. In the opposite case, it also allows the grid aggregator to get the PHOENIX flexibility messages in USEF (XML) compliant format following the reverse order in the translation.

Figure 6 shows an example of the translation of a FlexRequest message.

Figure 6 FlexRequest Translation

On the other hand, an example of the reverse translation of a FlexOffer message is shown in Figure 7.

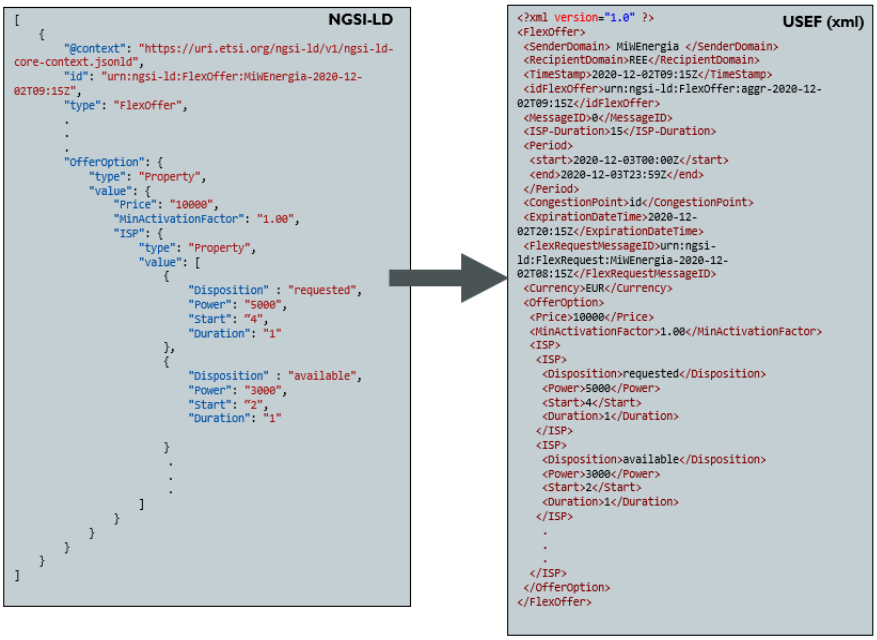

Figure 7 FlexOffer Translation

2.4 Smart contracts integration

So far, the communication with the grid by the PHOENIX platform focuses on the process of negotiation and realisation of flexibility by interacting with the involved grid agents, such as the aggregator. In contrast, PHOENIX offers services that go beyond the negotiation and operation of flexibility. The PHOENIX platform is able to learn about the behaviour of the grid and its needs, because a log on the flexibility engine records historic congestion signals and flexibility demands, this will allow algorithms to be trained so they can anticipate to grid congestion points and adapting the behaviours and consumption patterns of end users.

The constant learning of grid and end-user behaviour can have a direct impact on the rate plans and contract billings established. Thanks to the development of the Flexibility Engine deployed in PHOENIX it is feasible to obtain the flexibility requirement patterns and to obtain the most frequent flexibility periods.

The components and mechanisms designed for the realisation of the flexibility services allow the historical storage of all processes and interactions performed between PHOENIX and the grid. Therefore, the design of a pattern search algorithm makes it possible to infer the behavioural pattern of the grid and to know in advance the possible flexibility requests to solve congestion. This pattern search algorithm has been considered to be implemented on the algorithm's engine as part of Task 4.4.

This analysis of the grid behaviour and the demand adjustment capacity of the building occupants for a flexibility anticipation is beneficial for both actors. On the one hand, it prevents possible congestion points in the grid and facilitates the task of the Aggregator. On the other hand, the load profile associated with the end-user is modified taking into account consumption and grid patterns thus allowing the proposal of new smart rate plans with anticipated flexibility.

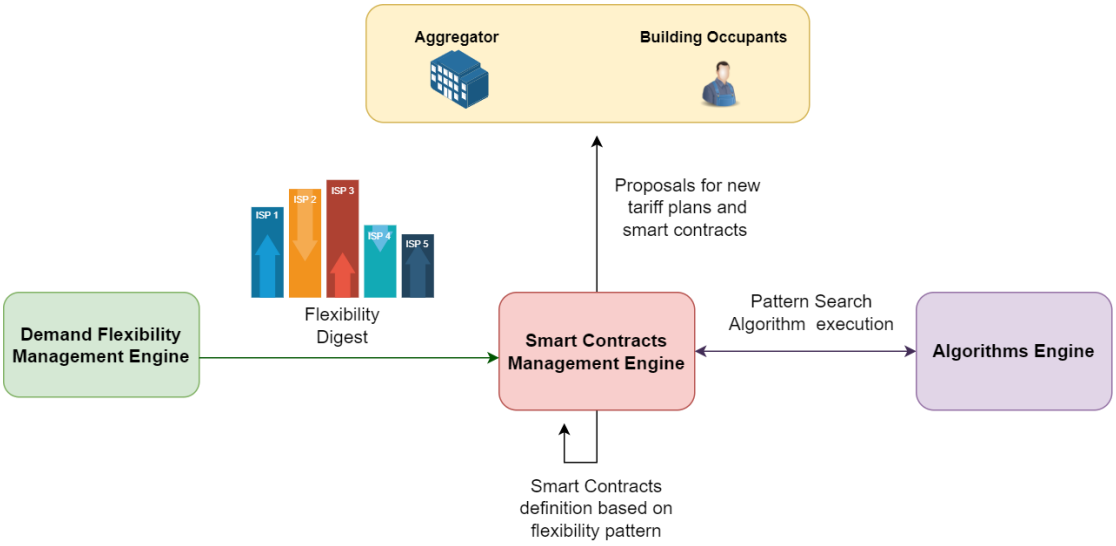

Figure 8 Smart Contracts design flow

As stated in D6.1, numerous requirements for grid flexibility emphasise the need for greater awareness of energy savings, energy efficiency and smart programs. This new paradigm in which the consumer is able to offer flexibility to the grid actors provides an economic incentive that helps the consumer to become aware of energy efficiency and smart programs. In addition, the proposal of personalised contracts and rate plans helps building occupants to learn more about their energy consumption and their behaviour with the grid (Figure 8).

3 Services for grid flexibility oriented to demand adjustment

3.1 Introduction

As reflected in T6.2, one of the scopes of PHOENIX is to offer a series of services to the grid that allow for the adjustment of the users' demand in order to achieve the flexibility required by the energy utility. To do this, the energy utility must first indicate the need for flexibility to solve the load at a specific congestion point. Therefore, PHOENIX must offer the necessary mechanisms to perform the interaction with the grid, negotiate the flexibility and evaluate the feasible and available demand adjustment in the ecosystem of controlled devices.

This document details the design of the software related to the Demand Flexibility Engine. In addition, a first version of the developed bundle is almost finalised for this document, providing the initial mechanisms to implement the flexibility negotiation and load demand adjustment functionality.

The following sections show the decisions taken in designing each of the mechanisms, the interaction with the different components involved for the service and the data analysis required for the design of the demand response events.

3.2 First version of Demand Flexibility Management Engine

The bundle is divided in different modules which provide scalability and modularity to the Demand Flexibility Management Engine. Figure 9 shows the Demand Flexibility Engine overview with the different modules. This is an evolution from preliminary versions of this part of the algorithm, after learning more about the situation of the state of the art/technologies.

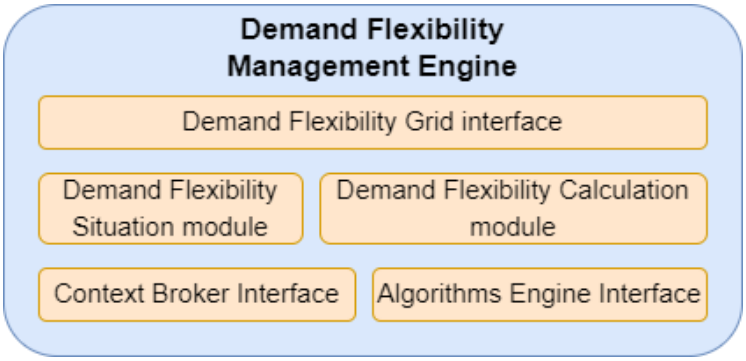

Figure 9 Demand Flexibility Engine Overview

Demand Flexibility Grid interface: This module is in charge of providing communication with the grid, offering a REST API that allows interacting with the Flexibility Engine to receive flexibility messages that will later be processed and performed. In particular, this module communicates the Flexibility Engine with the USEF interpreter developed for the integration of grid agents, such as aggregator, in PHOENIX. The different USEF messages coming from the grid through the USEF interpreter are arrived through the API endpoints offered by this module.

Demand Flexibility Situation module: The Flexibility Engine must be able to assess the devices it has deployed and their capabilities, being able to measure the feasibility of the required flexibility. There are many devices integrated in PHOENIX, but not all of them offer flexibility. This module is in charge of extracting all the information of the devices that are able to alter their operation pattern and estimate the amount of flexibility they are able to offer and evaluate if it fits with the request received from the grid. It makes use of the data stored in PHOENIX Real Time Data Broker (Orion-LD) and PHOENIX Platform Data Repository (Historic Data Base), including consumption data, flexibility configuration of the devices and other data sources. In addition, it also uses data analysis from the Algorithms Engine, such as next-day temperature forecasting, which is useful for estimating HVAC performance.

Demand Flexibility Calculation module: The role of this module is the design and execution of Demand Response Events to be performed in each of the devices involved in the flexibility process. Previously, the flexibility negotiation has been performed with the aggregator and the feasibility of the agreed flexibility has been evaluated. Once the flexibility has been agreed, this module designs the action in each of the devices with the aim of altering their behaviour (e.g. consumption reduction or sliding over time its period of operation) implying a reduction of the peak load in the period required by the grid. This functionality makes use of the actuation mechanism provided by PHOENIX through the Context Broker, controlling the operation of the devices at a low level.

Context Broker Interface: The connection to the Context Broker is essential for the realisation of services related to flexibility with the grid. In the Context Broker all PHOENIX context information including all measured data from the devices, such as energy consumption, temperature, active power, etc. are stored. In addition, the connection to the context broker is necessary to perform the command actuation on each of the devices involved in the Demand Response Events designed for flexibility. Therefore, this module is responsible for providing this communication with PHOENIX Real Time Data Broker (Orion-LD) and PHOENIX Platform Data Repository.

Algorithms Engine Interface: In order to assess the request for flexibility from the grid, data analysis is necessary to estimate the consumption load of the devices involved. Therefore, forecasting algorithms are necessary to obtain the load estimation in a specific period. In addition, some forecasting analysis from external data sources can be used to adjust the designed Demand Response Events. For example, the next day's temperature forecast to decide the HVAC setpoint to be modified. The communication mechanism with the Algorithms Engine is done through NGSI-LD subscriptions. Therefore, a module is required for design the subscription and data extraction functionalities.

The services for grid flexibility have been developed using the following tech stack

- Backend: Flask, Celery frameworks
- Databases: MongoDB
- Analytics: Python and its respective libraries such as Pandas, PyCaret, Sklearn.

The development of the Demand Flexibility Management Engine is in progress in this version of the document, pending the final steps of action and design of the DR. Therefore, the deployment is done locally during the development period.

For the production deployment, once the development is finished, Docker technology will be used for the creation of Docker images that will host the developed services and will be deployed as containers, as stated in D2.3. The details of the deployment phase will be detailed in the next version of the document.

3.3 Demand Flexibility Management Engine Integration

According to the demand adjustment service offered in PHOENIX, some components from PHOENIX architecture have been used and others have been developed, in order to perform the interaction with the grid, the design of the Demand Response Events and the execution of these events on the devices allowing flexibility. Figure 10 shows the infrastructure designed and developed to perform this grid-oriented service which offers flexibility to aggregators. It shows the components involved and the communication between them.

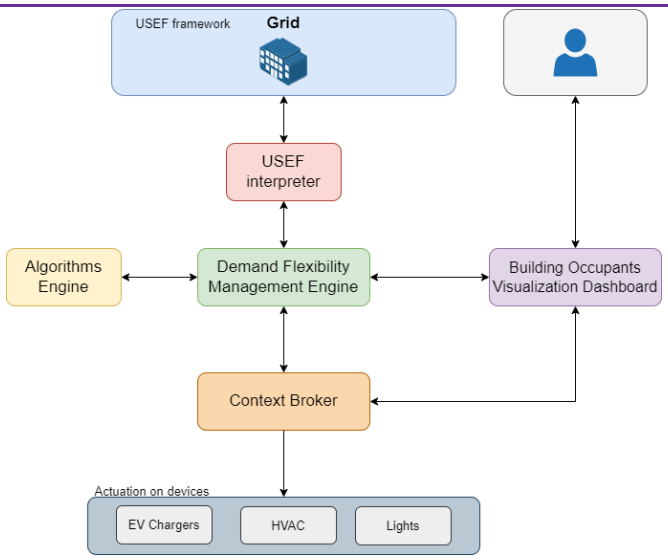

Figure 10 Demand Flexibility Management Engine Integration Diagram

Communication with the grid agents (as the aggregator) is performed by the USEF Interpreter component, which is responsible for translating the messages coming from the grid (defined in the format proposed by USEF, i.e. XML) into a PHOENIX compliant data language, compatible with the NGSI-LD API. Integration with this component is therefore essential.

On the other hand, the integration with the PHOENIX Real Time data broker (Context Broker) is indispensable to obtain all the context information of the platform and the deployed devices that will allow the adjustment of the building's demand. There are numerous data sources required by the flexibility engine that will be used for the evaluation process and the design of demand response events.

These are some of the data sources used in the first version of the engine:

- Device energy consumption.
- Device flexibility configuration.
- External temperature.
- Current configuration of HVAC devices.

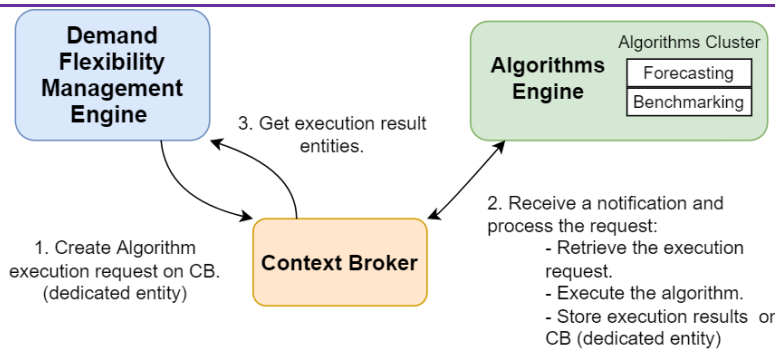

Figure 11 Demand Flexibility Management Engine and Algorithms Engine Communication

The demand adjustment through demand response events is performed thanks to the actuation mechanism provided by the Context Broker, acting on the devices at a low level. It is therefore clear that integration with this component is a fundamental step in the development of the flexibility engine.

In the same way, the engine requires full integration with the PHOENIX Platform Data Repository to obtain historical data, adding extra value to the information used to process flexibility decisions. In addition to the context information discussed above, data analytics are required by the Demand Flexibility Management Engine for accurate processing and evaluation of Demand Response Events. For this purpose, the integration with the Algorithms Engine allows obtaining these results in a scalable and transparent way, without involving the Flexibility Engine in complex algorithmic tasks. In this first version of the flexibility engine, the integration with the algorithm engine is not finalised as they are developed in parallel.

So, in these first steps, data analytics algorithms are executed locally. The interaction between the two components is done through the subscription mechanism offered by the Context Broker. Figure 11 shows how this works.

In PHOENIX, a large number of devices are integrated into the platform, providing a wealth of information from the measurements taken. On the other hand, in addition to providing measurements, typical of a sensor, some of them are also actuators, being able to control the behaviour of the device. These actuator devices are key to the demand adjustment performed by the flexibility engine. When a flexibility signal arrives from the grid, a direct load control strategy can be performed on the actuator devices that allows altering their behaviour (e.g. consumption reduction, sliding over time its period of operation) implying a reduction of the peak load in the period required by the grid. There are different types of devices, depending on their flexibility of use. Table 1 shows the types of devices in terms of flexibility.

Table 1 Type of devices for flexibility

Shiftable devices	Allow time-shift load. It is possible to deviate the power consumption from one time period to another, e.g. EV chargers
Non-shiftable devices	Do not provide flexibility as consumption cannot be controlled and time-shift load is not allowed.
Controllable devices	Allow load reduction. It is possible to control the device and perform a reduction of power consumption, e.g. HVAC.

To offer flexible services to the grid, only controllable and shiftable devices are required. When the Flexibility Engine receives a flexibility request, it evaluates these two types of devices and calculates the Flexibility Offers according to the devices' capabilities to alter their operating pattern. Therefore, the initial configuration of these devices is necessary to establish the flexibility characteristics of each device. These characteristics are:

- Flexibility Type: Type of device in terms of flexibility. Possible values: Controllable, Shiftable, Non-shiftable.
- Controllability Factor: Only for controllable devices. Controllability factor associated with the device. Integer between 0 and 1. For example: 0.20 means that it can be controlled to 20 percent of the total control. (HVAC whose setpoint can only be increased or decreased by 20 %).

- Operation Window: Only for shiftable devices. Time window available for the operation of the device. This allows sliding the operating period with the operation window as a limit, allowing time-shift load.

For the definition and configuration of these flexibility features, a dedicated entity has been defined called FlexConfiguration. Therefore, initial configuration flexibility should be the first step for the building to be able to offer flexibility by adjusting the demand side of its devices. This configuration task is performed by the building manager through the Building Occupant Dashboard, as it presents a dedicated panel where the building manager can set the device type and the flexibility features discussed above. As a result of this configuration, an entity of type FlexConfiguration is created for each device. With this, the initial FlexConfiguration entities are created for the appropriate devices with some initial values (this conceptually will correspond to the time 0 of the problem), then stored in the broker, and then the user by selecting options through the dashboard can perform updates to these entities. An example of this entity used for the first version of the software is shown in Figure 12 FlexConfiguration entity.


```
{
   "id": "urn:ngsi-ld:FlexConfiguration:UMU-Pleiades-BlockB-B1.1.015-Thermostat-FlexConfiguration",
   "type": "FlexConfiguration",
                                                "refDevice": {
      "type": "Relationship",
      "object": "urn:ngsi-ld:Device:UMU-Pleiades-BlockB-B1.1.015-Thermostat"
   },
   "flexibilityType": {
      "type": "Property",
      "value": [
         "controllable"
      ]
   },
   "controllabilityFactor": {
      "type": "Property",
      "value": 1
   },
   "operationWindow": {
      "type": "Property",
      "value": {
            "startTime": "",
            "endTime": ""
```


Figure 12 FlexConfiguration entity

Finally, it is important to note that this integration with the dashboard is done in an initial phase after the deployment of the devices or when updating the flexibility settings of some of them afterwards. On the other hand, the whole process of negotiation, agreement and execution of flexibility with the grid agent is performed through the Business Stakeholder API interface and USEF interpreter.

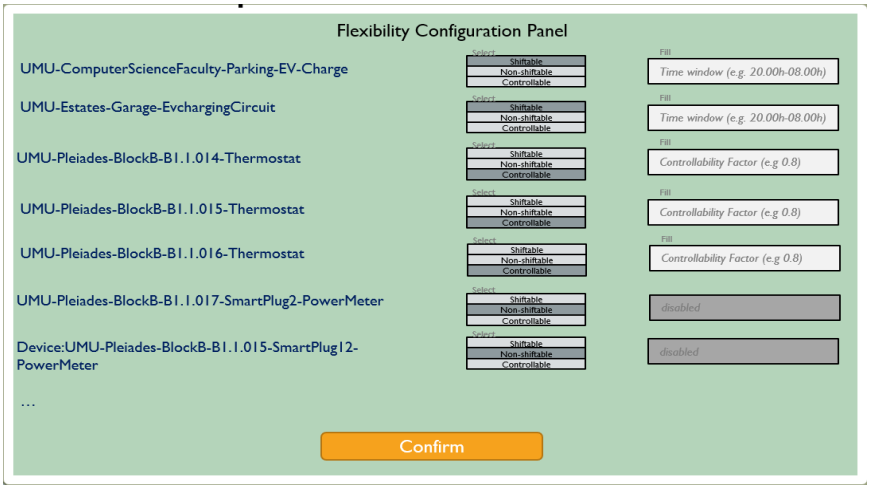

Figure 13 Demand Flexibility Configuration Panel Mock-up

According to the dedicated panel for this initial configuration phase, Figure 13 Demand Flexibility Configuration Panel Mock-up shows the mock-up designed in the first step of integration with the dashboard. Finally, Figure 14 Current Demand Flexibility Configuration Panel shows the current state of the configuration panel, which is still under development:

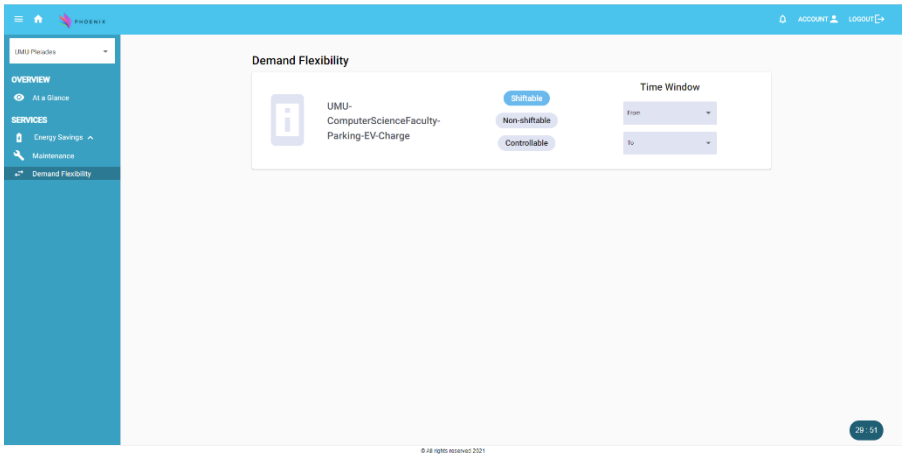

Figure 14 Current Demand Flexibility Configuration Panel

3.4 Demand Flexibility Management Engine Workflow

The flexibility negotiation with the grid is mainly based on three messages offered by USEF. These messages are as follows:

- FlexRequest: Flexibility Request. It is sent by the aggregator to the PHOENIX platform and indicates the magnitude and timing (ISP) of the requested flexibility. That is, it indicates the amount of energy to reduce and the ISPs available to perform time-shift load.
- FlexOffer: Flexibility Offer. This is a response to the FlexRequest. It contains the load profile change that can be made to perform the flexibility.
- FlexOrder: Flexibility Order. After receiving the different FlexOffers, the aggregator chooses the most suitable offer and sends the FlexOrder. FlexOrder has the same content as the previous FlexOffer, to confirm the agreed flexibility. Once received by PHOENIX, the flexibility must be performed.

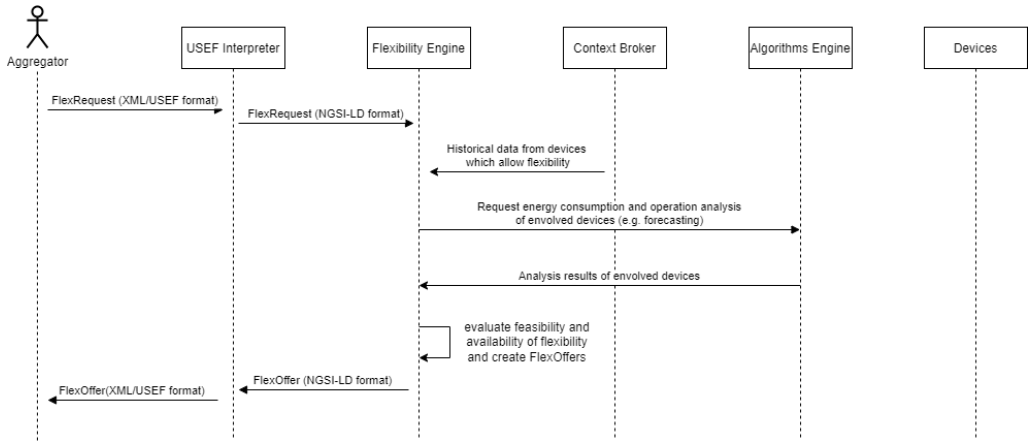

Figure 15 Flexibility Negotiation Workflow

These three messages guide the operation of the Flexibility Engine, including the negotiation phase with the grid aggregator and the subsequent execution of Demand Response Events.

Figure 15 Flexibility Negotiation Workflow shows the flexibility negotiation steps between the aggregator and PHOENIX.

First, the aggregator makes use of the USEF interpreter to access the PHOENIX platform and therefore a compliant data language, i.e., the one offered by NGSI-LD, is necessary. Once the flexibility engine has received the flexibility request, it must examine in the Context Broker all the devices that may be involved in the demand response event to solve that flexibility. The devices that can be involved in a Demand Response event are shiftable devices and controllable devices.

After obtaining the devices that can be controlled or time-shift loaded, it is necessary to estimate what the load profile of each device will be in the period to which the flexibility request refers. So far, the data analytics involved in the design of demand response events has been forecasting algorithms. For this purpose, the connection to the algorithm engine is required. This engine is in charge of the analysis of the data coming from the devices involved in the flexibility.

When all the required information is collected, the Flexibility Engine evaluates the feasibility of the flexibility request and calculates whether it is possible to decrease the load on the indicated ISPs. To do so, it first evaluates the controllable devices and estimates the amount of energy that can be decreased taking into account their flexibility characteristics, such as the controllability factor. In a second step, it evaluates the shiftable devices and tries to shift the operation pattern to other ISPs with available power (indicated in the FlexRequest), thus releasing the ISPs where demand adjustment is required. With this estimation, flexibility offers are proposed to the aggregator by means of the FlexOffer messages.

So far, there has been a negotiation between PHOENIX and the aggregator in which a decrease of the load on specific ISPs has been required and PHOENIX has offered flexibility offers with the demand adjustment profile that it is able to reach. Once the aggregator has the flexibility offers, he has to take the decision and choses the one that best suits his needs.

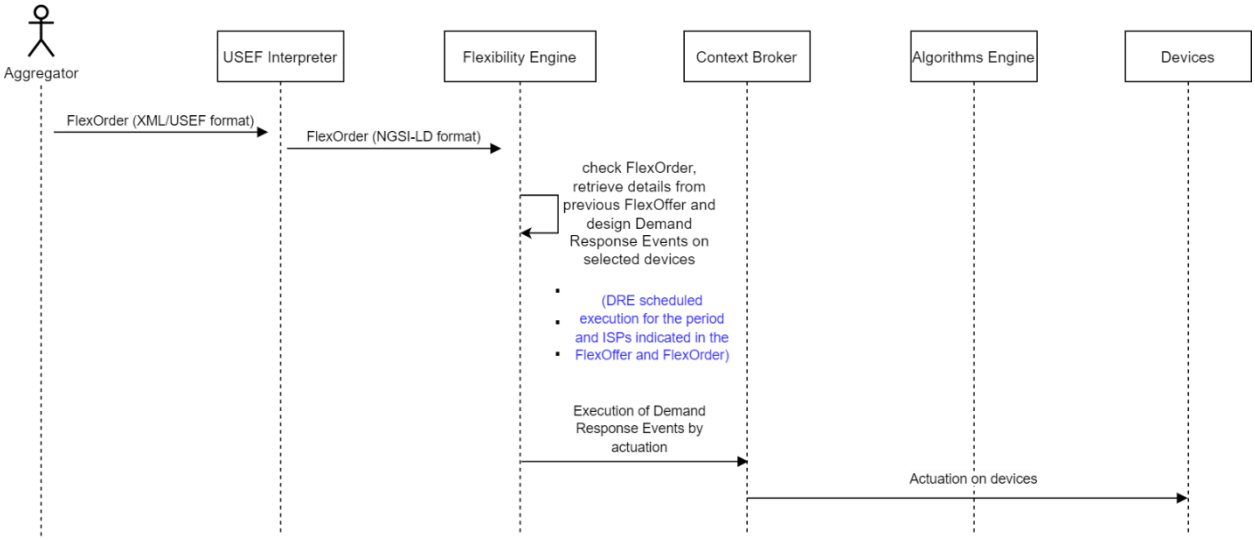

Figure 16 FlexOrder: flexibility execution workflow

Once the aggregator has chosen the appropriate flexibility offer, it proceeds to command the agreed flexibility via the FlexOrder message. In our case, as the aggregator is out of the scope of the project, we will simulate this demands, so we can test this functionality. The content of this message is the same as the content of the associated FlexOffer, as it is a confirmation and acceptance of the offer sent by PHOENIX. Therefore, this message contains the ISPs with the amount of energy to be decreased or shifted over time, taking into account the demand adjustment capability of the building devices integrated in PHOENIX. Figure 16 shows the workflow of this flexibility ordering and execution process.

4 Services for self-generation and energy storage

4.1 Introduction

The promotion of self-consumption of PV electricity is based on the idea that PV electricity will be used first for local consumption in order to reduce the buying of electricity from other producers (at least in the simplest form of a self-consumption scheme). In practice, self-consumption ratios can vary from a few percent to a theoretical maximum of 100%, depending on the PV system size and the local load profile.

There exist many policies allowing for self-consumption that are being implemented worldwide, but mainly we can divide them into two main categories: Onsite Self-Consumption and Excess PV Electricity [3]. In the context of the PHOENIX platform, we examine the first category and more specifically the following schemes:

- Right to self-consume.
- Revenues for self-consumed PV electricity, where savings on the variable price of electricity from the grid can be accomplished.

In other words, the prosumers of the demonstrators that participate in the Phoenix platform have the right to connect a PV system to the grid and self-consume a part of its generated electricity. And if there is also a battery available connected to the PV, then storage of part of the generated electricity is also permitted. Regarding the revenues from the prosumer side, if the self-consumption is arranged at times where the price of the energy coming from the grid is more expensive, this directly leads to cost savings in the energy bills. From the perspective of the ESCOs or energy utilities, additional energy efficiency strategies can be designed to promote self-consumption using bonus/premium schemes or green certificates.

PHOENIX will examine various self-consumption optimization scenarios according to the business interest of each demonstrator, but in this current version of the document work is focused on the PV generation and EV charging patterns of the UMU pilot. The purpose is to schedule EV charging in the timeslots of maximum PV generation for the UMU pilot. Another objective function in this case could be the usage of electricity from the grid at the cheapest times according to Spain’s energy tariffs. So, to sum up, the optimization algorithm looks ahead 24 hours and selects the best times to charge.

As a further integration step, if there is an additional percentage of generated energy, it will be used as an input to the flexibility engine as described in section 3. In that section, the case of positive flexibility is examined, where due to a foreseen congestion grid event, there is a simulated request coming from the grid to increase the amount of energy by either ramping up a generation or reducing energy usage. Here, in the case of excess PV generation (and since we are not examining any excess PV electricity self-consumption scheme) we will use the amount as a negative flexibility request towards the flexibility engine, in order to increase energy consumption by scheduling appropriately one or more flexible loads without compromising user comfort in the premises.

4.2 Optimization problem

The optimization process that we have followed is the standard methodology as described in the PuLP Python library, used for solving a Linear Programming Problem (LPP):

- Identification of the Decision Variables: Self-consumption ratio, EV optimized charging/discharging schedule.
- Formulation of the Objective Function: Maximization of self-consumption ratio, minimization of cost in the energy bill. In the PV-EV scenario this is done by maximizing the energy that the EV battery is receiving by the PV panel while minimizing the energy received by the grid, thus minimizing the costs.
- Formulation of the Constraints and assumptions: In general, the EV charging volume is not a straightforward process and can be influenced by many factors such as the condition and type of the vehicle, weather, the battery's SOC, daily driving patterns and the distance to the next destination. Unfortunately, most of the times, such crucial information about the vehicle is almost never available so either some assumptions must be made, or some implicit techniques must be used for their estimation. The EVs under consideration here are used for distances 20-30 km per day with an average consumption of 100-150 Wh/km. In fact, the Technical Unit Area of the University has four electric vehicles for short trips within the Espinardo Campus which is a 1.7-mile (4,000-step) route. In addition, restrictions apply concerning the max and min capacity of the EV battery and also, we want to make sure that there is enough energy for the EV's next planned trip.
- Identification of data needed for the Objective Function and Constraints:
    - Day-ahead forecasted energy produced
    - Day-ahead forecasted energy consumed
    - Initial EV battery energy available and battery specifications

- Trip patterns
- Day-ahead grid prices

4.3 Technical implementation and deployment of the solution

The initial design of the self-consumption optimization engine was delivered in D2.3. Figure 16 below demonstrates the interactions of the engine with the interfaces of all the other components of the PHOENIX platform that participate in the optimization process :

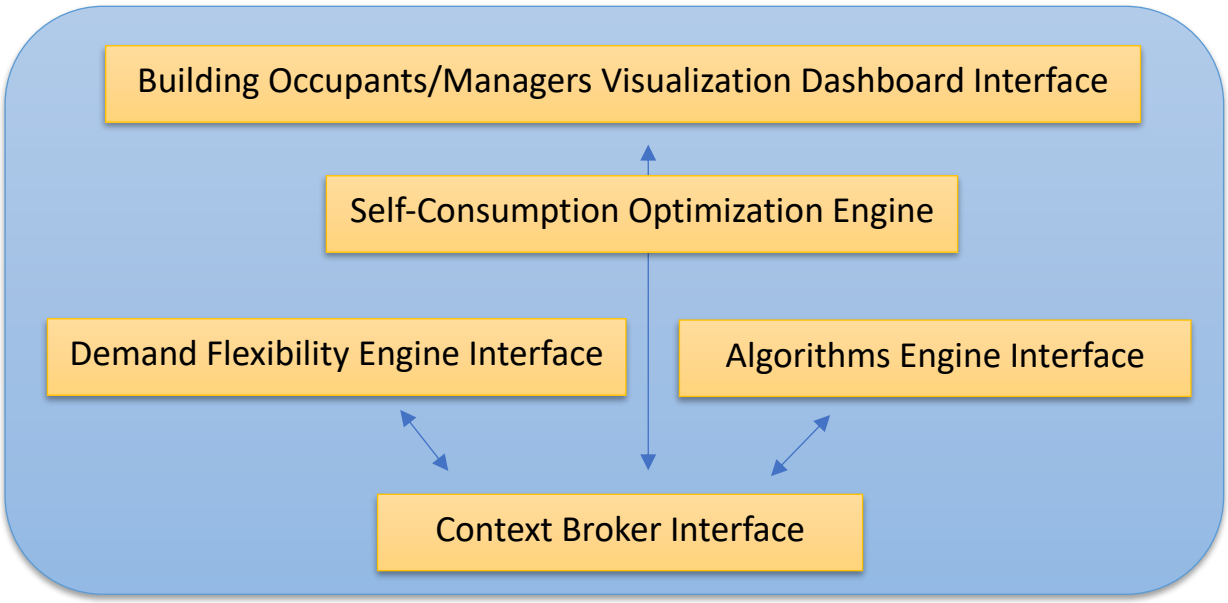

Figure 17 Interactions of self-consumption optimization engine with other components

The flow regarding the scenario of PV-EV can be summarized as follows:

As a first step and because the subscription mechanism towards specific topics of the Context Broker is not fully productionized yet, a scheduled cron job is triggered in the background in the Dashboard Sever backend. The time of the triggering is chosen to be at 21:00 UTC (this can change in the future).

The self-consumption optimization engine will ask from the Context Broker specific input regarding the day-ahead forecasted generation values and the day-ahead forecasted consumption values. If the results are not yet available in the CB, then a request is made towards the Algorithms Engine to start the execution of the respective forecasting algorithms. Any other static values or parameters based on assumptions needed for the solver will be given as hard-coded constants for the sake of simplicity and in the final version of the platform, they will also be available in the form of entities in the CB.

In case an amount of generated energy cannot be used to charge the EV, a request is made to the Demand Flexibility Engine, so that Phoenix can move the scheduling of some flexible loads like HVAC, lights etc. to specified timeslots. In the latter case additional energy cost savings can be calculated after the successful execution of this flow.

The services for self-generation and energy storage are delivered as a web-based application with the following tech stack:

- Backend: Django framework
- Frontend: Angular framework
- Databases: Postgres, Elasticsearch
- Analytics: Python and respective libraries like Pandas, Sklearn, Pulp
- Deployment: Kubernetes cluster

Each new code commit can practically represent a new deployment. The gitlab CI/CD pipeline is triggered and new docker images are created and stored in Ubitech's private gitlab registry. The frontend and/or backend Kubernetes deployment services are restarted so as the new images to be pulled and run in the Kubernetes cluster. In principle the functionality of the services for self-generation and energy storage is packaged in the backend Kubernetes service whereas the visualization aspects of the results and the energy charts/KPIs are packaged in the frontend Kubernetes service.

4.4 Visualization of Results in the Energy Stakeholders Dashboard

The most important information that need to be visualized in the Dashboard for the UMU pilot regard energy generation, energy demand, EV charging/discharging schedules, self-consumption ratio results and cost savings results.

Regarding the energy consumption, and with a pre-selected building, the user can click on the "energy savings" tab and choose the option "energy consumption". Figure 18 shows an example regarding UMU-Pleiades building's energy consumption in kWh from 06/03/2022 – 12/-03/2022. The values range from 300-1200kWh. From the left dropdown "Per Day" the user will have the option to choose also "Per hour" depicting energy consumption of the last 24h and of course this can be extended to per Week and per Month options. We should be noting here that because these kinds of dynamic graphs are user-defined (click based) their reconstruction takes some seconds. This has been spotted as a performance issue and will be optimized in the final version of the dashboard.

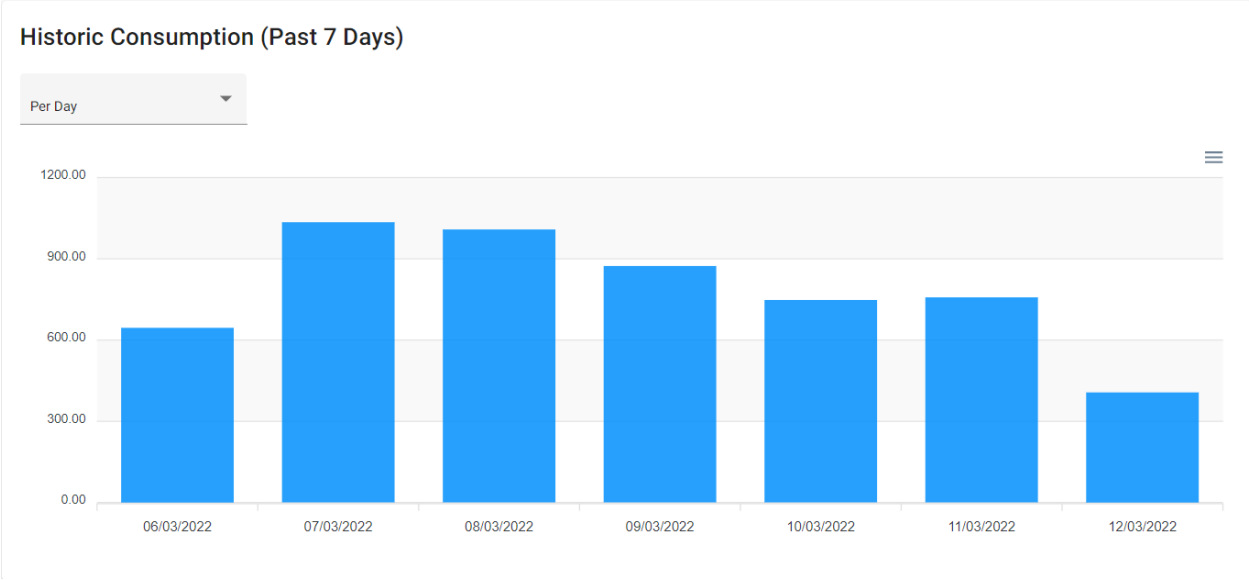

Figure 18 Historic Energy consumption of UMU-Pleiades

Figure 19 below shows the corresponding building energy generation in kWh when the user chooses "energy generation" in the "energy savings" button, from 16-22 of March. The values range from 50-320 kWh for the respective week.

Another interesting feature of these kind of graphs is a small dropdown menu in the upper right of the graph where the user has the option of downloading the x-y axis values in CSV format, downloading the graph as a .png image or open it in another page .svg format.

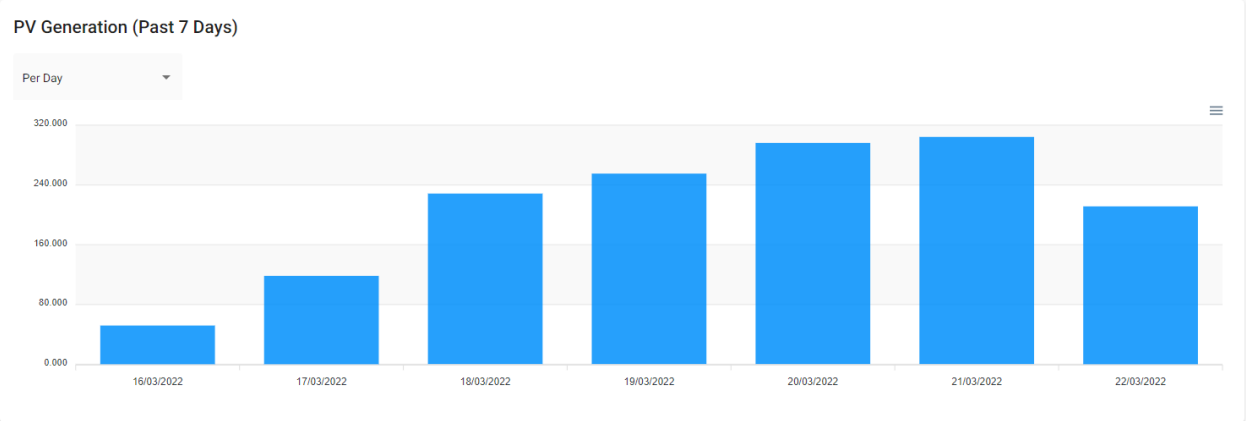

Figure 19 Historic Energy generation from the PV installed in UMU-Pool-Parking

Finally, Figure 20 demonstrates an initial analysis on the charging patterns of the EVs charger of the buggies of the UMU pilot that are now available in the UMU demo site. Since there is no straightforward way to detect how many cars are charging at the same time, we will make the assumption that the lower peak corresponds to 1 vehicle and the large peak corresponds to 3 vehicles. Values range from 1-5kW. Our solver will suggest a day-ahead schedule of charging after also analysing past charging data.

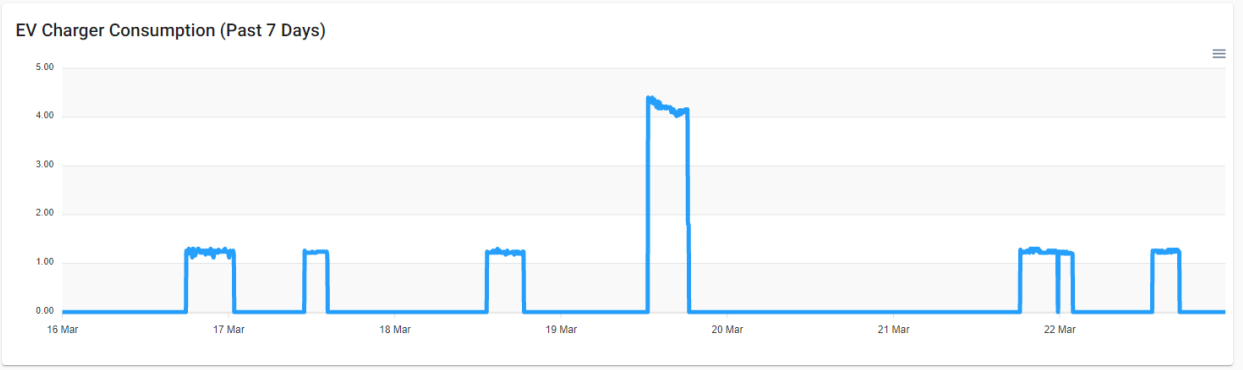

Figure 20 Charging patterns of EVs in UMU

5 Integration, testing and refinement

5.1 Integration

In terms of integration, every service must provide an interface so that other services, devices, users or even the testing infrastructure itself can interact with them.

Whenever it is possible, it is recommended to provide an API so that service tests can be automated. In this sense, the preferred API type is REST as it is the most platform-independent solution and will be the only one supported at this point of the project. Support for other types of APIs might be added in the future depending on the requirements of the service developers.

5.2 Testing

As indicated in D5.2, different services can have different requirements that must be provided by the testing infrastructure and therefore the goal at this point of the project of both T5.4 and T6.4 is to offer an initial set of alternatives so that service developers can choose between different ways to define their own tests. Every testing methodology can be used by services from both WP5 and WP6 so that is something to be taken into account by WP5 service developers and not only by WP6 service developers.

As stated before, in order to define a test, the service to be tested must provide an API. The first thing to do is to fill a test template using the format defined for this purpose.

The template is divided in two sections:

- A global description of the test.
- A definition of each one of the stages that will be executed.

5.2.1 Description of a test

The description of a test includes the next information:

- Information about the service to be tested (work package's number and title, service name, etc.).
- General information of the test itself (description, periodicity, number of stages, etc.).

The next table shows an example of the description of a test using the defined template.

Table 2 Definition of a test

Work Package	6	**Title**	Synergic grid interaction and automatic demand response			
Service name		Demand/Response in UMU Pleiades building (send demand request)				
Test	**Description**	The service will send the configured demand requests				
	ID	T6.2-S1.T1A	**Periodicity**	Twice a day, first at 09:00:00 and second at 16:00:00 both in local time	**Number of stages**	1

Every test will have a unique ID that will be used in the future for creating a combined list of tests grouped by WP and service, and will include some additional high-level information such as the category of the test, priority, criticality level, etc.

5.2.2 Definition of a stage

Each stage of a test will have its own separate definition. The template defined at this point of the project is designed to be used on services using REST APIs, and includes all the information required to execute the specific task:

- Description.
- Endpoints required for sending the queries.
- Format of the request (method, parameters, headers, payload, etc.).
- Example of a request.
- Format of a response.
- Examples for valid and invalid responses.
- Actions to be executed depending on the result of the execution, namely valid response, error response or timeout.

The next table shows an example of the definition of a stage of a test following the template format.

Table 3 Definition of a stage

Test ID	T6.2-S1.T1A	**Stage**	1
Description	Send the demand request to be executed during the next 2 hours.		
Type of interface	REST		
Endpoint	https://phoenix.odins.es:1031/services/demand-response/request		
Method	POST		
Parameters	None		
Headers	• Content-Type: application/json • Accept: */*		
Payload	A json array including 8 values indicating the requested demands for each 15-minute ISP starting at the time the request is sent. Knowing that the service will look for the closest ISP as start point to feed the flexibility engine, the first value will be used for the (09:00:00,09:15:00] interval, the second will be used for the (09:15:00,09:30:00] interval and so on. Each value of the array can be: • Negative if there is a peak-available offer for the ISP. • 0 if there is no request. • Positive if there is a peak-reduce request for the ISP.		
Example of query	https://phoenix.odins.es:1031/services/demand-response/request [0, 0, 0.5, -1.3, ….]		
Format of response	A valid response will return a json object in the payload with a *status* attribute indicating the result of the request (valid values are *Accepted*, *Denied* and *Engine Busy*).		

Example of valid response	HTTP/1.1 200 OK
	Content-Type: application/json
	{
	“status”:”Accepted”
	}		

Actions on valid response	1. Send an email to the next destinations: - admin.services@phoenix.org - demand-response.services@phoenix.org including: - Subject → Demand/Request / request status $status - $TIME - Payload → Demand: [0, 0, 0.5, -1.3, ….] 2. Send a POST request to the next destination: https://phoenix.odins.es:1031/services/demand-response/log including as payload: Demand/Request: status $status - $TIME - [0, 0, 0.5, -1.3, ….]
Example of error response	HTTP/1.1 400
Actions on error response	1. Send an email to the next destinations: - admin.services@phoenix.org - demand-response.services@phoenix.org including: - Subject → Demand/Request / error - $TIME - Payload → Error: $statusCode (400 in this case)
Actions on timeout	1. Send an email to the next destinations: - admin.services@phoenix.org - demand-response.services@phoenix.org including: - Subject → Demand/Request / timeout - $TIME

6 Conclusions

This document presented the refinements in the energy services for energy utilities and the grid done under the context of WP6. Three basic engines (components of the PHOENIX platform) have been presented, namely Smart Contracts Management Engine (Section 2), Demand Flexibility Management Engine (section 3) and the Self-Consumption Optimization Engine (section 4).

More specifically, regarding the progress on the smart contracts mechanism and taking always into account the features per component as described in the architecture deliverable D2.3, the main work done was around the implementation of a USEF compatible module for transactions communication. Regarding flexibility, significant work was presented around the identification of potential automatic actuations that result on Demand Flexibility as well as the flexibility triggering system itself.

Finally, on the aspect of self-consumption the reporting of relevant information to ESCOs/Utilities was demonstrated, which of course will remain a recurring task until all services share the content needed for the visualization dashboard. The feature of day-ahead optimizations for energy generation, storage and demand is still ongoing at the time of writing this deliverable and the next phases will be the delivery of an optimal day-ahead EV schedule as well as an optimal day-ahead battery schedule.

API Testing of the services can be completely automated using Jenkins framework (example can be found in the Annex I), without excluding of course the use of other popular testing frameworks in the project.

7 References

1. USEF Foundation, USEF The Framework Explained," 20 06 2021. [Online]. Available: www.usef.energy.
2. USEF Foundation, USEF Flexibility Trading Protocol Specifications," 28 01 2020. [Online]. Available: www.usef.energy.
3. REVIEW AND ANALYSIS OF PV SELF-CONSUMPTION POLICIES, Report IEA-PVPS T1-28:2016

8 Annex I - Testing with Jenkins

One of the tools studied for testing services is Jenkins. Jenkins is an open-source server for testing and/or continuous integration. It is a cross-platform tool written in Java, accessible through a web interface.

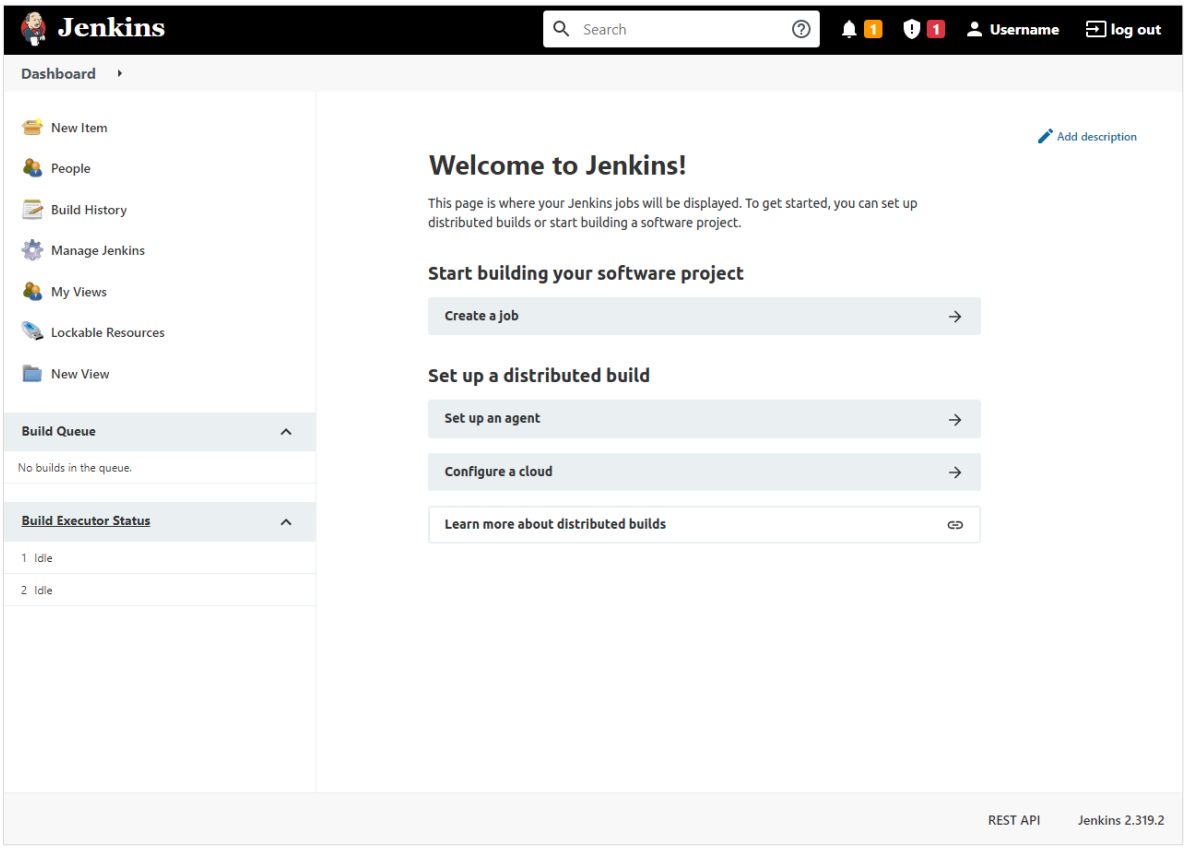

Figure 21 Jenkins frontend

By default Jenkins offers a set of built-in utilities which offer certain functionality, but can also be extended by installing additional plug-ins including improvements in the UI, administration tools, integration with Kubernetes, ssh, GitHub, etc.

8.1 Main functionality

Out of the many options provided by Jenkins, there are a few of them especially relevant as they are enough to implement fully operational tests:

- Job. A job is a task that can be executed either periodically or triggered by an event, such as the end of another job.
- Pipeline. A pipeline is a type of job that's formed by a set of ordered stages.

- Stage. A stage is a subset of the tasks of a pipeline. The criteria used for dividing a pipeline in stages depend on the characteristics of the global task. In the end it's just a logical division.
- Step. Each stage is divided as well in one or more steps. A step is the smallest unit that has its own integrity in a pipeline and its purpose is to perform a very specific operation. Groovy is the language in which the different steps of a pipeline are coded. It could be defined as a sort of simplified and modified version of Java. However additional libraries can be installed for using other languages when coding steps such as python, perl, etc.

Using the frontend, jobs can be created using a wizard.

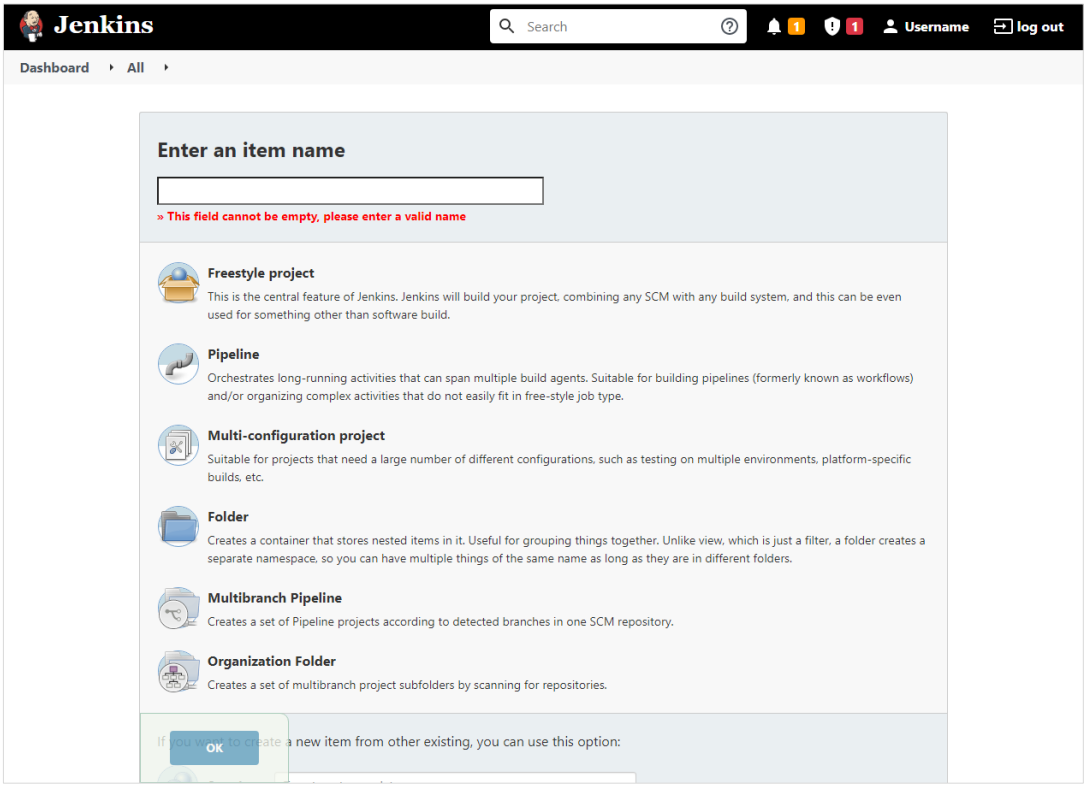

Figure 22 Creating a job

During the job creation, the two most important items that must be configured are the scheduling and the code of the pipeline itself.

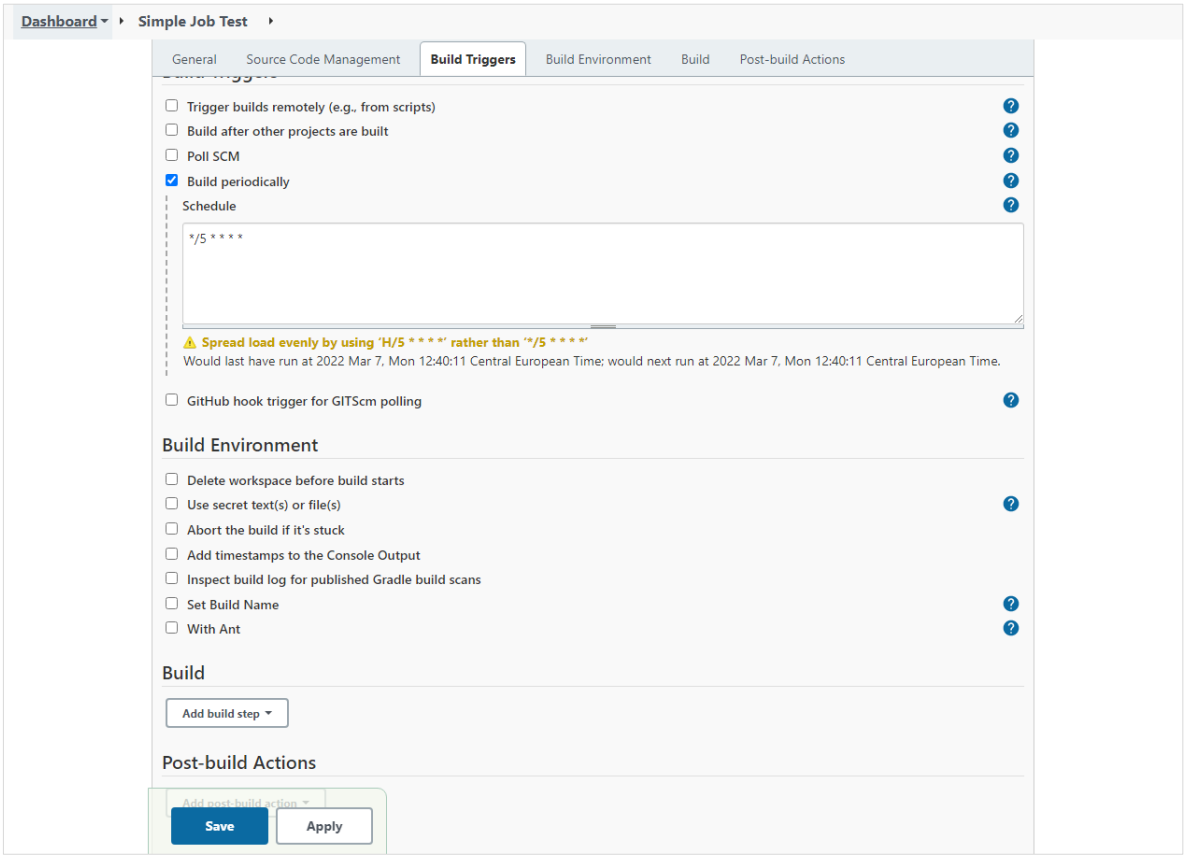

Figure 23 Configuration of a job

For programming the execution of the test, assuming it is going to be launched periodically, Jenkins uses a syntax based on the one used by cron but slightly extended.

The code of the pipeline is configured in the pipeline section and can be defined by either adding steps one after another of by pasting directly the full content of a Jenkinsfile as introduced in the next section.

8.2 Jenkinsfile

A Jenkinsfile includes all the code of a pipeline. It provides access to environment variables, definition of stages and steps, etc.

```
pipeline {
    agent any

    stages {
        stage('Service1') {
            environment {
                status1 = """${sh(
                        returnStdout: true,
                        script: 'curl -i --location --request GET \'http://IP1:PORT1/\' | grep \"HTTP/1.\"
| cut -d \' \' -f2'
                    ).trim()}"""
            }
            steps {
                script {
                    if (status1 == "200") {
                        echo "Service 1 Status: ${status1}"
                        emailext body: """Service 1 Status: ${status1}""", subject: "Simple Pipeline Test
- Service 1 Status", to: "email@mail.com"
                    } else {
                        error "Error status1: ${status1}"
                    }
                }
            }
        }
        stage('Service2') {
            environment {
                status2 = """${sh(
                        returnStdout: true,
                        script: 'curl -i --location --request GET \'http://IP2:PORT2/\' | grep \"HTTP/1.\"
| cut -d \' \' -f2'
                    ).trim()}"""
            }
            steps {
                script {
                    if (status2 == "200") {
                        echo "Service 2 Status: ${status2}"
                        emailext body: """Service 2 Status: ${status2}""", subject: "Simple Pipeline Test
- Service 2 Status", to: "email@mail.com"
                    } else {
                        error "Error status2: ${status2}"
                    }
                }
            }
        }
    }
}
```

As can be seen in the previous sample, direct calls to scripts or command line utilities such as *curl* are supported and, if the scripts are called in the **environment** section, the return value can be used to trigger actions (send emails, launch a script that could update an entity in the PHOENIX Real-Time Data Broker, etc.) or detect an **error** (that will be logged) and abort the execution of the test.

8.3 Test implementation

Tests will be implemented by the developers of the services themselves using the information included in the *Test implementation guide for Jenkins*. This guide is under development and will be offered to tests' implementers so that they can use it as a reference.

Later, additional tests can be implemented by third-party developers in order to offer additional testing support so that this second validation level can strengthen the testing phase.

8.4 Test deployment

There will be at least one Jenkins server for deploying all the tests that use it. During the testing phase, all the participants will work in a coordinate way to monitor, debug and trace the execution of the tests and to validate the proper operation of the testing infrastructure itself.