MARKET ENABLING INTERFACE TO UNLOCK FLEXIBILITY SOLUTIONS FOR COST-EFFECTIVE MANAGEMENT OF SMARTER DISTRIBUTION GRIDS

Deliverable 7.3
PT - Validation and results of UMEI concept test, data collection and Demo results assessment report

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 864334

H2020 – LC-ES-1-2019

Document	
D7.3	PT - Validation and results of UMEI concept test, data collection and Demo results assessment report

Dissemination level	
PU	Public

Author(s)	Institution	Contact (*e-mail, phone*)
Carlos Damas Silva	E-REDES	carlos.damassilva@e-redes.pt
Pedro Marques	E-REDES	carlospedro.marques@e-redes.pt
Gesa Milzer	NODES AS	gesa.milzer@nodesmarket.com
Gil Sampaio	INESC TEC	gil.s.sampaio@inesctec.pt
Louise Adam	N-SIDE	lad@n-side.com
Pierre Crucifix	N-SIDE	pcu@n-side.com
Clement Gohlke	CENTRICA	clement.gohlke@centrica.com

Key word	Demonstration
Due Delivery Date	2023/11/30
Date of Delivery	2023/12/29

Document version	Date	Change
0.0	2023/08/30	1st draft
1.0	2023/12/22	Version for revision
2.0	2023/12/29	Final version after revision

Reviewers		email	Validation date
MITNETZ	David Brummund	david.brummund@mitnetz-strom.de	2023/12/28
ENERGA	Miroslaw Matusewicz	miroslaw.matusewicz@energa-operator.pl	2023/12/27

Table of Contents

List of Abbreviations

UMEI	Universal Market Enabling Interface
API	Application Programming Interface
DER	Distributed Energy Resources
DG	Distributed Generation
DSO	Distribution System Operator
FSP	Flexibility Service Provider
BRP	Balancing Responsible Party
RES	Renewable Energy Sources
LV	Low Voltage
MV	Medium Voltage
FMO	Flexibility Market Operator
CRUD	Create, Read, Update, Delete
TSO	Transmission System Operator
GDPR	General Data Protection Regulation
ICT	Information and Communication Technology
CIM	Common Information Model
IEC	International Electrotechnical Commission
IEGSA	Interoperable European Grid Services Architecture
IT	Information Technology
GUI	Graphical User Interface
SRA	Scalability and Replicability Analysis
UC	Use Case

Executive Summary

Under the European H2020 program, the EUniversal Project has the main objective to foster the universal access of system operators to the available flexibility, mainly provided by Distributed Energy Resources (DER), through the interaction with new Flexibility Markets and innovative services. With the development of solutions and services that allow the massive integration of Distributed Generation (DG), energy storage, and the active participation of consumers, the project aims to tailor the concept of the Universal Market Enabling Interface (UMEI). The UMEI will look to overcome the limitations that Distribution System Operators (DSOs), experience in the use of flexibilities, addressing the interlinking of electricity markets with active system management.

The EUniversal project aims to develop a universal approach to the use of flexibility by DSO and their interaction with the new flexibility markets, enabled through the development of the UMEI. The UMEI has materialized in the conceptual architecture design and the implementation of a standard, agnostic, adaptable, and modular combination of different APIs to link DSOs and market parties with flexibility market platforms, in coordination with other flexibility users. This approach allows distributed communication without the need for a central hub.

The UMEI consists of publicly available APIs, allowing any stakeholder to adopt them or to develop new APIs concerning new services while complying with the UMEI interface specification.

The aim of this deliverable is to describe the implementation of the UMEI and its usage in the demonstration held in Portugal.

1. Introduction

1.1 Background

The electrical system historically relied upon a set of "implicit services", provided by classical generation plants. Assuming the future scenarios and the upcoming perspectives, the availability of resources that provide these classical types of services will be significantly reduced. The current outlook of electrical systems shows a growing trend towards the incorporation of DER in the networks. Consequently, the introduction of new ancillary services and explicit services (replacing the previous implicit ones) turns out to be an essential requirement to assure the safe management of the electrical system. Addressing such new services strictly follows the evolution and the integration of multiple electricity markets, to stress the new services needed in the forthcoming panorama, whilst promoting the participation of new flexibility resources in the markets, finally leading these markets to integration at the European level.

The constant evolution of the electricity networks associated with electricity markets' structures, follows the advances in promoting renewables, and, through this, new participants are entering the electricity markets, such as the aggregators, the Flexibility Service Providers (FSP), the Balance Responsible Parties (BRP), among many others.

Many projects and conceptual initiatives have been proposed to improve DER integration as active flexibility providers in local (oriented to distribution grid) and system-wide (oriented to transmission grid) services to contribute to a more efficient operation of the system.

The EUniversal project aims to develop a universal approach to the use of flexibility by DSOs and their interaction with the new flexibility markets, enabled through the development of the concept of the Universal Market Enabling Interface (UMEI) – a unique approach to foster interoperability across Europe.

The need for flexibility provision that is fulfilled with distributed resources and the Renewable Energy Source (RES) that are also connected mostly to the distribution grid are clear signs of the decentralization of the energy system. This leads to the need for enhanced coordination schemes to integrate the decentralized actors and the UMEI aims to be part of the enhanced coordination schemes.

Therefore, within the EUniversal project, a Universal Market Enabling Interface (UMEI) has been developed to overcome current barriers between different systems and to facilitate the use of flexibility services and interlink DSO's active system management with flexibility markets. A set of market-oriented flexibility services from Distributed Energy Resources (DER) will be implemented to serve DSO's needs in a cost-effective way, supporting the energy transition.

1.2 WP7 Objectives

The operative objective of WP7 is to validate the Universal Market Enabling Interface (UMEI) and the developed tools in different contexts and scenarios made available in the Portuguese Demo. It assesses flexibility for distribution grids and the market capacity to provide new services to the DSO. Figure 1 depicts the high-level methodology of development and implementation of the UMEI.

The Portuguese demo has tested the provision of market-based flexibilities from prosumers of the LV and MV grid via the UMEI and the integration for an improved and smarter distribution grid operation. As the first step, establishing the estimation and forecast of the grid state from the chosen LV grid has led to enhanced grid observability. This has helped aggregate and predict the flexibility potential in the LV grid. Other objectives of the demonstration were the use of flexibilities from the low voltage grid to supply the LV/MV connection point and therefore to relieve the MV grid.

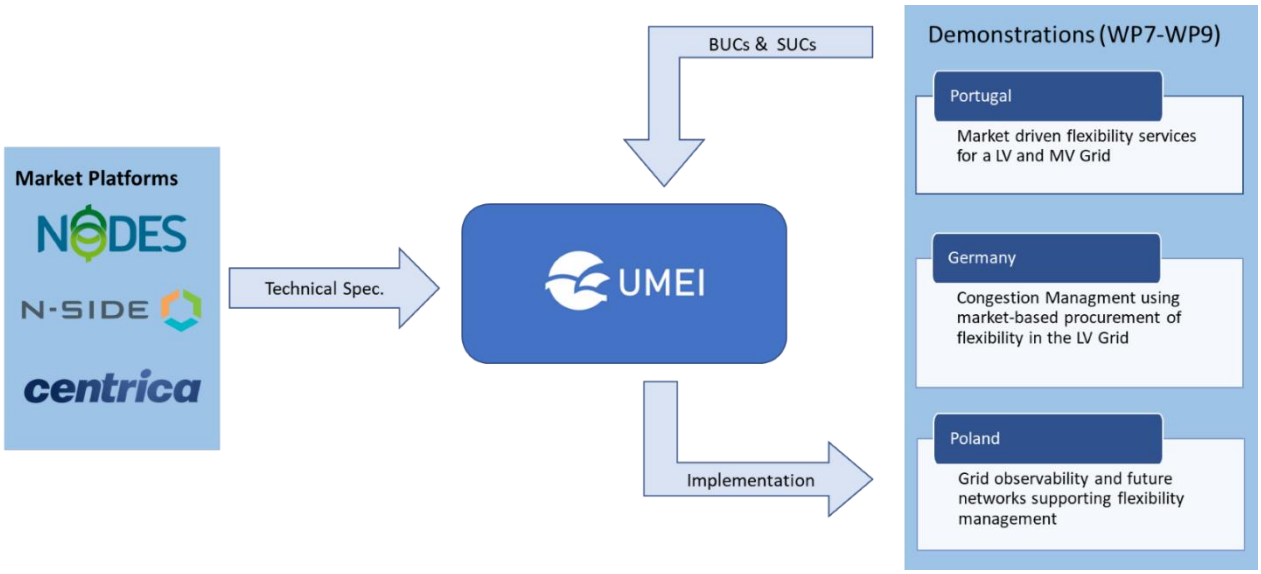

Figure 1 - High-level development methodology [1]

1.3 The UMEI

The UMEI has materialized in the conceptual architecture design and the implementation of a standard, agnostic, adaptable, and modular combination of different APIs to link DSOs and market parties with flexibility market platforms, in coordination with other flexibility users. This approach allows distributed communication without the need for a central hub.

The development of the UMEI was inserted in the workstream of WP2 [2] [3]- which had several objectives, namely the definition of flexibility services, the definition of global architecture for the project through the several developed business and system use cases, and the technical development of the UMEI. Hence, the UMEI is the materialization of the whole process which strengthens the cooperation between the several market parties – DSOs, FMOs and FSPs – to enable the provisioning of the previously defined flexibility services, represented in Figure 2.

Through this distributed architecture, in which every party is responsible for performing the necessary setup that is needed for the whole system to operate, several challenges are also posed. As the UMEI is distributed by design, there are no central registries, databases, or instances of applications, so all the data that is exchanged resorting to this tool must be kept at the origin and/or the destination, being each party responsible for ensuring the good handling of the data.

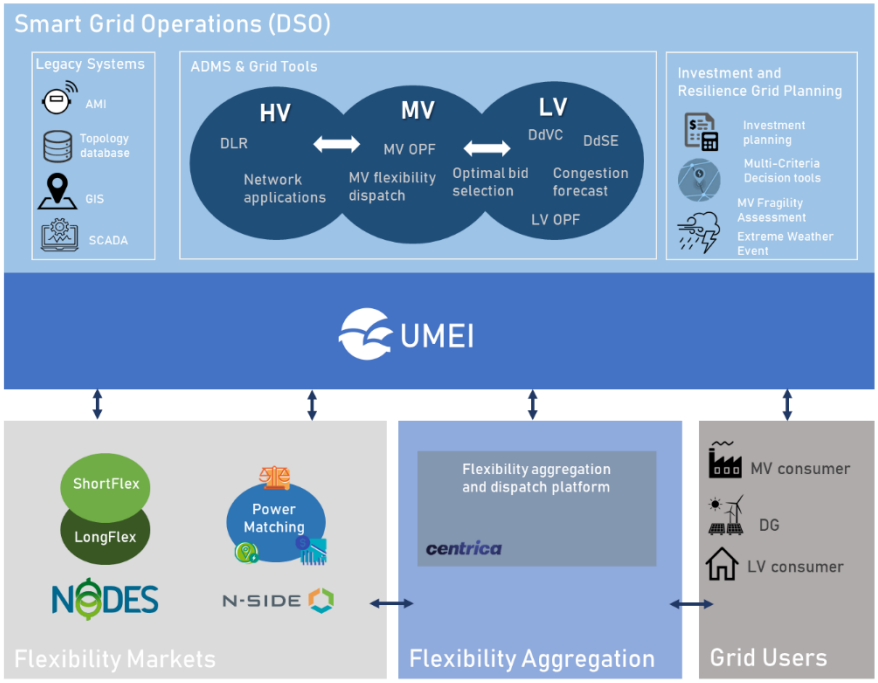

Figure 2 - EUniversal Reference Architecture [1]

The UMEI is composed of a set of APIs, organized in several functional groups, as represented in Figure 3, which allows market participants to retrieve and send information to the Flexibility Market Operators (FMO).

The API interface is divided into several functional groups. The division follows the natural chronological and functional order of operations:

- Prequalification phase – *not covered by this version of the UMEI. In this phase, basic master data like portfolios, grid nodes, etc. are set up*
- Pre-trading phase
  - Defining baselines, the expected power usage, for portfolios
  - Defining flexibility zones – the expected demand for flexibility
  - Extracting this information, along with market and portfolio information, for use in the next phase
- Trading phase
  - Posting orders, reading orders, and trades
- Post-trading phase
  - Providing meter readings for assets used in trades
  - Settlement - *not covered by this version of the UMEI*.

Each group is composed of a set of APIs that allow for CRUD (Create, Read, Update and Delete) operations to be performed on the resources registered on the market platform for flexibility trading. For instance, with the order group it is possible to submit new buy and sell orders, coming from the FSP and DSO side, but also to fetch, eliminate and update orders already submitted before the clearing of the market occurs.

Figure 3 - UMEI Groups [1]

This set of interactions may change depending on the adopted implementation of the APIs. For instance, the implementation of meter data exchange may be done bilaterally between the DSO (in its role as meter data operator) and the FSP (which has control over the resources). In this case, the FSPs implement the Meter Reading group on its own servers, and the DSO acts as an API client directly on that implementation, allowing for the very same CRUD operations to be performed in the same way, although interacting with a different actor of the ecosystem. This last case represents the modularity and flexibility of the UMEI.

Currently, the UMEI focuses on the technical and operational requirements of each stakeholder. The registration and pre-qualification phases were therefore not foreseen to be developed and tested within the scope of the project. The payment processes included in the settlement phase are also not covered by the specification. Certainly, the successful testing of the UMEI will lead to further developments regarding registration and pre-qualification as well as validation and settlement. Anyhow, EUniversal as a research project is not being developed in a specific regulated environment, and without the present definition of flexibility remuneration, this will not allow for any monetary transaction.

The UMEI is publicly available on GitHub[1], allowing for any stakeholder (DSO, Market Operator, Aggregators, Consumer, and even TSO) to adopt it or to develop new APIs concerning new services while complying with the UMEI interface specification.

In chapter 1.4, an overview of the data exchange mechanisms from previous projects, such as InteGrid, CoordiNet, INTERRFACE, and InterFlex, is done.

The UMEI stands apart from these projects by providing a standard, agnostic, adaptable, and modular API framework that is designed to facilitate a distributed communication network for trading flexibility services without relying on a centralized data management system.

1.4 Data exchange mechanisms from previous projects

When analysing previous projects encompassing communication mechanisms for the usage of flexibility, some have developed data exchange tools to cover the flexibility acquisition and

[1] https://euniversal.eu/the-umei/

mobilisation process. EUniversal D1.2 made an extensive summarization of core initiatives in this field, from which some tools are relevant to analysis, this is the case for the InteGrid, Coordinet, INTERRFACE, and InterFlex projects.

Each indentified solution had their own focus areas and technical solutions. InteGrid's mechanism was centered on a data-driven approach to enable smart grid functionalities with a Grid and Market Hub, which differed from UMEI's decentralized approach with no central hub or registry. CoordiNet sought to streamline coordination between TSOs, DSOs, and consumers, creating a cooperation platform for a pan-European market, whereas UMEI's focus is on enabling a set of standard APIs for flexibility services. INTERRFACE aimed to link European electricity market platforms to create a unified marketplace. InterFlex developed the E-Flex platform to facilitate the exchange of local flexibility information.

As it can be seen from the description of the projects below, very often the developed solutions contain a certain degree of platform centralization, relying on a sole software instance to orchestrate tha process data exchange. EUniversal differs from this approach since it developed the UMEI following a decentralized approach, in which there are no single orchestration points and the communications happen end-to-end.

1.4.1 InteGrid

The H2020 InteGrid project[2], was focused on empowering the European electricity grid by integrating advanced smart grid functionalities and enhancing the role of consumers through a data-driven approach. Central to this project is the Grid and Market Hub (GM-Hub), which acts as a neutral, interoperable platform facilitating the connection between various energy sector stakeholders.

The main objective of the GM-Hub was to:

- Enable demand response and smart grid capabilities by allowing different market players, such as DSOs, TSOs, market operators, consumers, and service providers, to interact and exchange data in a secure environment.
- Provide both basic and advanced services, which include:
    - User registration and authentication.
    - Data download and sharing compliant with GDPR for consumer data protection.
    - Advanced functionalities like traffic light systems for flexibility management, feedback on energy consumption, alerts on high consumption patterns, and services for enhancing consumer engagement.
- Maintain a three-tier cloud-based ICT architecture[3] to ensure flexibility, security, and independence between its presentation, application, and data layers. This design supports a wide range of functionalities and can be adapted for use outside its initial cloud environment with minimal integration effort.
- Adhere to standardized data exchange models that are based on industry standards like CIM IEC 61968, ENTSO-E processes, and custom models for innovative services. These models facilitate efficient and standardized communication across the platform's users.

[2] https://cordis.europa.eu/project/id/731218

[3] Three-tier architecture is a software application architecture that organizes applications into three: presentation tier, application tier, and the data tier.

The Gm-Hub (Figure 4) created a standardized, secure, and user-friendly platform that enhances grid operation and encourages consumer participation in the energy market.

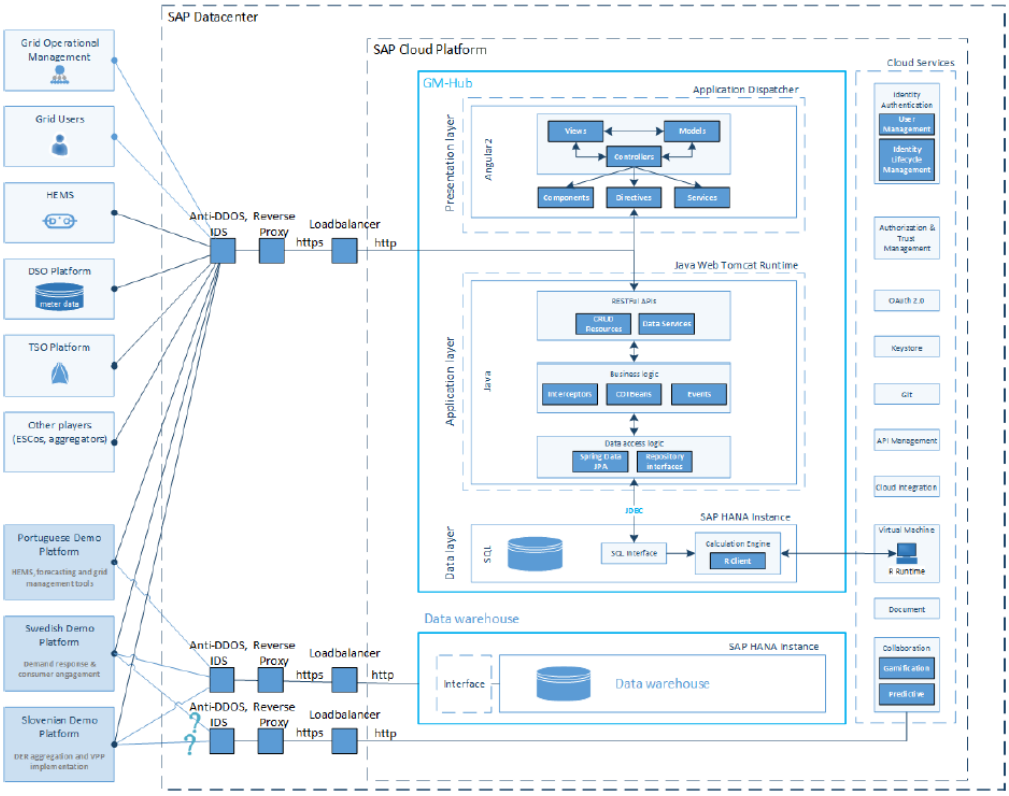

Figure 4 – InteGrid GM-Hub Technical Architecture

1.4.2 CoordiNet

The H2020 CoordiNet[4] aimed at demonstrating the benefits of coordination between TSOs, DSOs, and consumers in providing a more cost-effective, reliable, and green electricity supply. The project revolves around three central objectives, including implementing three large-scale demonstrations across Spain, Sweden, and Greece to validate the potential efficiency gains through coordinated actions in the electricity market.

The initiative also focuses on defining and testing a set of standardized products for system services, which will cover the entire process from reservation to activation and settlement. These standardized processes are intended to facilitate smoother transactions and use of assets within the energy market. A significant outcome of the project is the development of a cooperation platform that will serve as a foundation for a pan-European market, enabling all market players, including end-consumers, to offer system services and thus, create new opportunities for revenue.

The technical emphasis is placed on crafting services for each phase of the operational process, which will interface with a coordination platform that is set up in each demonstration country. This means that specific services are designed to interact seamlessly with the national platforms to ensure

[4] https://coordinet-project.eu/

efficient coordination and data sharing between TSOs, DSOs, and consumers within the energy market of each participating country.

Figure 5 shows the system architecture for the Spanish demonstrator, which leverages the defined data exchange models and sets up a common coordination platform for the exchange of data. A link to the Euniversal project is N-SIDE as active market party.

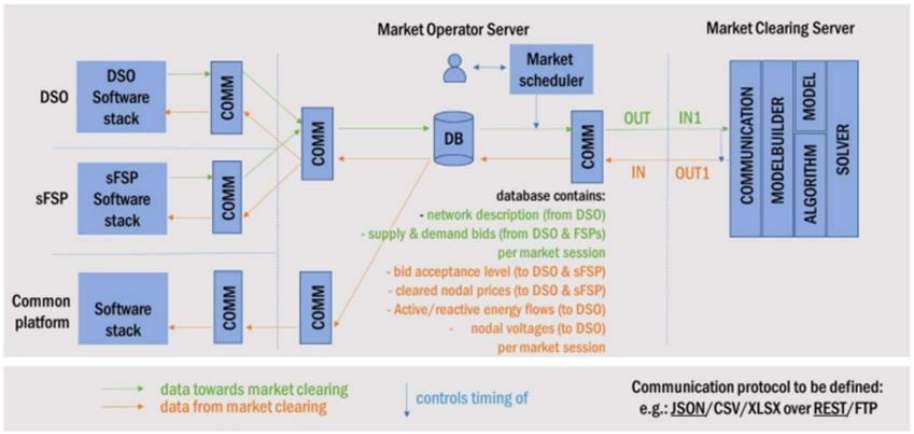

Figure 5 – Coordinet System Architecture for the Spanish Demonstrator

1.4.3 INTERRFACE

The INTERRFACE[5] project aimed to connect European electricity market platforms, creating a trading space for energy services. Its strategic goals involved establishing a common market structure, developing standardized energy service products, improving TSO and DSO cooperation, and integrating a wide range of energy assets to enhance market offerings. The project also intended to introduce digital technologies that are already familiar to consumers in other domains into the energy market.

On a technical level, the project focused on designing the IEGSA, Figure 6, to link market platforms and enable cross-border energy trading. A reference IT infrastructure is being developed to support this architecture. The project is experimenting with digital technologies such as blockchain and IoT to facilitate energy transactions and smart asset management. These technologies are crucial for congestion management, activating local flexibility for system services, and incorporating DERs into the market. INTERRFACE also prioritized consumer engagement in the electricity markets, utilizing demand response and local market mechanisms to offset the variability of renewable energy sources. Demonstrating the effectiveness of the IEGSA and its supporting IT infrastructure is a key objective, as is fostering further research and creating new business opportunities, especially for SMEs and startups through an Open Call funding initiative.

[5] http://www.interrface.eu/

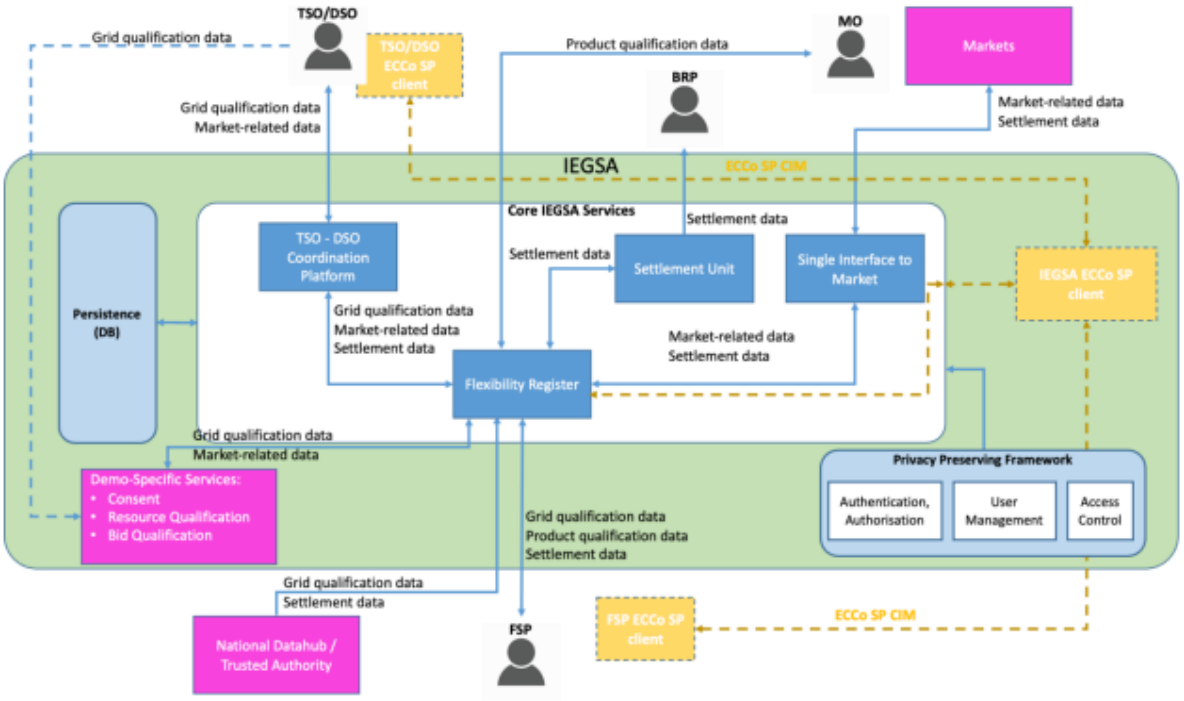

Figure 6 – INTERRFACE IEGSA Architecture [5]

1.4.4 InterFlex

The InterFlex[6] project, under the Horizon 2020 framework, involved the development of the E-Flex platform by Enedis. E-Flex is designed to facilitate the exchange of information for matching supply and demand through a local flexibility mechanism. Aggregators use this platform to post their flexibility offers and receive requests for activation from the distributor as needed. E-Flex manages most of the flexibility process, but it does not handle constraint identification, contract management for flexibility offers, or the calculation of activated volumes. Despite these gaps, Enedis considers E-Flex a success and uses it as a foundation to integrate the flexibility concept into their IT systems.

Enedis is focusing on improving the management of flexibility on four fronts: identifying flexibility needs and constraints, enhancing topological and contractual information, streamlining interactions with flexibility service providers, and providing accurate data for verification and clearing services.

The project utilizes the Common Information Model (CIM), endorsed by international bodies like IEC and ENTSO-e, because it provides a standardized approach that ensures interoperability, particularly between TSOs and DSOs. The E-Flex system has yet to include contractual information, and Enedis is considering various scenarios to address this as part of the Clean Energy Package implementation in France. The concept of reservation is also under development to ensure readiness for offer activation.

[6] https://interflex-h2020.com/

2 Implementation of the UMEI

2.1 Flexibility reference process for the demo

Within WP2, a mapping of the flexibility process, involving several stakeholders and covered by the UMEI was done. The UMEI, as developed in Euniversal, covers the following stages:

- Flexibility needs assessment
- Flexibility procurement/trading
- Flexibility activation
- Measurement data retrieval

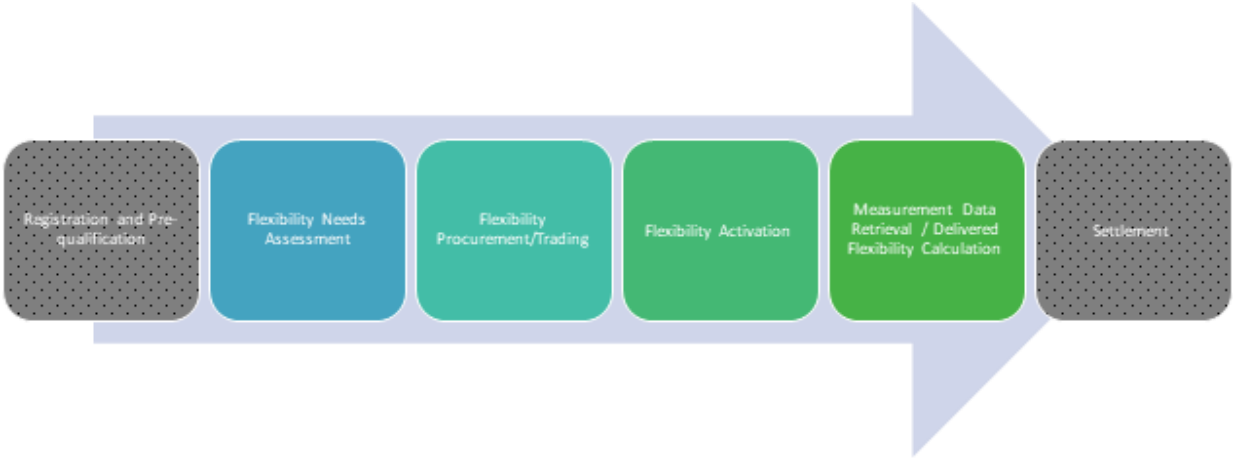

Figure 7 – Flexibility process steps covered by the UMEI [1]

This process was taken by the PT demo and more detailed technical processes were designed and implemented to exchange the necessary information with the market operator, flexibility provider, and respective customers providing the services, in the latter case, outside of the UMEI, since the UMEI is meant to facilitate DSO-FMO-FSP (Aggregator) data exchange. These processes, for NODES and N-SIDE short-term flexibility use cases, are shown in Figure 8 and Figure 9. The steps of the process which contain UMEI-based requests are identified with the blue EUniversal logo.

Further information on the detailed use case steps can be found in [6].

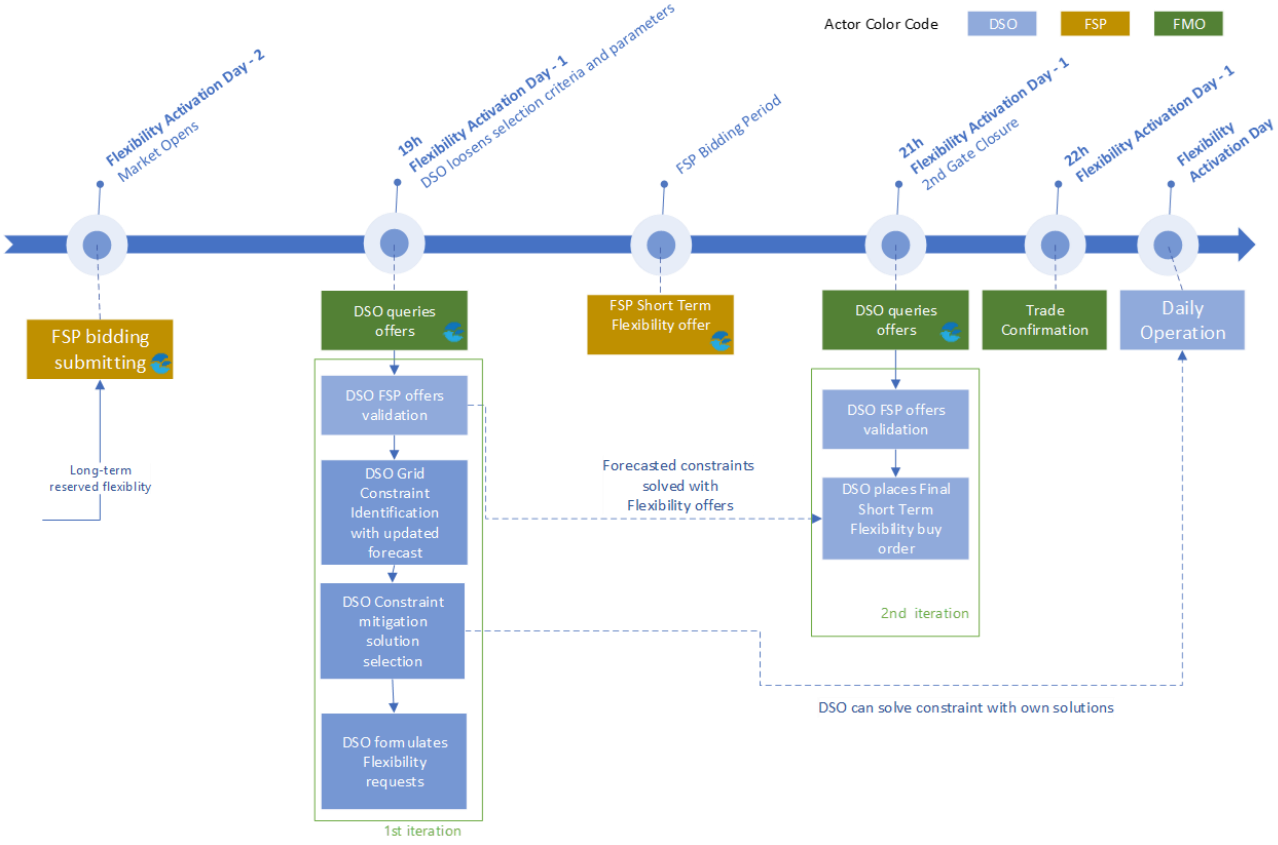

Figure 8 - NODES short-term use case

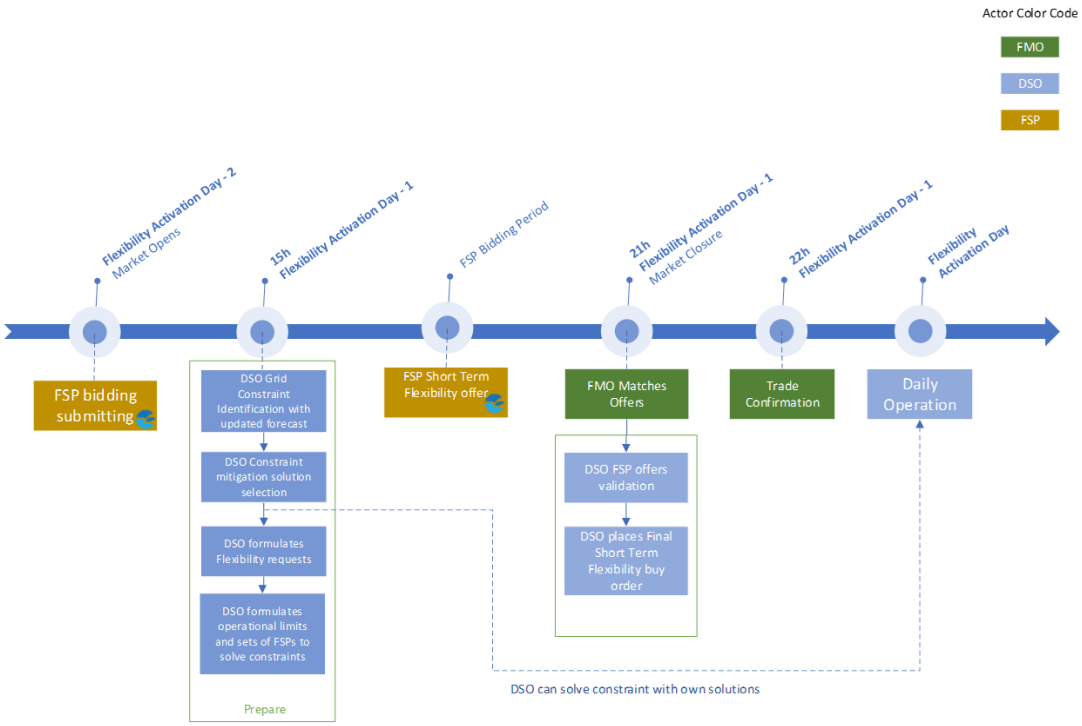

Figure 9 - N-SIDE short-term use case

2.2 Actors and responsibility mapping

A testing and implementation methodology with defined responsibility mapping has been developed jointly between the demo partners to ensure clear delineation of roles and precise daily timings for the execution of tasks by each system participating in the demo, and consequently data exchange through the UMEI.

Each actor's responsibilities and the corresponding timeline are outlined in the provided tables (Table 1 and Table 2) for both the NODES UC and the N-SIDE UC scenarios, indicating a structured approach to managing the bid creation, collection, submission, and activation signal processes within the market platforms. This systematic assignment of tasks and coordination ensures a streamlined workflow, essential for the efficacy of the operations within the flexibility market.

Table 1 - NODES short-term use case daily sequence of actions

#	Step	Responsible	Time (PT)	CET time
1	Create the bids on NODES	CENTRICA	At 2:15	At 2:15pm
2	Collect bids from NODES platform	E-REDES	2:35pm to 2:50pm	3:35pm to 3:50pm
3	Send the bids to INESC TEC tool	E-REDES	2:35pm to 2:50pm	3:35pm to 3:50pm
4	Data process	INESC TEC	min:5 min	
5	Request selected offers (MV&LV)	E-REDES	2:55pm to 3pm	From 3:55pm
6	Submit offers to NODES platform	E-REDES	2:55pm to 3pm	From 3:55pm
7	Collect NODES market results	CENTRICA	3pm	4pm
8	Activation Signal	CENTRICA	From 3pm	LV: Automatic MV: Message to E-REDES

Table 2 - N-SIDE short-term use case daily sequence of actions

#	Step	Responsible	Time (PT)	CET time
1	Available Flex Areas	INESC TEC	2pm	3pm
2	Submit flex areas to N-SIDE platform	E-REDES	From 2pm	From 3pm
3	Retrieve Flex Areas from N-SIDE	CENTRICA	From 2pm	From 3pm
4	Submit offers to N-SIDE	CENTRICA	At 2:15pm	At 3:15pm
5	Market closing	N-SIDE	2:50pm	3:50pm
6	Market clearing	N-SIDE	3pm	4pm
7	Collect market results	CENTRICA	From 3pm	From 4pm
8	Activation Signal	CENTRICA	From 3pm	LV: Automatic MV: Message to E-REDES

2.3 UMEI API and Host selection

The steps for the implementation of the UMEI are simple for anyone familiar with REST API implementation. The following steps are necessary:

1) Map the interactions for your flexibility-related process and the information to be exchanged
2) Identify the involved actors in the exchange of data
3) Identify which actor is going to provide and receive information upon request (reactively)
4) Check the UMEI list of available APIs and identify which one fits the purpose
5) The actor identified in 3. will be the one to host the chosen API (implement server-side code) and make it available for the other(s)

For the implementation, actors can as an example leverage OpenAPI features enabled by Swagger[7] and generate client and server-side boilerplate code. The setup process for the implementation implies that there is the possibility to have more than one hosting actor for the system, and no need for a mediation platform, making the UMEI a truly distributed data exchange interface. Naturally, whoever party implements the API becomes responsible for managing and providing credentials to others. This makes the usage of the UMEI particularly straightforward for actors that act mostly as UMEI API clients (e.g., DSO and FSP) since they just need to invoke an API endpoint to send and retrieve information.

[7] https://swagger.io/

Table 3 - UMEI overall structure

Group	Name	Usage
Baseline	Managing portfolio baselines	Used by the FSPs to manage baselines into the market platform.
Order	Manage Market Orders	Used by the DSOs and FSPs to view and execute orders' related operations in the market platform. The FMO will perform clearing/matching, either continuously or on a specific schedule, between orders. The result of this process will be the trades.
Meter Reading	Manage Meter Readings	Used by the DSOs, and possibly other market participants, to submit and manage metering data
Market	List All Markets	Used by market participants to get the available markets
Portfolio	Manage Portfolios	Used by FSPs to submit and manage portfolios on the market
Trade	List Market Trades	Used by market participants to retrieve the market trades (the result of the matching process between buy/sell orders)
Flexibility Zones	Manage Flexibility Zones	Used by DSOs to define specific flexibility areas, composed of a set of portfolios

The table below shows the steps which were implemented in the PT demo resorting the UMEI, corresponding to the white cells. It is possible to note that the API host was always the FMO. The creation of portfolios was done in the GUI of the market operator because it was more straightforward and the resources and grids were static, and the meter readings retrieval from the DSO side was not applicable to the PT demo scope.

Table 4 - Flexibility steps and participating actors

Step	Actors			UMEI Endpoint
	DSO	FMO	FSP	
Send flexibility orders to the market	X	X (API host)	X	POST /Orders And POST /Flexibility Zones
Consult the sell bids in the market	X	X (API host)		GET /PublicOrders
Retrieve market clearing results	X	X (API host)	X	GET /Trades
Create a new portfolio of flexible resources		X (API host)	X	POST /Portfolios
Periodically send meter readings to the FSP	X		X (API host)	POST /MeterReadings/create-multiple
Send meter readings	X	X	X (API host)	POST /MeterReadings

In Table 4, the lines in grey color were not tested in the PT demo because these steps were done directly in the GUI of the market platforms since they were one-time configurations.

3 Technical deployment

3.1 Technological choices and architecture

The development and implementation of the necessary software to use available APIs from NODES, N-SIDE, and CENTRICA, and integrate them with the legacy DSO software, has been achieved by implementing a set of modules using an Azure cloud resource group (infrastructure used by the IT processes of E-REDES) which will host the necessary DSO tools defined for performing the necessary calculations and mechanisms to retrieve and process data. The architecture for such a system is presented in Figure 10.

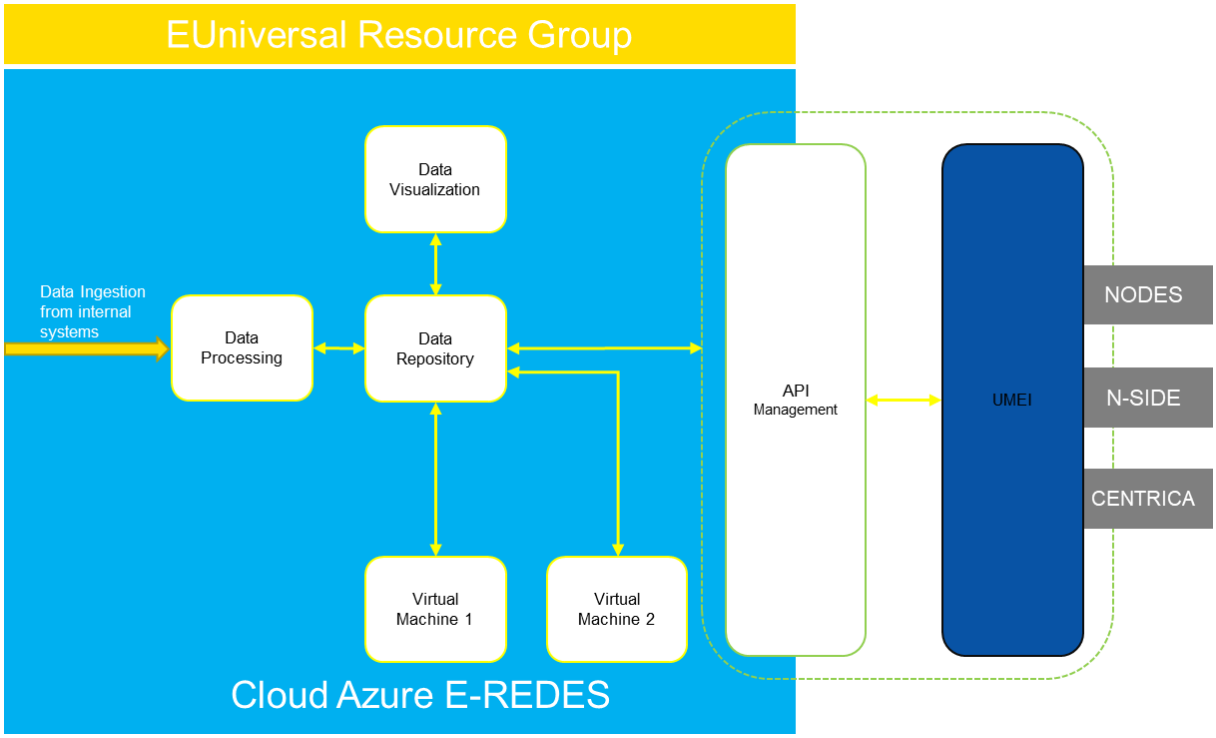

Figure 10 - DSO Architecture [1]

To ensure the security of the proposed solution, this platform is divided into two areas with different communication requirements/constraints:

- Area 1: Dedicated to important systems that use customer data. This area is designed to contain both the information and the resources, so that is only accessible to key users of the project. The objective of this area is to ensure systems' isolation from potential external threats.
- Area 2: Dedicated to communication with external platforms and the implantation of the software to use the referred APIs. The purpose of this area is to ensure that a separate layer exists between the solution's "core" systems (area 1) and the components responsible for communicating with external platforms.

3.2 Authentication mechanisms

The UMEI API specification allows implementers to choose their preferred authentication methods. Rather than dictating a specific approach, flexibility is offered for users to select authentication mechanisms that best fit their corporate security requirements. This practical approach ensures easy integration into various technological environments, allowing prioritisation of security based the specific needs.

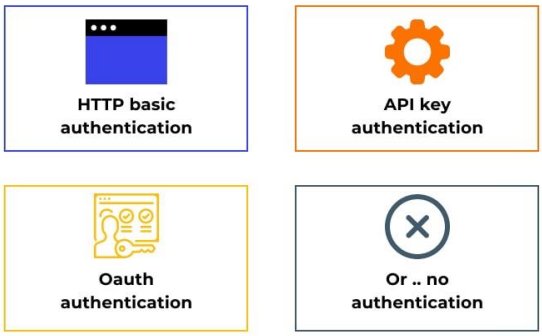

Figure 11 - Basic API Authentication modes [7]

Ensuring the secure and authorized access of sensitive data through APIs is a fundamental aspect of modern application development. In this context, two prevalent authentication mechanisms come to the forefront as main examples: OAuth 2.0 and token-based authentication. Each one of these authentication mechanisms was used by the Flexibility Market Operators in EUniversal, by NODES and N-SIDE, respectively.

OAuth 2.0 is a protocol explicitly designed for the secure authorization of third-party applications, allowing them access to user data without exposing sensitive credentials. This process involves four key entities: the resource owner (typically the user), the client application, the authorization server, and the resource server. Within the OAuth 2.0 framework, the client initiates the process by requesting authorization from the resource owner. Upon approval, the client obtains an access token from the authorization server. This token serves as a temporary credential that the client presents to the resource server to gain access to the specified resources. The authorization server plays a crucial role in validating the request and ensuring the proper issuance of access tokens.

On the other hand, token-based authentication is a method where the client is authenticated through a unique token rather than a direct verification of credentials with each request. This approach is particularly common in stateless applications like mobile or single-page applications. In a token-based authentication scenario, the user first provides their credentials for authentication. Upon successful authentication, the server generates a unique token, which is then transmitted to the client and securely stored. When the client needs to access protected resources, it includes this token in the header of its API requests. The server, in turn, validates the authenticity of the token and checks its permissions before granting access to the requested resources.

In the context of the PT pilot, as E-REDES (DSO) and CENTRICA (FSP) have to exchange data with both platforms, they had to implement both authentication methods, however this is a fairly straightforward process that is easily achieved when having the appropriate credentials and the right testing tools (e.g. postman, swagger, etc.).

4 Usage of the UMEI

This chapter presents an analysis of the usage patterns of the UMEI, particularly from E-REDES, CENTRICA, N-SIDE, and NODES systems, including statistics over different periods of time.

It offers insights into the number and types of requests made, the endpoints targeted, as well as the success and error rates. This analysis serves as a tool to assess the system's current usage patterns and potential bottlenecks.

4.1 Usage statistics of the API for N-SIDE (Market Operator)

Between the 1st of September and 30th of November 2023, a total of 17,061 calls were initiated. Out of these, 13,480 were GET requests, with the majority aimed at the FlexZone endpoint, and 3,581 were POST requests. Successful calls during this period numbered 14,545, representing an 85.2% success rate.

A significant portion of the error rate, specifically 93%, was due to failures in POST requests.

Considering the data from the most recent complete 24-hour window, there were 363 calls. A breakdown of these calls is as follows: 165 were GET requests, all made by a user likely to be identified as Centrica, and 198 were POST requests, with 135 initiated by Centrica and the remaining 63 by a different user. The success rate for this day was exceptionally high at 98%, with 356 calls being successful.

All 7 failed calls within this last day were the result of POST requests targeted at the Orders endpoint that contained invalid FlexZone data.

In summary, since the beginning of September (3 months):

- 17061 calls have been made
- 13480 GET (mainly on the FlexZone)
- 3581 POST
- 14545 calls succeed (85.2%)
- 93% of the error come from POST request

1 day (i.e. exemplar 24h time window):

- 363 calls have been made
- 165 GET (all from the same user: Centrica I guess)
- 198 POST (135 from Centrica, 63 from another user)
- 356 calls succeed (98%)
- The 7 errors come from POST on Orders with an invalid Flexzone

4.2 CENTRICA API Usage (Flexibility Service Provider)

Takin the week period of December 11th to 19th, a total of 1166 bids were placed by Centrica to the flexibility market: 589 bids were sent to NODES and 577 were sent to N-SIDE.

For every N-SIDE bid, an API call was made to the FlexibilityZones API (see 1.3) to check the availability of zones before placing the bid, resulting in at least 577 API calls. The Trading API was called 1166 times to verify if the bids were accepted once the market cleared.

The logs below are examples of the process in action: A query for flexibility zones would be sent, and depending on the availability, responses could either indicate that there were no zones available or provide details of the available zones.

For sell order bids, information about the order, such as the organization details, type of order, and pricing, was sent, followed by a confirmation of reception with the status "Pending".

Checking if a bid was accepted was also logged, with an API call. A response with no trade indicated an unaccepted order, whereas the presence of a URL meant that follow-up actions were necessary to determine the status of a bid.

Below is a sample of the obtained logs from CENTRICA systems for several cases: (1) Get flex zones, (2) Sell order bid, (3) Check bid acceptance.

Get flex zones:

```
2023-12-16 14:15:21.197 | DEBUG | euniversal.control_service.umei.nside_client:get_flexibility_zone:62 - send to UMEI https://pom-euniversal-pt.n-side.com/v0/FlexibilityZones?periodFrom.gte=2023-12-17T08:30:00Z&periodTo.lte=2023-12-17T09:00:00Z
```

Return if no zones available:

```
2023-12-16 14:15:21.298 | DEBUG | euniversal.control_service.umei.nside_client:get_flexibility_zone:68 - received b'{"items":[]}'
```

Otherwise:

```
2023-12-11 14:15:05.947 | DEBUG | euniversal.control_service.umei.nside_client:get_flexibility_zone:62 - send to UMEI https://pom-euniversal-pt.n-side.com/v0/FlexibilityZones?periodFrom.gte=2023-12-12T13:30:00Z&periodTo.lte=2023-12-12T14:00:00Z

2023-12-11 14:15:06.033 | DEBUG | euniversal.control_service.umei.nside_client:get_flexibility_zone:68 - received b'{"items":[{"id":"3648f0b6-08f8-491f-bd04-e1f2bf8e1947","periodFrom":"2023-12-12T13:30:00Z","periodTo":"2023-12-12T14:00:00Z","portfolioIds":["portfolio6"]},{"id":"8c8dd3fd-d415-4af9-a429-7167b5d667f1","periodFrom":"2023-12-12T13:30:00Z","periodTo":"2023-12-12T14:00:00Z","portfolioIds":["portfolio1"]},{"id":"bac1a9e8-a87a-4cd5-a28d-f4aa5b18a36e","periodFrom":"2023-12-12T13:30:00Z","periodTo":"2023-12-12T14:00:00Z","portfolioIds":["portfolio2"]}]}'
```

Sell order bid:

Send

```
2023-12-16 14:15:17.021 | INFO | euniversal.control_service.umei.umei_client:send_sell_order:48 - Sell order is {'ownerOrganizationId': 'db2e743a-1fb0-4c09-b06a-4318a07fb8ae', 'gridNodeId': '4fa2dc70-f413-40f1-a59a-d6646283abaa', 'marketId': '8ae941a5-21b5-4350-bac0-01fc66ad24aa', 'portfolioId': '269daac5-401d-474d-951a-1565e40175d0', 'regulationType': 'Down', 'side': 'Sell', 'pricePoints': [{'quantity': 0.002, 'unitPrice': 0.00042855999999999987}], 'minimumAcceptanceQuantity': 0.001, 'periodFrom': '2023-12-17T01:00:00 +0000', 'periodTo': '2023-12-17T01:30:00 +0000'}
```

Receive

```
2023-12-16 14:15:17.216 | DEBUG | euniversal.control_service.umei.umei_client:send_sell_order:57 - received b'{"id":"550738d3-67b3-4b21-ab7b-21ba9e376861","ownerOrganizationId":"db2e743a-1fb0-4c09-b06a-4318a07fb8ae","status":"Pending","completionType":null,"gridNodeId":"4fa2dc70-f413-40f1-a59a-d6646283abaa","flexibilityZoneId":null,"marketId":"8ae941a5-21b5-4350-bac0-01fc66ad24aa","portfolioId":"269daac5-401d-474d-951a-1565e40175d0","regulationType":"Down","side":"Sell","pricePoints":[{"quantity":0.002,"unitPrice":0}],"minimumAcceptanceQuantity":0.001,"periodFrom":"2023-12-17T01:00:00Z","periodTo":"2023-12-17T01:30:00Z","longflexContractId":null}'
```

Checking if a bid was accepted

```
2023-12-11 17:30:01.075 | DEBUG | euniversal.control_service.umei.umei_client:get_trade_by_orderid:110 - send to UMEI https://umei-extern-test.nodesmarket.com/umei/Trades?orderId=74a007a3-218d-41c6-920d-c7cd8748cc94
```

If order was not accepted, there are not trades

```
2023-12-11 17:30:01.572 | DEBUG | euniversal.control_service.umei.umei_client:get_trade_by_orderid:112 - received b'{"numberOfHits":0,"items":[],"links":null}'
```

Otherwise

```
2023-12-11 17:30:00.767 | DEBUG | euniversal.control_service.umei.umei_client:get_trade_by_orderid:110 - send to UMEI https://pom-euniversal-pt.n-side.com/v0/Trades?orderId=b3d3df93-ee8d-4ba9-85ee-8f51d59aa49b
```

4.3 E-REDES API Usage (DSO)

This section provides statistics on the usage of the UMEI during a week-long PT demo conducted by E-REDES. The demo involved integration with two flexibility market operators, N-SIDE and NODES. The focus of the usage was on the Orders and Flexibility Zones endpoints, as the portfolios were already pre-defined, and the measurements were analysed in a post-operation scenario.

Figure 12 refers to the number of API calls per day of operation, this number is quite stable throughout the different days except 7/12, in which there weren't many bids to the market being done. The error rate of the API requests was typically around 5%.

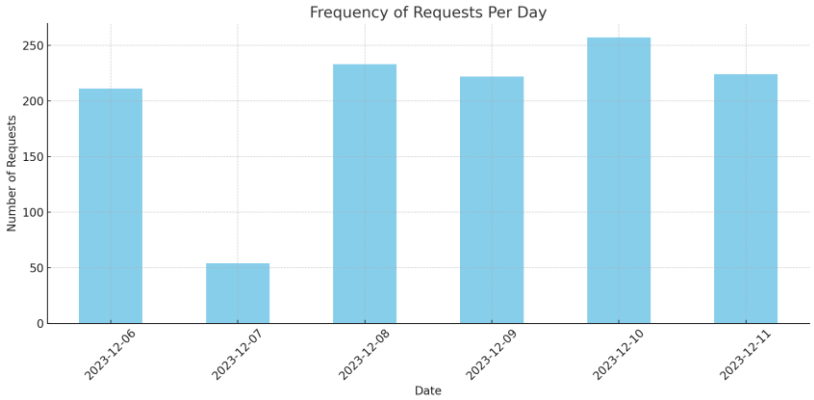

Figure 12 - Frequency of API requests / day

Throughout the week of operation, the majority of the HTTP methods used were focused on posting data, specifically creating bids in the local flexibility markets. A smaller portion of the methods were used for consulting data, retrieving bids from the FSPs, and assessing the market potential for addressing the identified flexibility requirement.

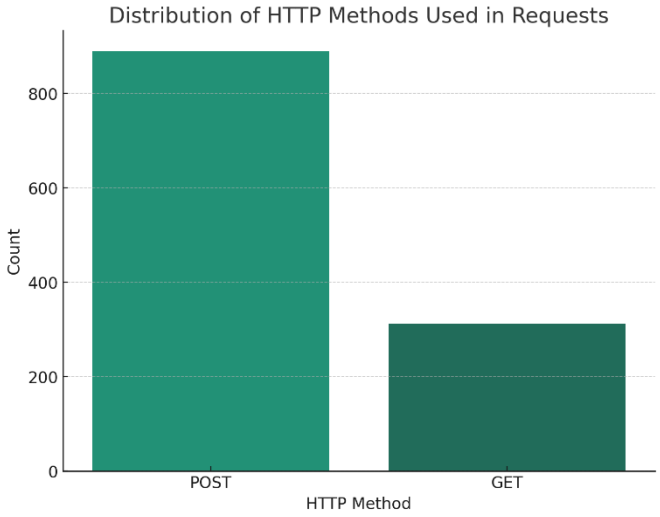

Figure 13 - HTTP Methods of the Requests

The Figure 13 and Figure 14 depict the distribution of HTTP requests made, categorized by the request type (POST and GET), the flexibility market operator (either NODES or N-SIDE), and the specific endpoint of the process group API utilized for each request.

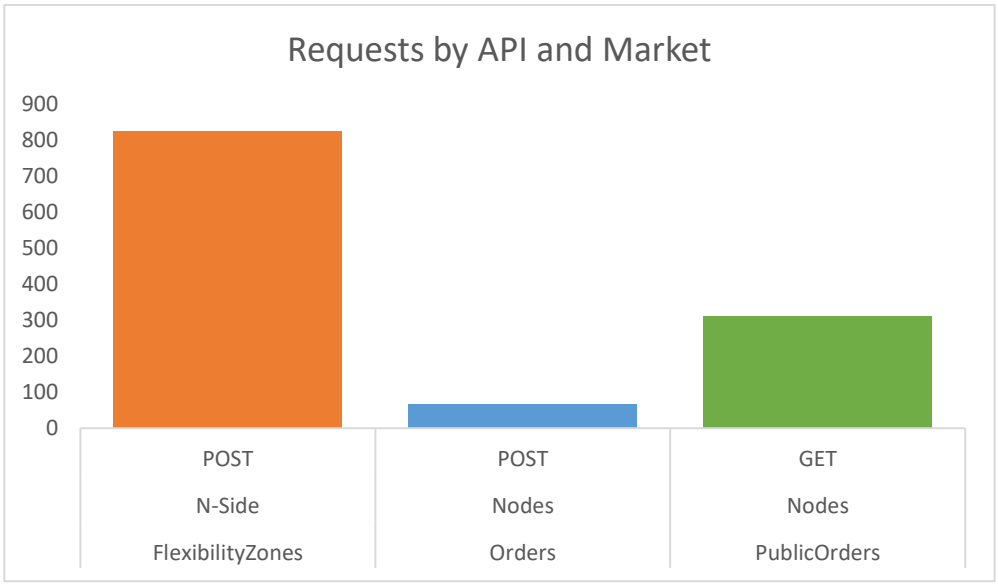

Figure 14 - Requests by API and Market

This week-long excerpt of the PT demo provides valuable insights into the functional utilization of the UMEI interface. The predominant part of the requests was cantered on the creation of bids via POST HTTP methods. The stable frequency of API calls depicts a consistent use-case scenario across the week. Despite this, the 5% error rate suggests there is room for improving the reliability and efficiency of the API and the procedures for using it.

These statistics underscore the importance of monitoring, analysis, and continual optimization of the system. Understanding usage patterns helps in identifying performance bottlenecks, informing future upgrades, and ensuring the API's alignment with user requirements and market dynamics. Moving

forward, fostering a lower error rate and ensuring high availability during peak market transactions will be strategic for enhancing user experience and ensuring the seamless operation of flexibility markets.

5 Conclusions

5.1 Implementation challenges and lessons learned

The lessons learned highlight that the UMEI does not eliminate the need for thorough testing the interoperation of systems from different companies once the APIs are implemented. One reason for continued testing is that the use of APIs may generate unexpected errors in the system setup phase. This unintended usage leads to scenarios where the same implementation cannot be simply copied and used across SOs and FSPs without appropriate testing.

Additionally, best practice guidelines would be beneficial for both clients and hosts as they implement these interfaces. Clear guidelines would help ensure that implementations meet the standards required for effective interaction between API clients and API hosts.

In the process of data exchange between FSPs and DSOs, there is a specific requirement from the DSO's voltage control tool that needs each FSP to provide detailed, disaggregated bid information [8]. This level of detail goes beyond aggregated bids, requiring that FSPs not only submit an overall bid but also break down these bids to the level needed by the DSO's grid interest points. To accommodate this requirement, the FSPs must send their disaggregated bids, or sufficiently detailed aggregate bids, to the FMO, from which the DSO can then retrieve the information. The UMEI supports this process by allowing these two types of messages, aggregated and disaggregated, to be sent in parallel without necessitating any modifications to the existing message specifications. However, the market rules and design from a business perspective may pose additional challenges in terms of providing visibility of the disaggregated bids under the same network, since the anonymous characteristic of a local flexibility market is not compatible with the usage of flexibility services for voltage control in the LV networks.

5.2 UMEI advantages and improvement opportunities

Utilizing the UMEI offers benefits for System Operators, FMOs, and FSPs, and ultimately may improve the consumer experience. With the UMEI in place, SOs can standardize their communication with market platforms, irrespective of the specific market platform in use. Present and emerging FMOs can leverage a tried-and-tested interface to initiate new markets, thus simplifying the collaboration process with other stakeholders. This paves the way for easier penetration into new markets and regions. For FSPs, the benefit lies in the ability to engage with multiple marketplaces via a single implementation of the UMEI, the only necessity being the appropriate endpoint details for each application. More importantly, the interface lets any actor independently host the APIs, eliminating dependence on a single hosting entity.

A Scalability and replicability analysis was performed as a task of the project [9]. The results described in the **ANNEX**, show that the UMEI API presents, in general, good compliance with best practices of REST API design. UMEI follows all the rules for using HTTP request methods, versioning, and representation design. In certain implementations, the UMEI can also apply all the rules related to client concerns and error handling.

The category where the UMEI presents lower quality is metadata design, followed by the category of client concerns when considering the baseline case, Figure 15. **Nevertheless, the best practices included in these two categories are the ones commonly considered by expert developers as the least relevant rules for API design** [11]. In addition to this, the rules in these categories account for less than 12% of the list. Therefore, considering this, the scalability and replicability of UMEI are expected to not be strongly affected by the low scores in these categories.

This study also highlighted potential technical improvements to the specification, which can be seen as potential avenues for evolution:

- TSO compatibility (for instance, for pre-qualification)
- Integrating additional flexibility process stages (like financial settlement)
- Synchronization of flexibility registers
- Creation and distribution of official client software in one or several programming languages
- Establishment and dissemination of an official test suite or toolset for implementation validation

.

6 ANNEX (From deliverable 10.3)

6.1 UMEI API SCALABILITY AND REPLICABILITY ANALYSIS

6.1.1 Motivation and methodology

The Universal Market Enabling Interface (UMEI) developed within the EUniversal project materializes into publicly available Application Programming Interfaces (APIs) that support the interactions between the different actors and the new flexibility markets. These APIs have been specified in EUniversal deliverables D2.4 [3] and D2.5 [2].

For every technical development, an SRA helps to determine the potential of a solution to be replicated outside the demonstration sites, and how it can increase its range of action, or the number of actors involved. When analyzing Information and Communication Technologies (ICT), two approaches can be differentiated: quantitative (e.g., simulations or laboratory experiments of communications between the devices/systems involved in a use case) or qualitative (e.g., aspects such as interoperability, robustness, or reliability).

A quantitative approach to analyze the UMEI API is not appropriate for two reasons. First, because the communications would be done through the internet, which is difficult to simulate accurately, and because it does not rely on ad-hoc communication infrastructures as other solutions. And secondly, because an API following a Representational State Transfer (REST) architecture, which is the case of the UMEI API, already provides great scalability from the technical point of view.

Qualitatively, by design, the UMEI API is conceived to be agnostic, adaptable, and modular, and to provide interoperability between DSOs, market parties, and platforms. This means that all the stakeholders should be able to implement it, regardless of the data models and standards they use in their systems (e.g., CIM, IEC 61850, etc.).

Despite the fact that these characteristics guarantee a great level of technical scalability and replicability, the implementation of an API may be facilitated or hampered by its design rules. That is to say, if other developers find it difficult to understand and use the designed API or following versions, the possibilities of replicating and scaling-up the UMEI are reduced. Therefore, the scalability and replicability of the UMEI API will be ultimately related to its understandability and reusability, which are achieved when the best practices for REST API development are applied [11].

To evaluate the quality of the UMEI API in these terms, a list of up to 69 best practices has been collected from existing guidelines and similar studies [11] [8] [12] [9] [13]. These best practices are divided into seven categories:

- **Uniform Resource Identifier (URI) design**. (Table 5) A list of best practices and common rules that would improve the understandability and reusability of the URIs by future developers that use the API.

- **Request methods**. (Table 6) The implementation of HTTP methods such as PUT, GET, POST, DELETE or HEAD, should follow some basic rules so that the API can be correctly implemented by future developers that use the API.

- **Error handling**. (Table 8) The practices in this category define some rules on how HTTP messages must be used as a response to a HTTP request method [11].

- **Metadata design**. (Table 9) The practices in this category specify how HTTP headers should be used to complete requests with metadata [11].

- **Representation design**. (Table 7) This category checks the consistency of the API to represent media type formats, schemas, resources, and error responses.

- **Client concerns**. (Table 10). Rules relevant for API clients.

- **Versioning**. (Table 11) This category provides the best practices in how the versions of the APIs should be identified [14]. This category is directly related to replicability, as a bad versioning system may make implementations of the API much more complex for developers.

To check the compliance of the UMEI API with this list of best practices, partners from WP2 were asked to fill in the checklist with a "Yes", "No", "Not sure", or "Not applicable N/A". The results obtained are discussed in the following subsection.

6.1.2 Results

Figure 15 shows the compliance of the UMEI API with the best practices for REST API design based on the information provided by WP2 partners. The score for each category, represented by a percentage, has been calculated by dividing the number of "Yes" (i.e., practices followed) by the total number of practices that could be applicable to UMEI. That is, those practices where the answer was "N/A" were not considered in the calculation. It must be highlighted that the UMEI API allows for a certain degree of freedom when implementing it, so some specific practices may be followed in some implementations and not in others. For this reason, Figure 15 shows two cases. The blue line represents the baseline case or worst-case scenario, that is, an implementation of the UMEI where none of the implementation-dependent practices are followed. On the other hand, the orange dashed line represents the potential case, which considers that all the best practices that may be followed during implementation are indeed applied.

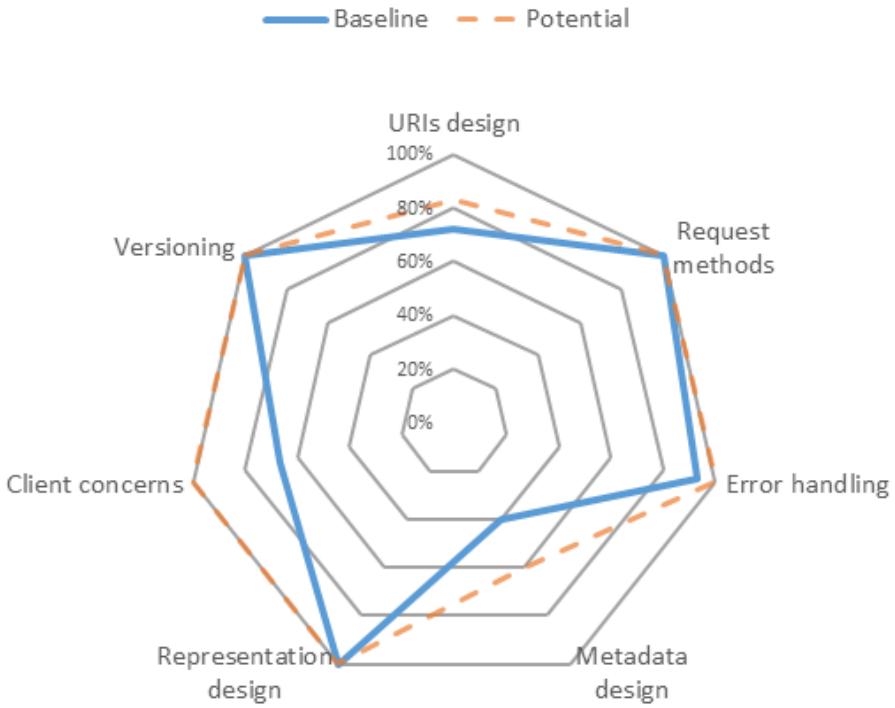

Figure 15 - Compliance of the UMEI API with the best practices for the design of REST APIs that have an impact on its scalability and replicability. [10]

Starting with how the **URIs are designed**, the UMEI API got a baseline score of 72.2% and a potential score of 83.3%. As shown by Table 5 three best practices were considered not applicable to the UMEI API so they were not considered to calculate these scores. There are two practices that are not followed:

- Using only lowercase letters in URI paths: the implementation of the UMEI API might be case sensitive. This may cause some trouble to developers in case an error arises during implementation due to this reason. Therefore, developers will have to pay special attention to the type of letters in URI paths.
- Avoiding version number in the path. It is expected that the UMEI API will include the version number in the URI path. Developers will have to know at every moment which API version they are using.

Table 5 - Best practices for URIs design

Category: URIs design	Compliance
A trailing forward slash (/) should not be included in URIs	No
File extensions should not be included in URIs	Yes
A plural noun should be used for store names	Yes
A verb or verb phrase should be used for controller names	Yes
The query component of a URI may be used to filter collections or stores	Yes
Forward slash separator (/) must be used to indicate a hierarchical relationship	Yes
Hyphens (-) should be used to improve the readability of URIs	N/A
Underscores (_) should not be used in URI	Yes
Lowercase letters should be preferred in URI paths	No *
A singular noun should be used for document names	N/A
A plural noun should be used for collection names	Yes
Variable path segments may be substituted with identity-based values	N/A
Avoiding version number in the path	No
Avoiding version number in the query parameters	Yes
Avoiding CRUD actions in query parameters	Yes
Consistent subdomain names should be used for the API	NS *
CRUD function names should not be used in URIs	Yes
Use path variables to separate elements of a hierarchy, or a path through a directed graph	Yes
API as part of the subdomain	NS
The query component of a URI should be used to paginate collection or store results	Yes
Keeping as much information as possible in the URI, and as little as possible in request metadata	Yes

*implementation specific

**implementation might be case sensitive

In addition to this, two best practices related to subdomains (using consistent subdomain names and including the API as part of the subdomain) depend on the specific implementation of UMEI.

For the best practices when using HTTP **request methods**, shown by Table 6, and **representation design**, shown by Table 7, the UMEI API got the maximum score of 100% in both the baseline and potential cases. Since the API is expected to not use the HEAD method, the rule associated to it was retrieved from the analysis.

Table 6 - Best practices for request methods

Category: Request methods	Compliance
PUT must be used to both insert and update a stored resource	Yes
GET and POST must not be used to tunnel other request methods	Yes
GET must be used to retrieve a representation of a resource	Yes
POST must be used to create a new resource in a collection	Yes
POST must be used to execute controllers	Yes
DELETE must be used to remove a resource from its parent	Yes
HEAD should be used to retrieve response headers	N/A
PUT must be used to update mutable resources	Yes

Table 7 - Best practices for representation design

Category: Representation design	Compliance
XML / JSON may optionally be used for resource representation	Yes
Minimize the number of advertised "entry point" API URIs	Yes
Consistent form to represent media type formats	Yes
Consistent form to represent media type schemas	Yes
Consistent form to represent error responses	Yes

The UMEI API also shows very good design in **error handling** with a score of 92.85% and 100% in the baseline and potential cases, respectively. As shown by Table 8, up to five practices were considered not applicable to the UMEI API, and only one depends on the implementation (HTTP error 304, "Not modified", that should be used to preserve bandwidth).

Table 8 - Best practices for error handling

Category: Error handling	Compliance
200 ("OK") should be used to indicate nonspecific success	Yes
200 ("OK") should not be used to communicate errors in the response body	Yes
201 ("Created") must be used to indicate successful resource creation	Yes
202 ("Accepted") must be used to indicate successful start of an asynchronous action	N/A
204 ("No content") should be used when the response body is intentionally empty	Yes
301 ("Moved permanently") should be used to relocate resources	N/A
302 ("Found") should not be used	Yes
304 ("Not modified") should be used to preserve bandwidth	No *
400 ("Bad request") may be used to indicate nonspecific failure	Yes
401 ("Unauthorized") must be used when there is a problem with the client's credentials	Yes
403 ("Forbidden") should be used to forbid access regardless of authorization state	Yes
404 ("Not found") must be used when a client's URI cannot be mapped to a resource	Yes
405 ("Method not allowed") must be used when the HTTP method is not supported	Yes
406 ("Not acceptable") must be used when the requested media type cannot be served	N/A
409 ("Conflict") should be used to indicate a violation of resource state	N/A
412 ("Precondition failed") should be used to support conditional operations	N/A
415 ("Unsupported Media Type") must be used when the media type of a request's payload cannot be processed	Yes
500 ("Internal Server Error") should be used to indicate API malfunction	Yes
Use JSON as error message response	Yes

*implementation specific

Regarding **metadata design**, it is the category where the UMEI API gets the lowest scores: 40% for the baseline, and 60% for the potential case. Table 9 shows that the UMEI API does not use content-length in the metadata and it also does not use location to specify the URI of a newly created resource. Depending on the implementation, caching may be used.

Table 9 - Best practices for metadata design

Category: Metadata design	Compliance
Content-Length should be used	No
Location must be used to specify the URI of a newly created resource	No
Caching should be encouraged	No *
Content-Type must be used	Yes
Custom HTTP headers must not be used to change the b**ehavior** of HTTP methods	Yes

*implementation specific

For the best practices regarding **client concerns**, the UMEI API gets a score of 66.67% for the baseline, and 100% for the potential case. However, it must be considered that the medium value of the baseline case is mainly caused by the reduced number of practices in this category (only three, as shown by Table 10). Depending on the implementation, Cross-Origin Resource Sharing (CORS) may be supported by the UMEI API to provide multi-origin read/write access from JavaScript.

Table 10 - Best practices for tackle client concerns

Category: Client concerns	Compliance
The query component of a URI should be used to support partial response	Yes
CORS should be supported to provide multi-origin read/write access from JavaScript	NS *
New URIs should be used to introduce new concepts	Yes

*implementation specific

For the last category, **versioning**, the UMEI API, as for the categories of request methods and representation design, also gets the maximum score of 100% for both the baseline and potential case. Table 11 shows that two practices were found to not be applicable to the UMEI API. However, in addition to the list of best practices for versioning, it was asked if the logic for handling the responses would change from one version to another, being the answer negative. In this case, [14] suggests, based on Apigee and Finnish Government's guidelines, to put the version on the HTTP header. This, which could be considered just a recommendation instead of a best practice, is something not covered by the current UMEI specification but that would depend on the specific implementation.

Table 11 - Best practices for API versioning

Category: Versioning	Compliance
Increments major version when incompatible API changes are made	Yes
Increment minor version when functionalities are added in a backwards-compatible way	N/A
Increment patch version when backwards compatible bug fixes are made	N/A
Increment draft version when changes are made during the review phase that are not related to production releases	Yes
API extensions do not take anything away	Yes
API extensions de not change processing rules	Yes
API extensions do not make optional things required	Yes
Anything added in the API extension is optional	Yes

6.1.3 Interim conclusions

To get an overall idea of the quality of the UMEI, for this analysis it has been considered that the outcome of the survey carried out by [11] about the importance of these practices perceived by eight expert developers. In that survey, the categories of URI design, HTTP request methods, error handling, and representation design are considered more relevant by developers. On the other hand, rules from the client concerns and metadata design categories were rated as less relevant. This means that, as long as an API performs reasonably well in the most relevant categories, a good level of understandability and reusability can be expected.

Results show that the UMEI API presents, in general, a good compliance of best practices of REST API design. UMEI follows all the rules for using HTTP request methods, versioning, and representation design. In certain implementations, the UMEI can also apply all the rules related to client concerns and error handling.

The category where the UMEI presents lower quality is metadata design, followed by the category of client concerns when considering the baseline case. Nevertheless, the best practices included in these two categories are the ones commonly considered by expert developers as the least relevant rules for API design [11]. In addition to this, the rules in these categories account for less than 12% of the list. Therefore, considering this, the scalability and replicability of UMEI are expected to not be strongly affected by the low scores in these categories.

As mentioned above, developers value more the best practices related to an appropriate URI design, a good use of HTTP request methods, good error handling, and a consistent representation design. These categories account for 77% of the best practices considered in this analysis. For these categories, as shown by Figure 15, the performance of the UMEI API is outstanding for the cases considered, so developers should not find many inconveniences when implementing UMEI according to its specification.

Regarding versioning, it was not considered by [11] in its survey. However, it can be considered a very relevant category to assure the scalability and replicability of an API; an API with a versioning system that follows the best practices will be easier to implement as it evolves. Results show that developers using the UMEI in future implementations should not have any problems to understand the functionality and usability of future versions of the API, given that all the best practices are followed and, during implementations, it can be even improved by putting the version on the HTTP headers. This sets a good basis for the replicability of the UMEI once the project finishes.

Despite the good performance of the UMEI regarding REST API design, it still has room for improvement concerning the seamless integration of additional actors and widening the scope in

terms of market processes covered. Regarding the former, the UMEI may present some limitations as it relies on a given data model and format for the flexibility services that may not be universal. Regarding the latter, it is relevant to point out that the UMEI, as it stands now, focuses exclusively on the trading process, leaving out other relevant processes that could be integrated, such as the registration of flexibility resources.

In order to address these limitations and facilitate replicability, future developments of the UMEI could provide compatibility with other ontologies that are currently being developed in the smart grid ecosystem. For example, one potentially relevant ontology is the Smart Applications REFerence (SAREF) ontology, which is used for the description of the features and capabilities of smart devices by different stakeholders (service providers, developers, manufacturers, etc.). In addition to this, SAREF also provides compatibility with the oneM2M Base ontology, for Internet of Things (IoT) devices. Although the description of these devices could get adapted to the UMEI, its additional compatibility with SAREF would facilitate the registration and prequalification of smart devices and their overall integration in the market processes where UMEI is implemented.

In general, the scalability and replicability of UMEI will be good, based on its expected good understandability and reusability by developers, which are related to the application of most of the best practices enumerated in the specialized literature on the topic. This good understandability and reusability could be used to expand the UMEI, in a structured way, to provide compatibility with standardized ontologies. This would facilitate the integration of new actors in the market processes and further improve the scalability and replicability of the UMEI.

7 References

[1] EUniversal, "Deliverable: D2.6 - UMEI API management and documentation," 2022.

[2] C. Silva, D. Silva, G. Milzer, N. Sætre, Ø. D. Eide, P. Crucifix, L. Debroux, A. Debray, C. Dumont, G. Marzano and M. Kaffash, "D2.5 UMEI prototype," 2021.

[3] L. Fonseca, D. Silva, G. Milzer, N. Sætre, P. Crucifix, L. Debroux, N. Metivier and M. Kaffash, "D2.4 UMEI API functional specification of DSO Interface for standardized market platforms Interaction," 2021.

[4] EUniversal, "EUniversal D2.1 - Grid flexibility services definition," 2021.

[5] ENTSO-E, "INTERRFACE Roadmap," 2022.

[6] EUniversal, " D7.4 - Portuguese Demonstrator — Demonstration of the UMEI concept in the management of market driven flexibility," 2023.

[7] Wallarm, "What Is An API Authentication?," 2023. [Online]. Available: https://www.wallarm.com/what/what-is-an-api-authentication.

[8] S. Kotstein and J. Bogner, "Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts," Communications in Computer and Information Science, 2021.

[9] M. Stowe, Undisturbed REST: A guide to designing the perfect API, MuleSoft, 2015.

[10] EUniversal, "D10.4 - Scalability and Replicability analysis of the EUniversal solutions," 2023.

[11] F. Petrillo, P. Merle, N. Moha and Y. G. Guéhéneuc, ""Are REST APIs for cloud computing well-designed? An exploratory study,"," in *Springer Int. Publ*, vol. 9936 LNCS, Springer Int. Publ, 2016, p. 157–170.

[12] M. Masse, REST API Design Rulebook, O'Reilly Media, Inc, 2011.

[13] C. Rodriguez, M. Baez, D. Florian, F. Casati and J. Trabucco, REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices, Lecture Notes in Computer Science, 2016.

[14] L. Murphy, T. Alliyu, A. Macvean, M. B. Kery and B. A. Myers, "Preliminary Analysis of REST API Style Guidelines," *PLATEAU'17 Workshop on Evaluation and Usability of Programming Languages and Tools,* p. 1–9, 2017.