D1.3 - SPECIFICATIONS FOR INTEROPERABLE SOFTWARE TOOLS

INTEROPERABLE CLIENT/SERVER AND LEGACY SYSTEMS PROTOCOL CONVERTER

WP1 - Requirements, Use Cases, Specifications

T1.3 - Specifications For Interoperable Software Tools

Submission date: 31 Oct 2023

Project Acronym	INTERSTORE
Call	HORIZON-CL5-2022-D3-01
Grant Agreement N°	101096511
Project Start Date	01-01-2023
Project End Date	31-12-2025
Duration	36 months

INFORMATION

Written By	Sunesis (SUN) CyberGrid (CYG) RWTH Aachen University (RWTH)	2023-10-03 2023-10-03
Checked by	Alexandre Lucas (InescTec)	2023-10-18
Reviewed by	Peter Nemcek, Andraz Andolsek (CYG) Nithin Manuel (RWTH)	2023-10-16
Approved by	Antonello Monti (RWTH) – Project Coordinator Francesco Guaraldi (ENX)	2023-10-27
Status	Final	2023-10-27

DISSEMINATION LEVEL

CO	Confidential	
CL	Classified	
PU	Public	X

VERSIONS

Date	Version	Comment
08-06-23	0.1	Outline - draft
12-06-23	0.2	Improved outline
15-06-23	0.3	Improved outline after partner meeting
26-06-23	0.4	Finished Ch 2
19-07-23	0.5	Finished Ch 4
07-08-23	0.6	Finished Ch 5
21-08-23	0.7	Finished Ch 6
04-09-23	0.8	Improvements of all chapters, Ch 7 and 8
18-09-23	0.9	Overall improvements, Chapter 9
26-09-23	0.91	Corrections, improvements
03-10-23	0.95	Corrections, improvements
03-10-23	0.96	Updated Use cases chapter

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

05-10-23	1.00 draft	Minor updates – ready for internal review
25-10-23	1.00 final	Final version

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

ACKNOWLEDGEMENT


InterSTORE is a EU-funded project that has received funding from the European Union's Horizon Research and Innovation Programme under Grant Agreement N. 101096511.

DISCLAIMER

ABBREVIATIONS AND ACRONYMS

EMS	Energy Management System
NATS	Neural Autonomic Transport System
msg	Message
MQTT	Message Queuing Telemetry Transport
XML	Extensible Markup Language
JSON	JavaScript Object Notation
TLS	Transport Layer Security
JWT	JSON web token
TCP	Transmission Control Protocol
QoS	Quality of Service
YAML	Yet Another Markup Language
LPC	Legacy Protocol Converter
API	Application Programming Interface
JAR	Java ARchive
JVM	Java Virtual Machine
LTS	Long Term Support
SSL	Secure Sockets Layer
MIT	Massachusetts Institute of Technology
GPL	General Public License
BSD	Berkeley Source Distribution
ASF	Apache Software Foundation
FSF	Free Software Foundation
HESS	Hybrid Energy Storage Systems

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

TABLE OF CONTENTS

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

EXECUTIVE SUMMARY

This document presents the specification of the interoperable open-source software tools to integrate hybrid energy storage systems (HESS) for the Horizon Europe project INTERSTORE. The tools consist of two components: an interoperable client/server for distributed energy storage and a legacy systems protocol converter. The main objective of these tools is to provide open source, out-of-the-box support for IEEE2030.5 communication between devices of distributed energy sources (DER) including energy storage systems (ESS) and energy management systems (EMS), as well as support for next generation NATS messaging as a communication protocol. The tools will also provide support for IEEE2030.5 messages in both XML and JSON formats, as well as other protocols such as MQTT and ModBus.

Providing support for the IEEE2030.5 standard together with next-generation cloud-native messaging architecture NATS is important not only for preparing the building blocks for seamless integration and interoperability of devices and systems, standard compliance, but also for improving technical aspects, such as reliability, scalability and capability to work with computer cloud environments and cloud-native software. Another important aspect is to provide support for IEEE2030.5 not only in the original XML format, but also in the widely used JSON format which supersedes the XML format.

The document defines the detailed specifications for each component, interoperable client/server for distributed energy storage and legacy systems protocol converter. This includes the architectural diagrams, communication scenarios, sequence diagrams, message exchange patterns, protocols, message schemas and structures, fault and exception signalling, message transformation and configuration framework, software architecture, and use cases. The document also describes how the software components will be delivered (as Docker containers, pre-built Java JAR archives, or custom built for specific use cases) and how they will be available on GitHub as open-source projects.

The document is intended for developers, researchers, and practitioners who are interested in developing, testing, deploying, or using the interoperable open-source software tools to integrate DER. The document assumes that the readers have some basic knowledge of DER, ESS, EMS, the IEEE2030.5 standard, the NATS messaging technology, and microservice architecture. The document is organized as follows:

- Section 1: Introduction. This section provides the background, motivation, objectives, and scope of the document.
- Section 2: Architecture and Development Methodology. This section describes the architecture and development methodology of the interoperable open-source software tools to integrate HESS, including the interoperable client/server and the legacy systems protocol converter components.
- Section 3: Use case scenarios. This section describes some use case scenarios to illustrate how the software tools will be used in various scenarios that are part of this project, including HESStec and HyDEMS, Capwatt, CyberGrid and CyberNoc, FZJ and ICT platform, Enel-X and VPP Flex platform.
- Section 4: Supported protocols. This section describes the supported protocols on the device (client) side and on the EMS (server) side, including ModBus, MQTT, NATS, and IEEE2030.5. It also provides a description of NATS as a next generation messaging protocol.
- Section 5: Message structures and schemas. This section describes the message structures and schemas for IEEE2030.5 messages in both XML and JSON formats, as well as the fault and exception messages. It also describes the supported message structures and formats in the client/server and the legacy protocol converter components.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

- Section 6: Message transformation and configuration. This section describes the message transformation and configuration framework that allows simple and fully configurable transformation of messages between ModBus, MQTT, NATS, IEEE2030.5 XML and JSON messages.
- Section 7: Software architecture and Development Methodology. This section describes the software architecture of the components, following the microservice architectural patterns. It also describes the build and deployment options, container support on different platforms, programming language and NATS libraries, security, and development methodology of the project.
- Sections 8 and 9: Software and test procedures requirements and proposed methodology for testing the interoperability software. These sections provide a specification of various testing procedures.
- Section 10: Open-source access on GitHub. This section provides information about the GitHub repository, the open-source license models, and the development methodology of the project.
- Section 11: Conclusion. This section summarizes the main points and outcomes of the document.

This document provides clear and comprehensive specifications of the interoperable open-source software tools. The software development will be carried out within WP 2, tasks T2.1, T2.2 and T2.3.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

1 Introduction

In this deliverable, which has been created within WP1, Task 1.3, we have specified the tools to replicate, adapt, and improve interoperable open-source software to integrate distributed energy sources (DER) devices with energy management systems (EMS). This specification addresses:

i) interoperable client/server for distributed energy storage,
ii) legacy systems protocol converter,
iii) testing procedures.

The objectives of the interoperable client/server and the legacy systems protocol converter are to:

- Provide an open source, out-of-the-box support for IEEE2030.5 communication between devices and EMS systems. The objective is to provide support for IEEE2030.5 messages in original XML format, as well as IEEE2030.5 in JSON format.
- Provide support for next generation NATS messaging as a communication protocol between devices and EMS systems, superseding other communication mechanisms (such as REST over HTTP) and enabling message-driven, loosely coupled and scalable communication platform.
- Provide a reference implementation as an open-source project, available on GitHub.

In this specification we have presented the detailed requirements for the interoperable client/server (we will use the wording *client/server* in the rest of the document) and the legacy systems protocol converter (we will use the wording *legacy protocol converter* in the rest of the document). For each of them, we have defined the architectural diagrams, communication scenarios and specified sequence diagrams to show the interaction of messages.

The specification also defines the message exchange patterns, including one-way, request/response and data streaming patterns, together with correlation, delivery options, subjects and registration and authentication of devices. It specifies the protocols supported on the device side and on the EMS side, including MQTT, ModBus and NATS.

The specification defines the message schemas and structures. Both client/server and legacy protocol converter will support the complete set of IEEE2030.5 messages and types – 321 messages and types, as defined by the IEEE2030.5 XML schema definitions. Also, fault and exception signalling has been defined.

Furthermore, within WP2, we will define the corresponding JSON schemas for all IEEE2030.5 data elements and types, which will enable using IEEE2030.5 data formats with JSON also (in addition to XML). As JSON is becoming more and more popular in software architectures, this will be an important step towards interoperability.

This specification also describes another very important building block for the legacy protocol converter, the message transformation and configuration framework. This framework will allow simple and fully configurable transformation of messages between ModBus, MQTT, NATS, IEEE2030.5 XML and JSON messages.

Finally, this specification also provides a description of software architecture, which will follow the microservice architectural patterns and describes how the software components will be delivered (as Docker containers, pre-built Java JAR archives, or custom built for specific use cases). As already mentioned, all software artifacts will be available on GitHub.

This specification also provides a description of testing procedures, testing methodologies and description of various testing scenarios. Furthermore, it provides brief description of use

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

cases to define, how the software components (client/server and legacy protocol converter) will be used in various use cases, which are part of InterSTORE project.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

2 Architecture and Development Methodology

In this chapter, the architecture of the client/server and the legacy protocol converter from the perspective of message exchange between the EMS (Energy Management System) and the various devices will be described. The different communication scenarios over NATS will be shown. Additionally, the development methodology will be outlined.

The main objective of the client/server and the legacy protocol converter is to enable communication between the device (client) and the EMS system (Server) over NATS massaging communication protocol and use standard IEEE2030.5 elements to enable interoperability, scalability, loose coupling, real-time data exchange, secure and reliable communication and overcome the typical problems of existing protocols and formats, such as REST over HTTP, MQTT and ModBus.

The client/server component is supposed to be integrated within the client device and the EMS systems. The legacy protocol converter is a separate software component that will provide a bridge between the devices using legacy protocols, such as ModBus and MQTT, and will also provide support for message payload transformation to IEE2030.5 using XML or JSON formats.

In the next sections we first describe the client/server and then the legacy protocol converter.

2.1 Client/server

2.1.1 Architecture

The architecture of the client/server tool can be seen in Figure 1. Devices act as clients; EMSs act as servers. Client or server software can be included as a library in existing programs, or it can run as a separate microservice. It is also possible for multiple EMSs to be present. Each EMS can communicate with other EMSs and each device can communicate with multiple EMSs.

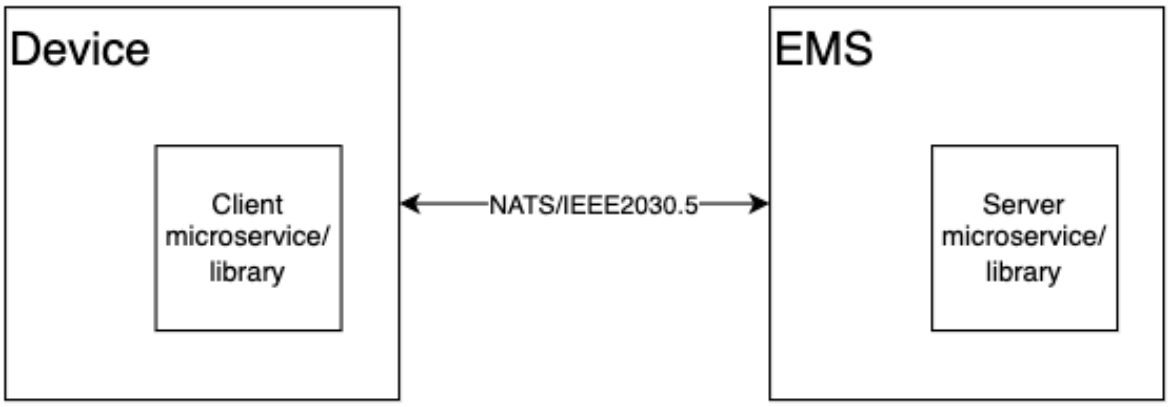

Figure 1: Client/server architecture.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

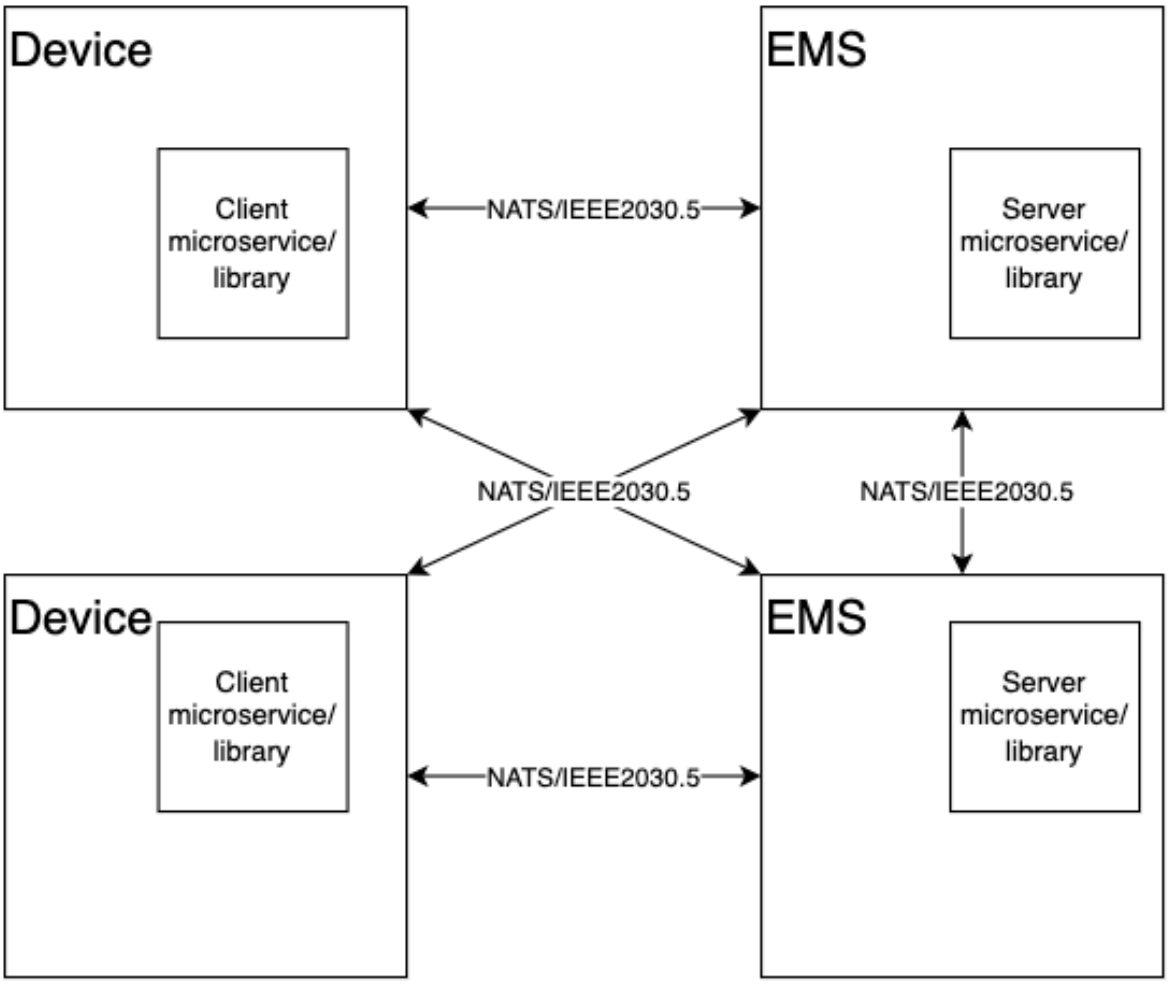

Figure 2: Client/server architecture where multiple devices and EMSs are present.

2.1.2 Communication scenarios

In the client/server scenario, the Energy Management Systems (EMS) functions as server, while devices serve as clients. Figure 3 provides a broad overview of the two-way communication dynamics between clients (devices) and the server (EMS). It is possible that there are multiple EMSs present and for EMSs to communicate between each other using the NATS/IEEE2030.5, so EMSs can act both as servers and as clients. This way we can achieve communication between EMS systems and their interoperability using IEEE2030.5 standard.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

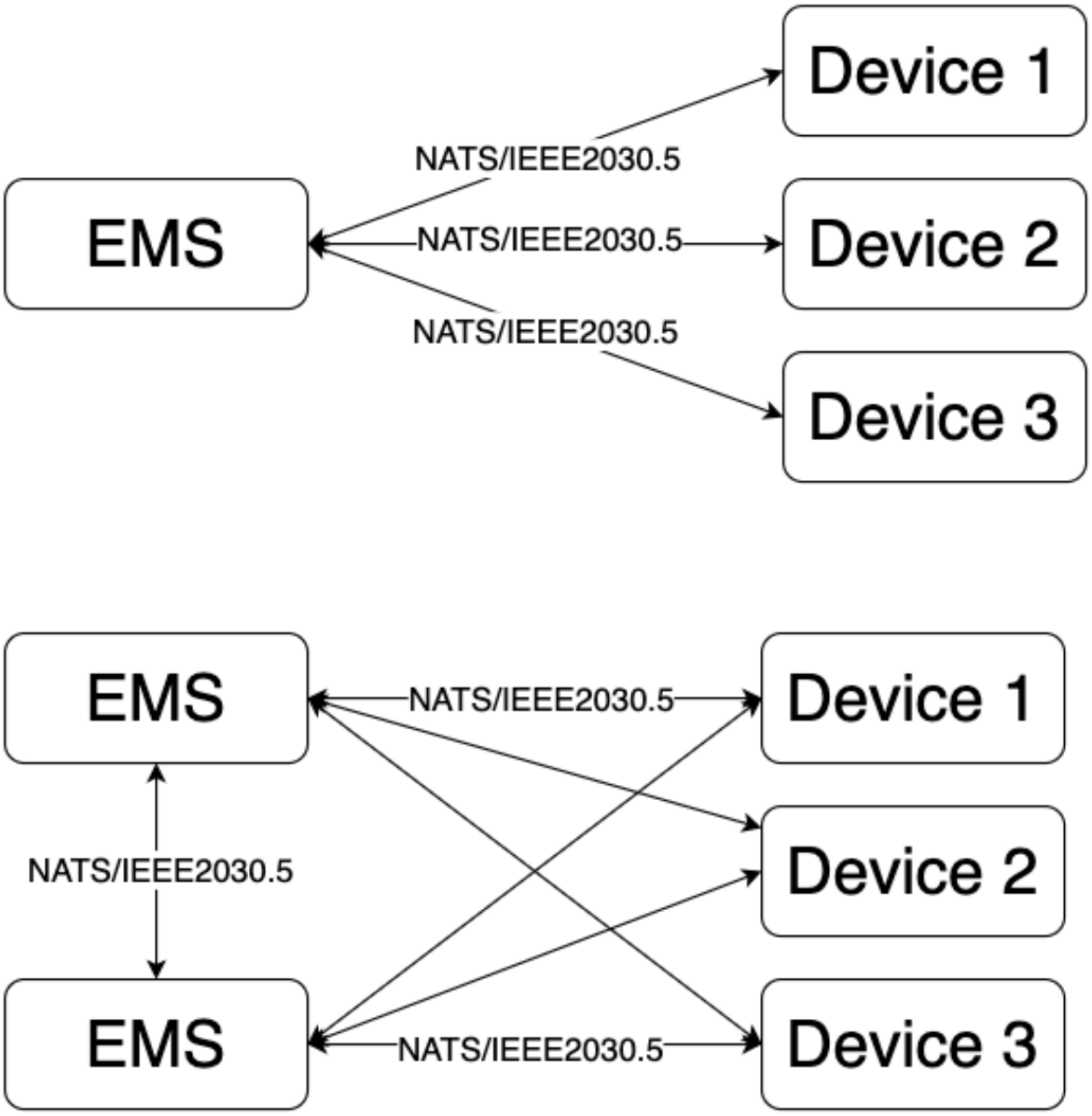

Figure 3: Overview of the client/server communication.

For a more comprehensive understanding of the communication between clients and the server, detailed descriptions are available in the subsequent figures.

The subject-based NATS messaging follows a publish-subscribe pattern, wherein devices both subscribe to and publish messages to subjects. Devices possess the flexibility to subscribe to multiple subjects, encompassing shared subjects or unique subjects dedicated to each device. It is also possible for multiple EMSs to subscribe to the same subject where they can publish-subscribe to device messages or messages from other EMSs.

Figure 4 shows the option, where devices use the same subject to communicate with the EMS. All communication is conducted through this subject.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

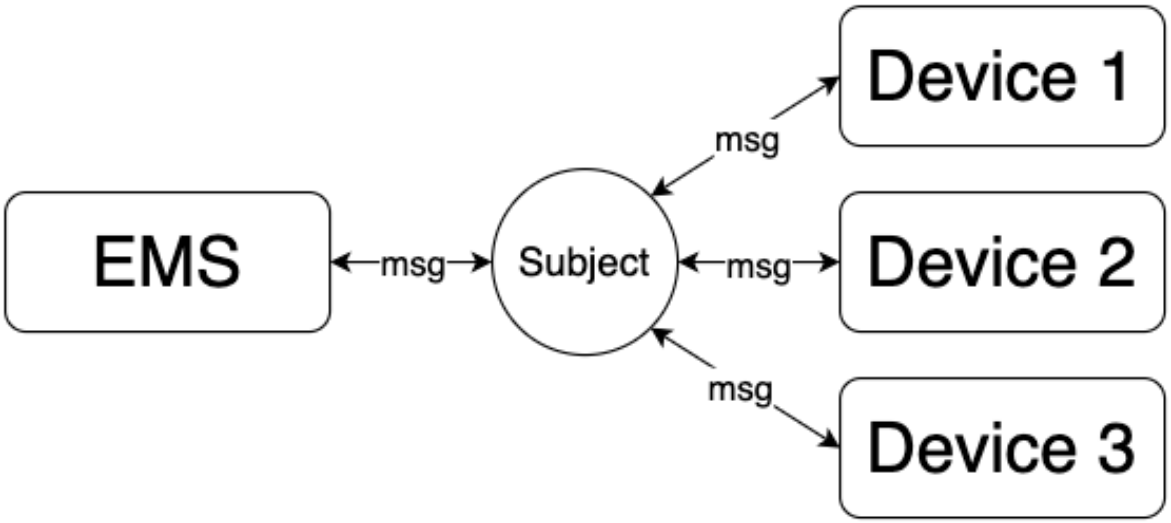

Figure 4: Publish-subscribe pattern in client/server communication using shared subject.

Figure 5 shows the option, where each device has its own subject. All communication from the EMS to devices is through the designated subject.

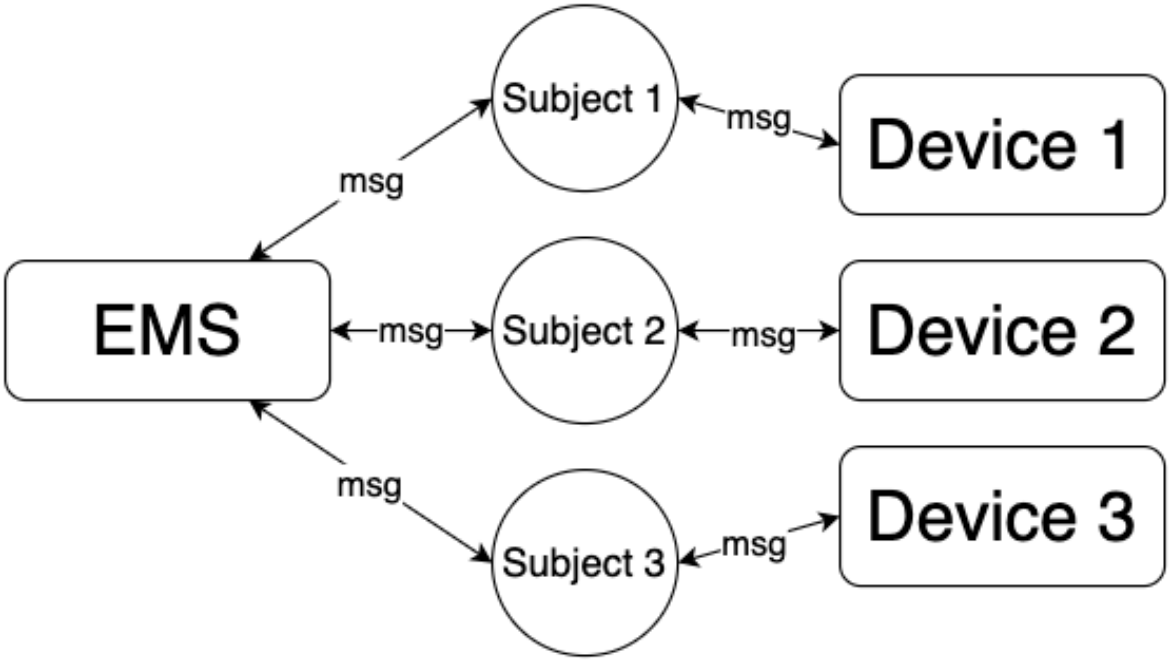

Figure 5: Publish-subscribe pattern in client/server communication using one subject per device.

Figure 6 shows the option, where multiple EMSs and devices are present. The main goal of this figure is to show that there can be multiple EMSs present in the network and they can subscribe to the same subjects for devices or to the subjects used only by EMSs to communicate between each other.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

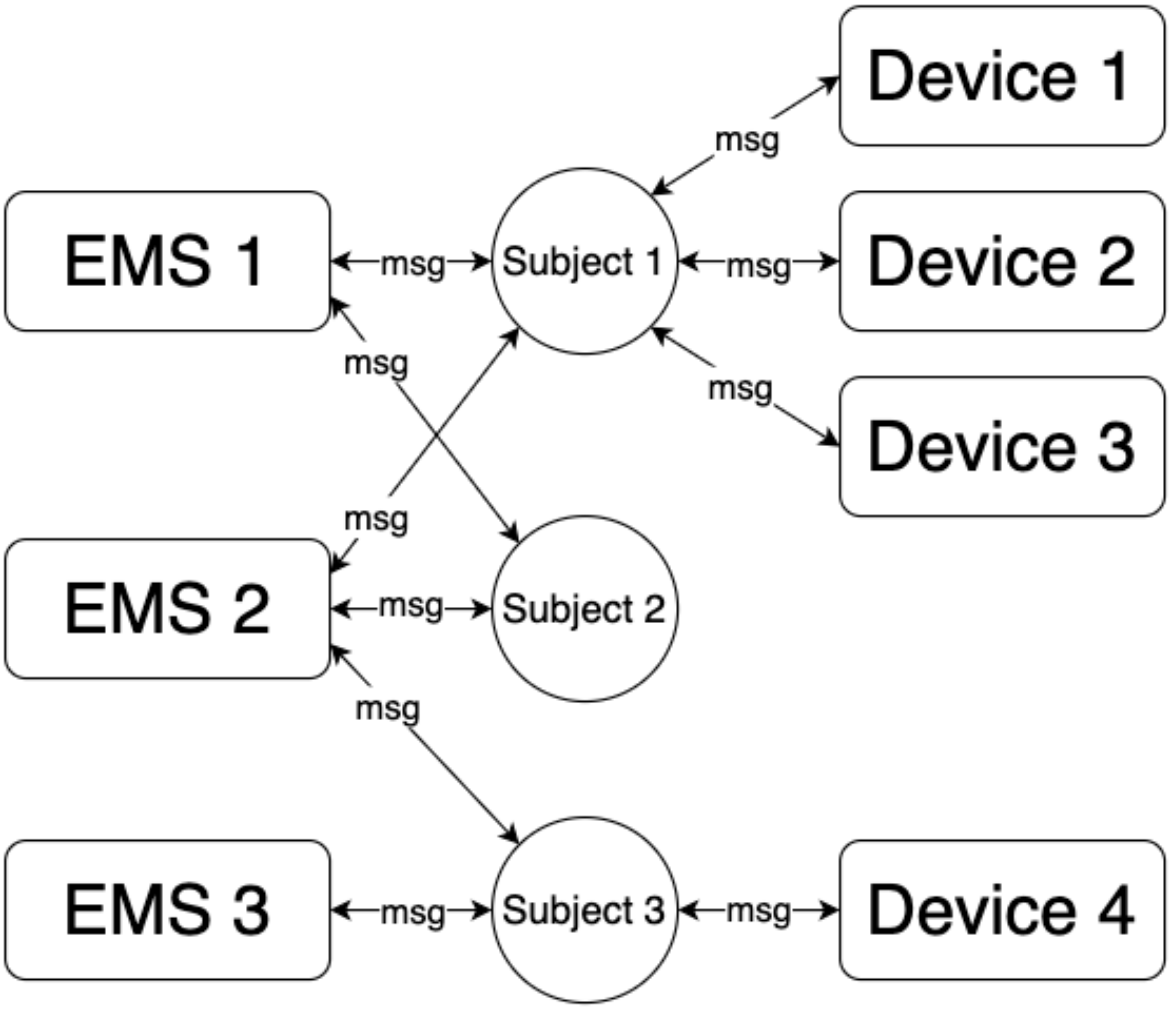

Figure 6: Publish-subscribe pattern in client/server communication with multiple EMSs and devices.

In the request-reply pattern, the server initiates a request to a specific subject and subsequently receives responses from devices. Devices can either reply exclusively to the sender or transmit responses to a subject designated within the request (Figure 7). It is also possible for the messages of the same subject to be replied through a different reply subject.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

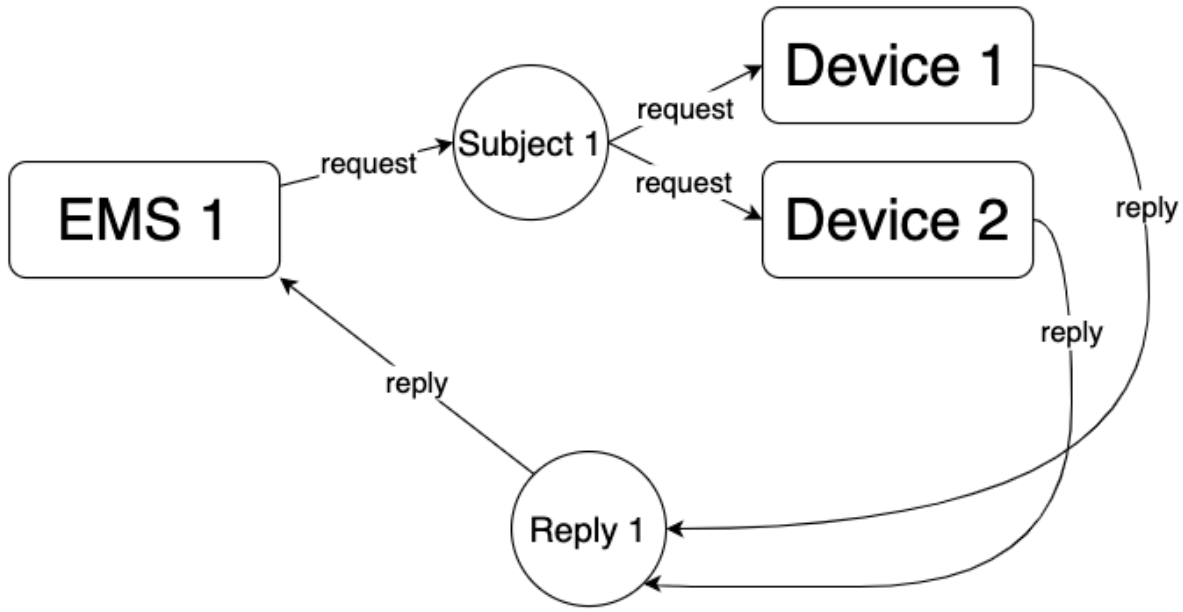

Figure 7: Request-reply pattern in client/server communication.

EMSs can also request those replies from other EMSs. For example, in the Figure 8, the EMS 2 can request a reply through Subject 2. This means that the request can be for Device 2 or for EMS 1. EMS 2 will receive a reply through Reply 2.

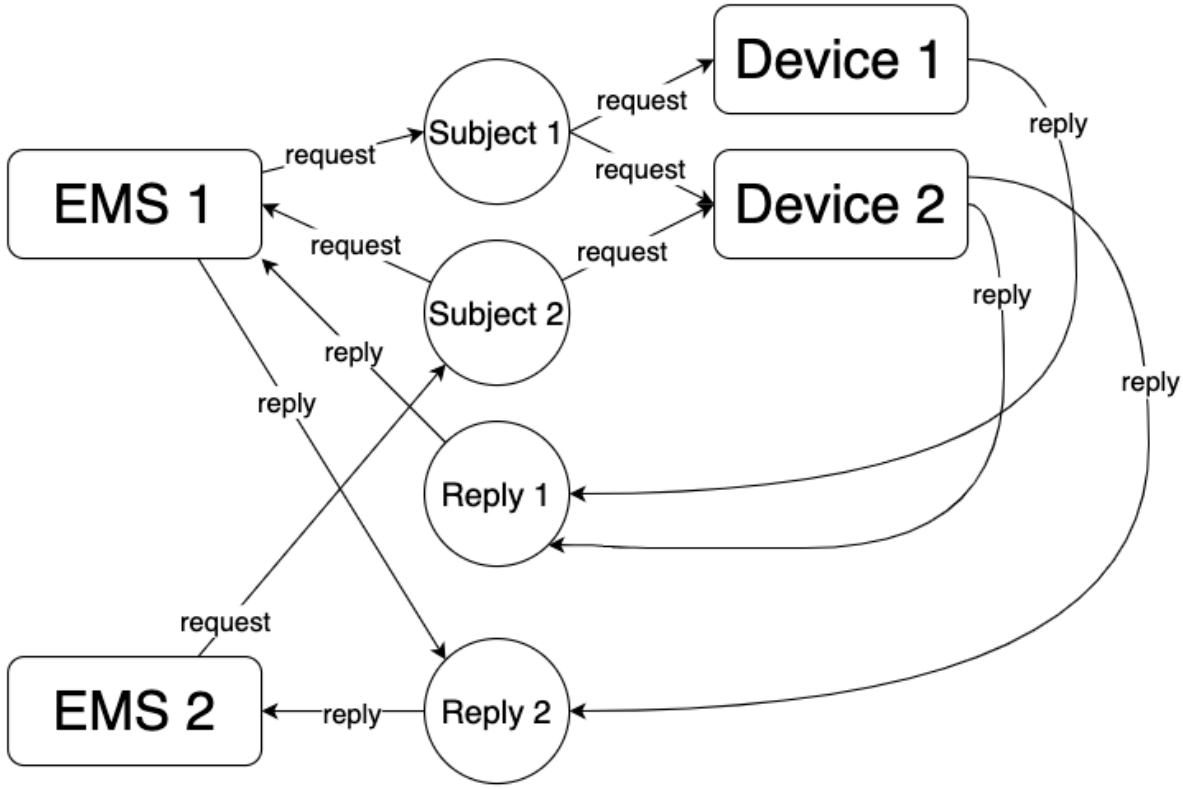

Figure 8: Request-reply pattern in client/server communication with multiple EMSs

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

An additional scenario involves the server (EMS) issuing a request to a subject and subsequently receiving a continuous stream of responses from the client (device), thereby illustrating the request-stream pattern (Figure 9). Examples of such scenarios are real-time or near-real-time metering data, temperature readings, multimedia data, etc. Multiple EMSs can request a stream over the same subject.

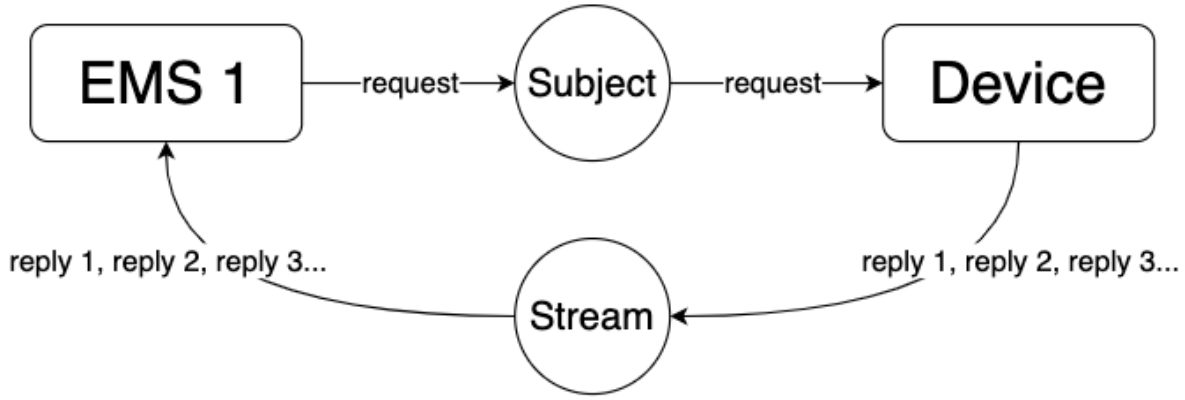

Figure 9: Request-stream pattern in client/server communication.

Figure 10 showcases the scenario where there is an additional EMS present. The role of this EMS 2 in the example is just to request streams from EMS 1.

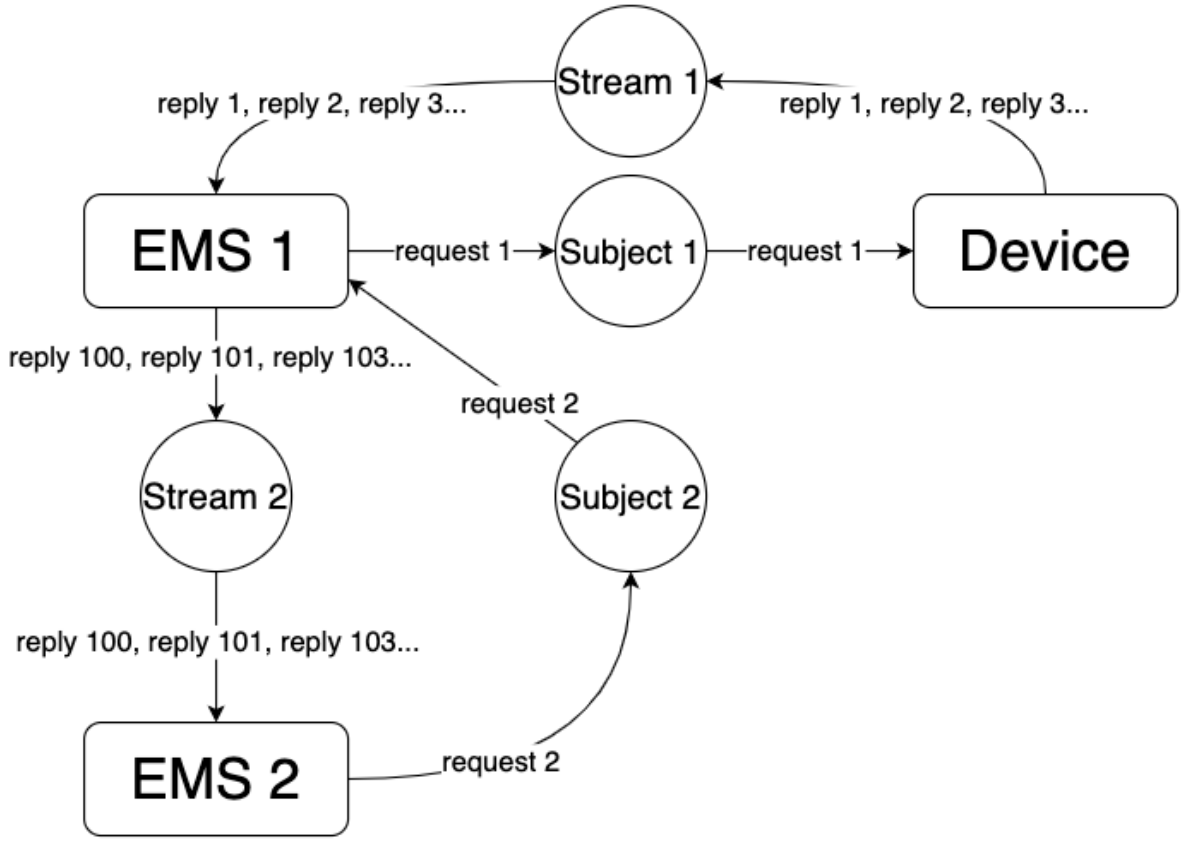

Figure 10: Request-stream pattern in client/server communication where another EMS is present.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

2.1.3 Sequence diagrams

Diagram on Figure 11 shows the authentication and registration flow of a client device with the EMS acting as a server. It also showcases the subscription to the subject on the client-side and the publishing of a message to the subject.

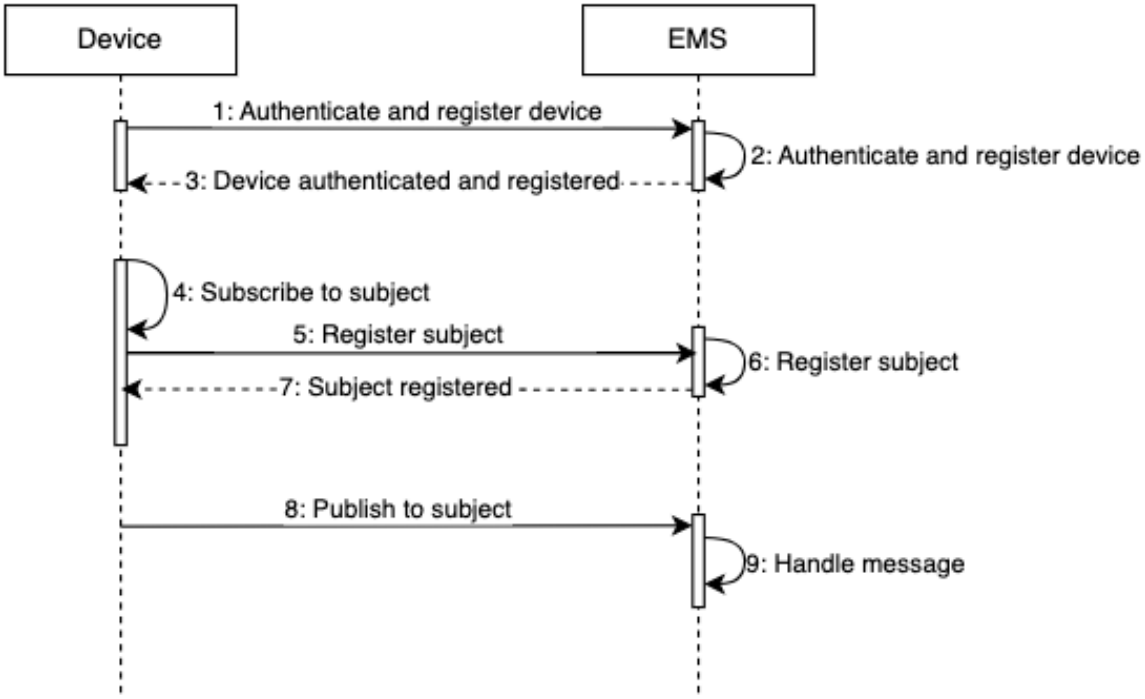

Figure 11: Sequence diagram showing authentication and registration of client device and publish-subscribe pattern.

Each of the following sequence diagrams assumes that the device is already authenticated and registered. Authentication and registration are described in Section 2.3.

The Diagram from Figure 12 shows the communication sequence when the client and the server are using the request-reply option.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

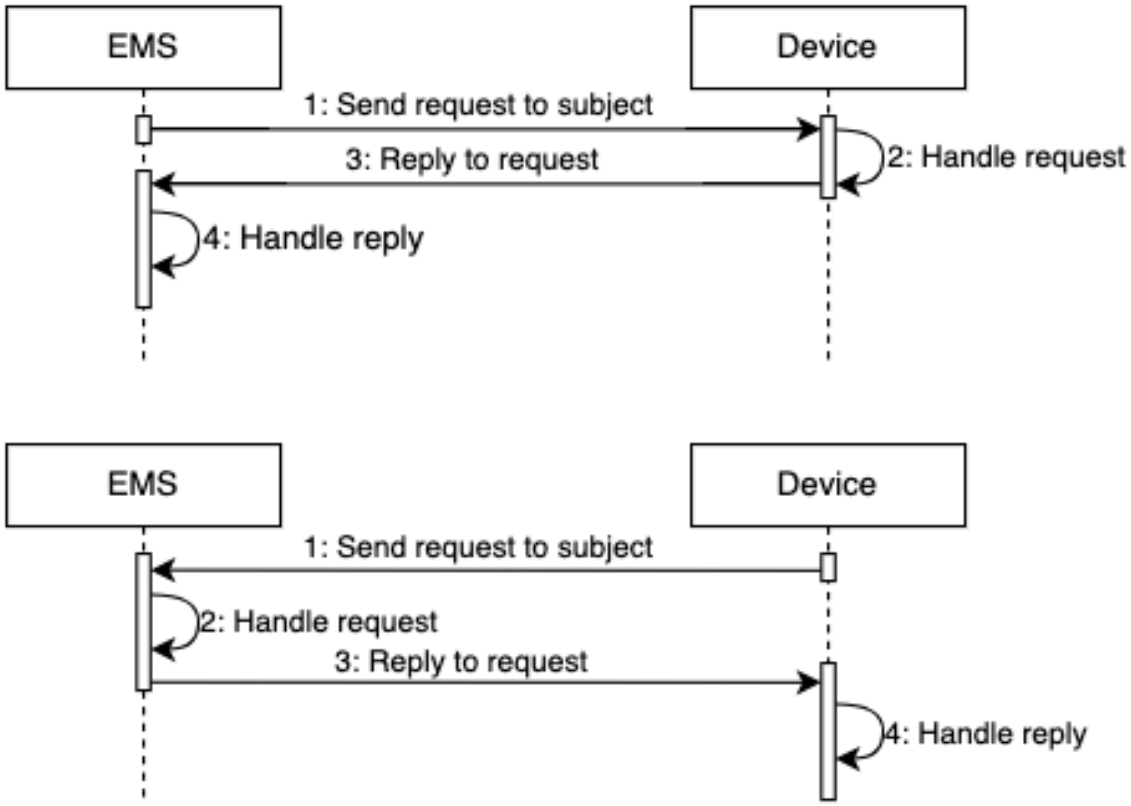

Figure 12: Sequence diagram showing request-reply pattern between client and server.

The Diagram from Figure 13 shows the communication between client and server when the server sends a request for streaming communication to the client. The Client sends continuous messages until it receives a stop message from the server, or some predefined condition from request holds such as elapsed time.

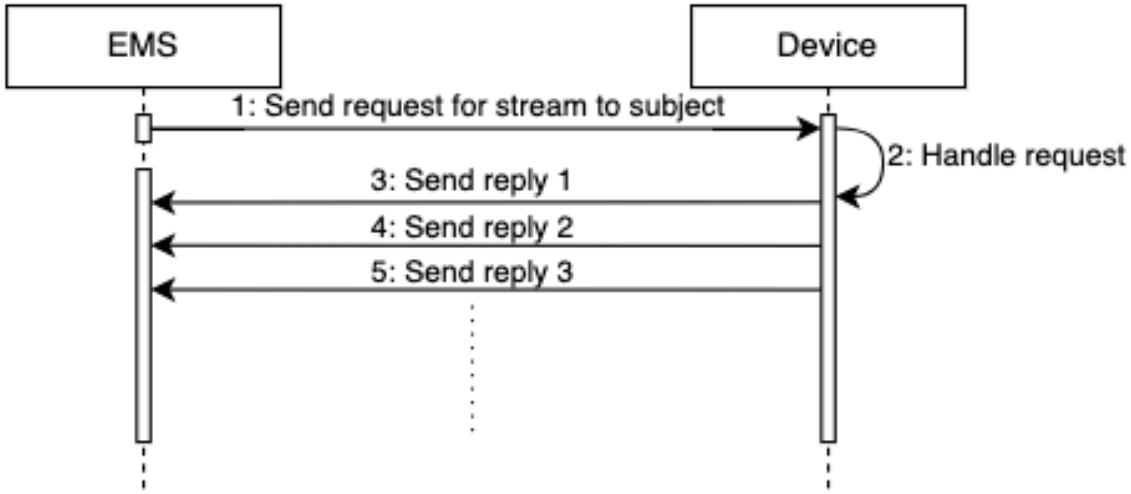

Figure 13: Sequence diagram showing request-stream pattern between client and server.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

2.1 Legacy protocol converter

2.1.1 Architecture

The Legacy protocol converter will act as a middleman between energy management systems and devices. The role of the legacy protocol converter is to convert messages from legacy protocols, used by devices, such as Modbus and MQTT[1] to protocols used by EMS, primarily NATS protocols Legacy protocol converter will also implement schema transformations, which will allow conversion of messages between XML, JSON and Modbus. On the EMS side, IEEE2030.5 will be fully supported with the possibility to use IEEE2030.5 messages as XML or JSON.

Devices have the role of clients; EMSs have the role of servers.

An overview of the architecture of the legacy protocol converter can be seen in Figure 14, Figure 15 and Figure 16.

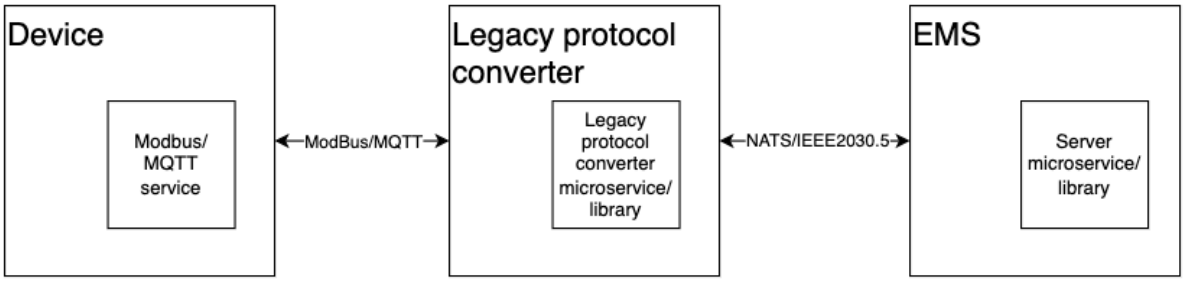

Figure 14: Architecture of legacy protocol converter.

[1] Support for other legacy protocols can be added by the open-source community.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

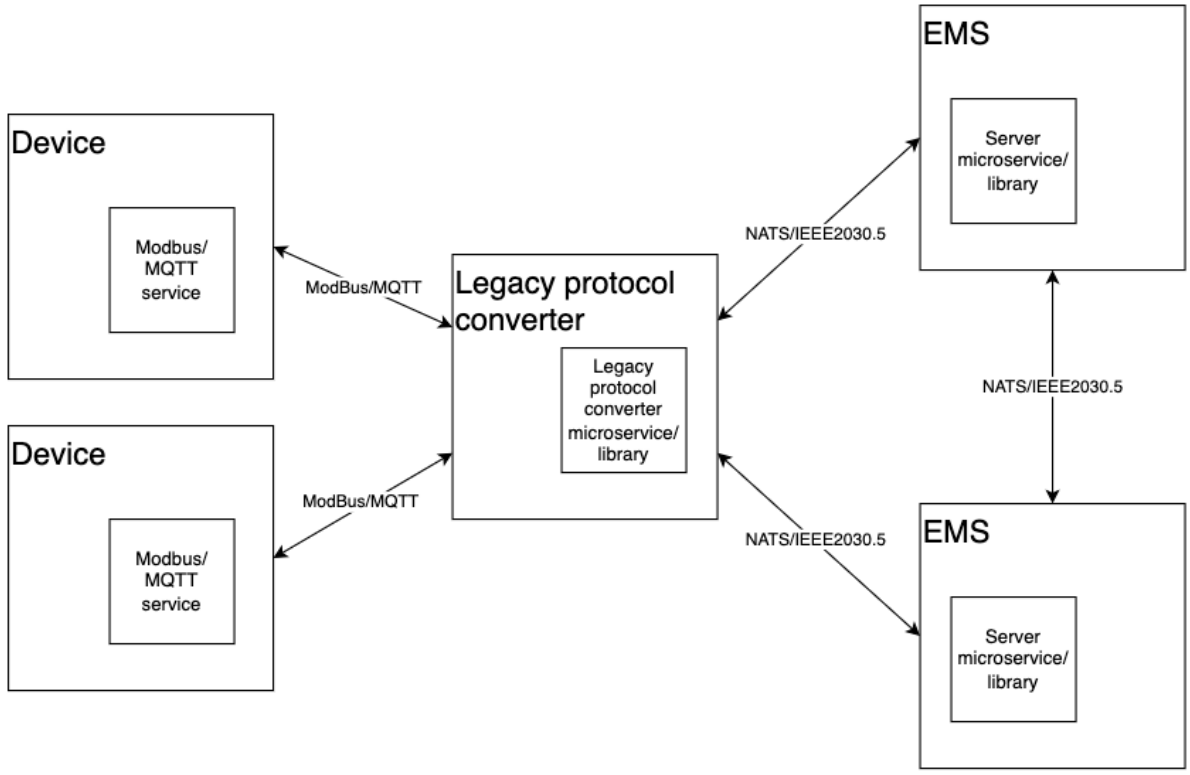

Figure 15: Architecture of legacy protocol converter with multiple devices and EMSs.

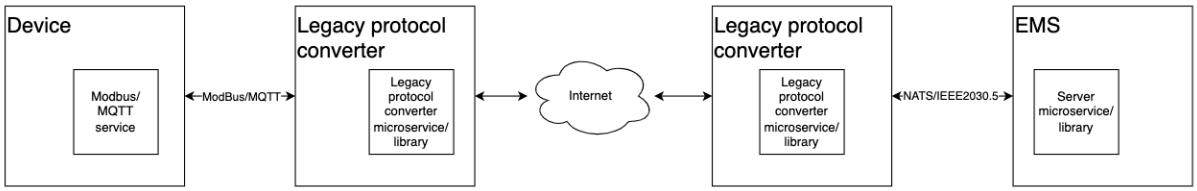

Figure 16: Architecture of legacy protocol converter where two legacy protocol converters communicate over internet.

2.1.2 Communication scenarios

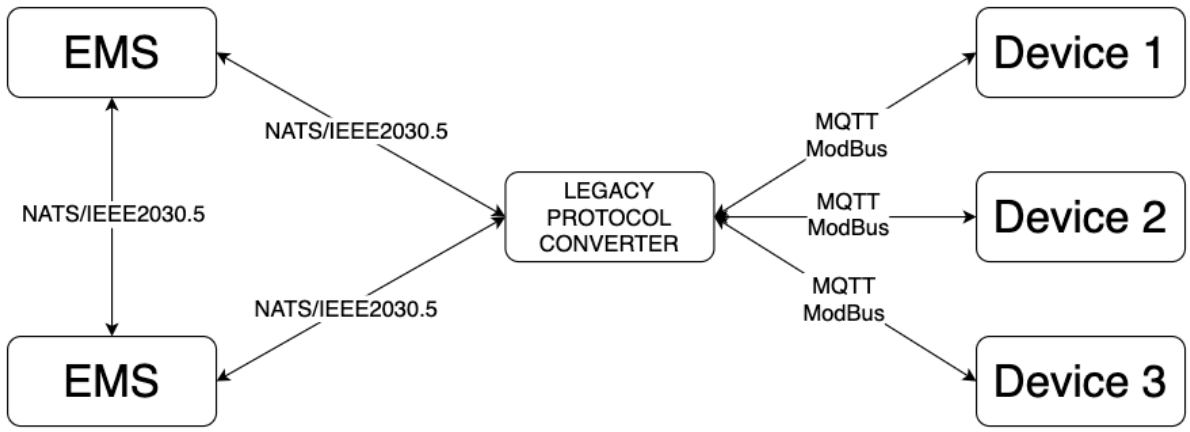

Figure 17: Overview of the communication between EMS, legacy protocol converter and devices.

Similarly to the client/server architecture, the legacy protocol converter will also support publish-subscribe, request-reply and the request-stream patterns.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

In Figure 18, communication, using publish/subscribe pattern, is shown between EMS, legacy protocol converter and devices. Like client/server architecture, here devices can also listen to shared subjects or to individual subjects. EMSs can also communicate over the same subject.

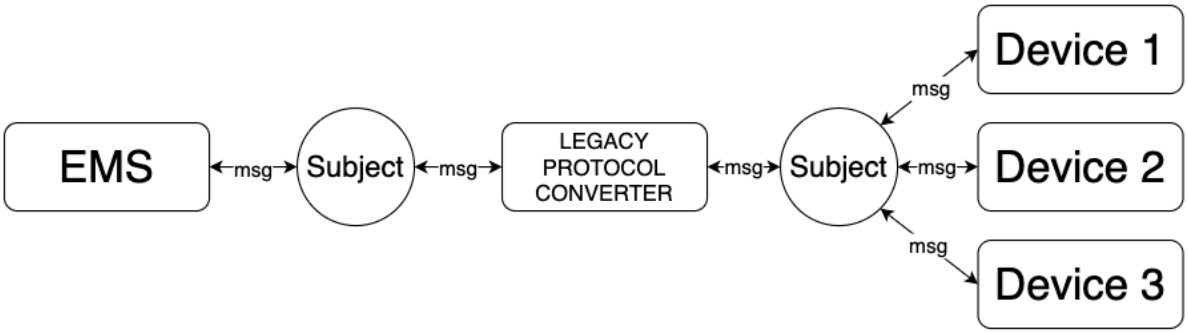

Figure 18: Publish/subscribe pattern using legacy protocol converter where devices share the same subject.

Figure 19 showcases the scenario where each device has its own subject. Communication with each device is conducted through the designated subject.

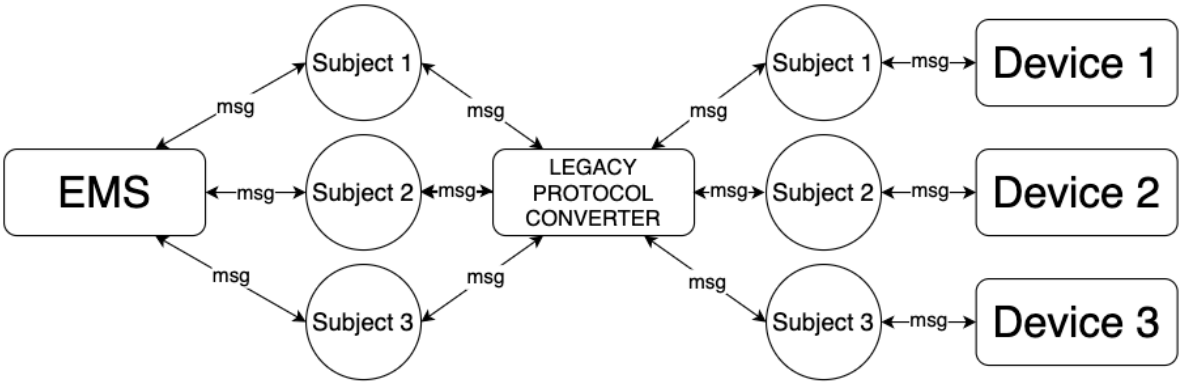

Figure 19: Publish/subscribe pattern using legacy protocol converter where each device has its own subject.

Figure 20 showcases the scenario where two legacy protocol converters, multiple EMSs and devices are present. The communication with other EMSs is through the subject to which all EMSs are subscribed to.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

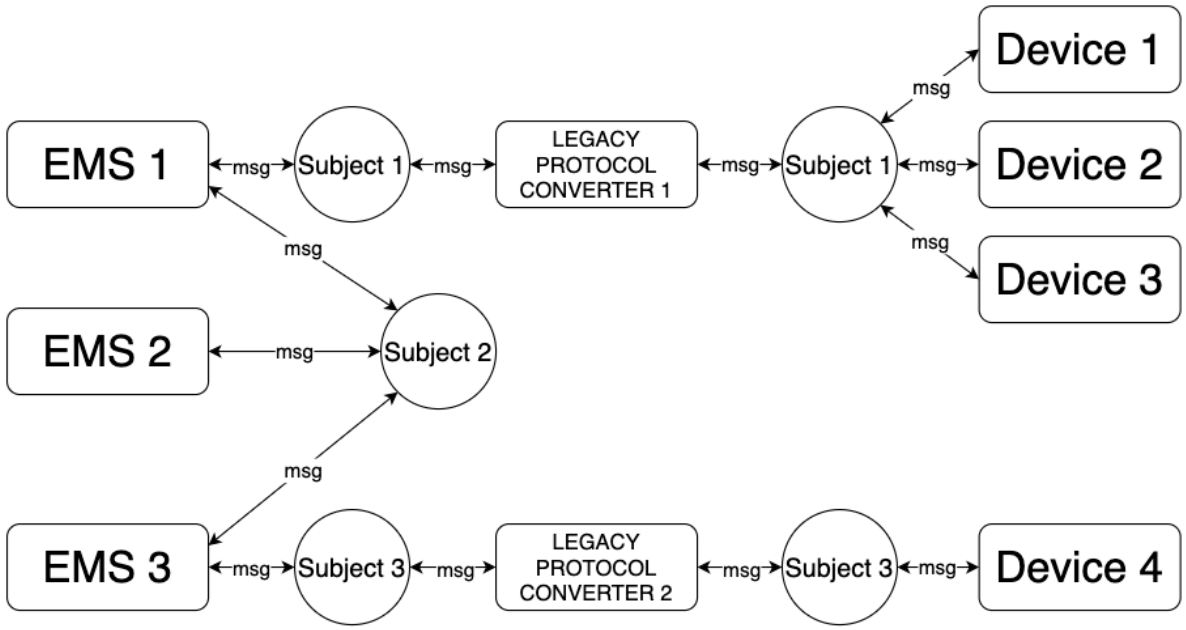

Figure 20: Publish/subscribe pattern using two legacy protocol converters where multiple EMSs and devices are present.

In the request-reply pattern, the server initiates a request to a specific subject and subsequently receives responses from devices. Devices can either reply exclusively to the sender or transmit responses to a subject designated within the request. The EMS communicates with the legacy protocol converter and devices also communicate with legacy protocol converter. This can be seen in Figure 21.

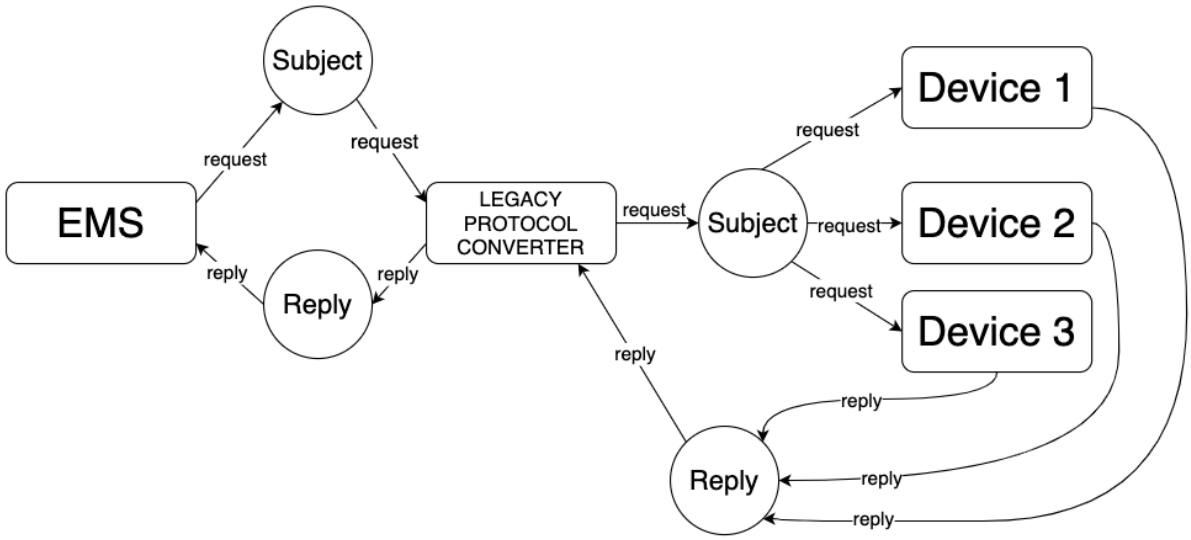

Figure 21: Request-reply pattern using legacy protocol converter.

Figure 22 shows the scenario where there are two EMSs present. EMS 1 requests replies through the subject Reply 1, EMS 2 requests replies through the subject Reply 2.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

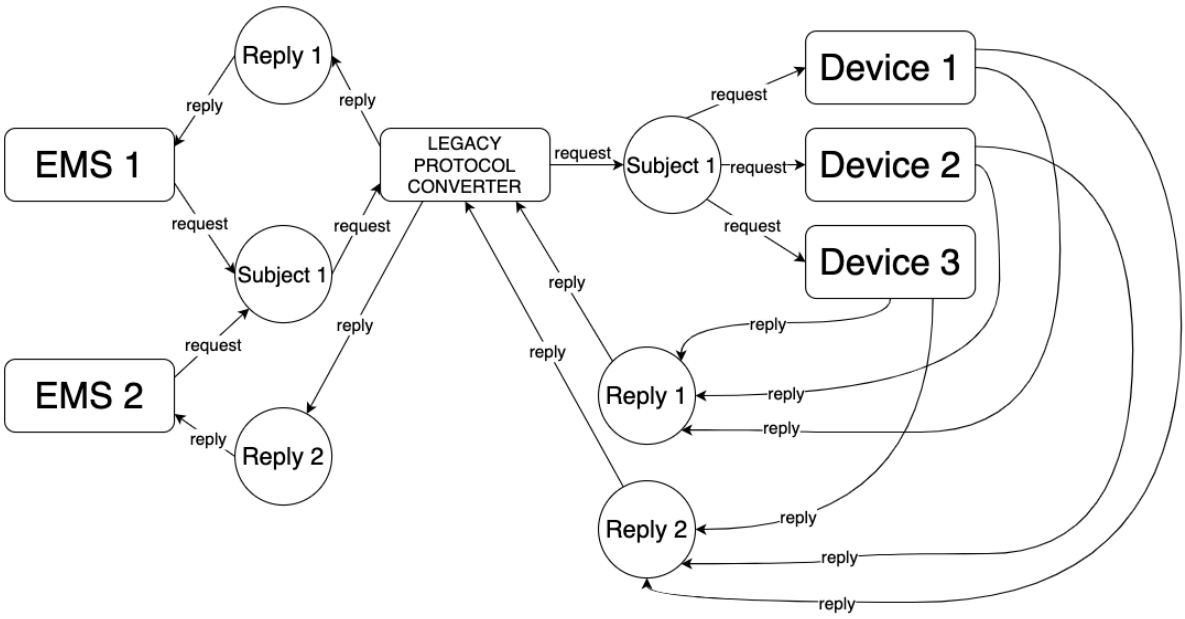

Figure 22: Request-reply pattern using legacy protocol converter where two EMSs are present.

An additional scenario involves the server issuing a request to a subject and subsequently receiving a continuous stream of responses from the devices, thereby illustrating the request-stream pattern. EMS sends a request to legacy protocol converter which forwards this request to devices, and devices send continuous stream of replies to the EMS through legacy protocol converter. This can be seen in Figure 23.

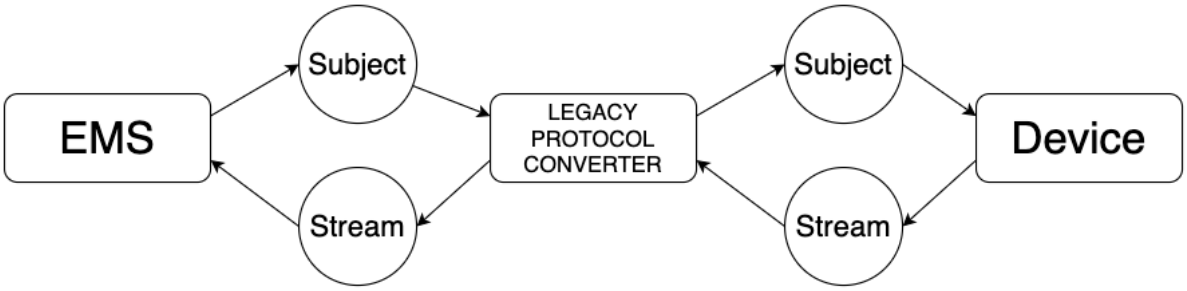

Figure 23: Request/stream pattern using legacy protocol converter.

2.1.3 Sequence diagrams

The Diagram in Figure 24 shows a simplified authentication and registration of client device. It also showcases the subscription to the subject on the client-side and the publishing of a message to the subject.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

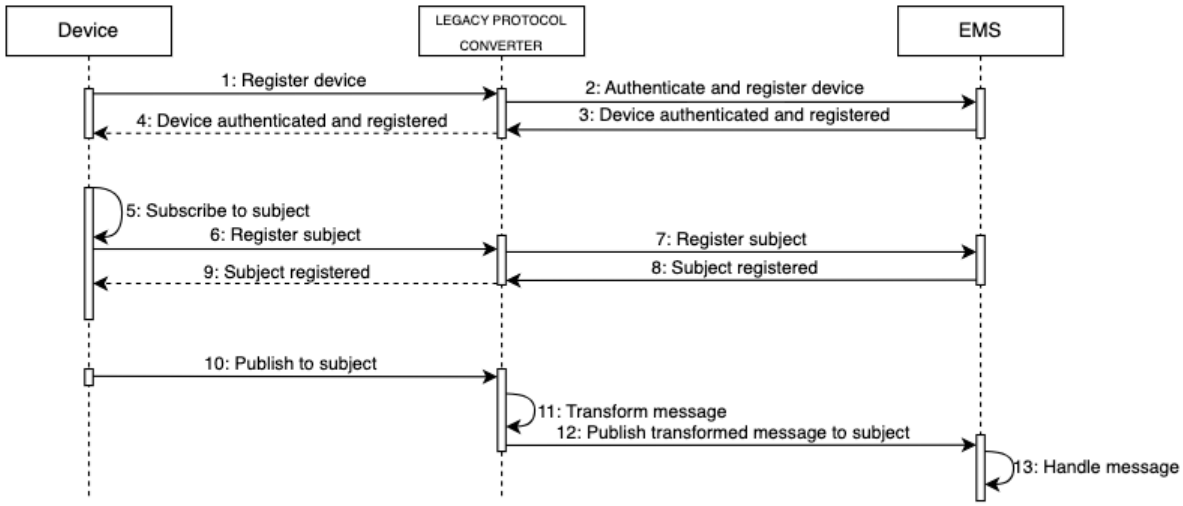

Figure 24: Sequence diagram showing authentication and registration of client device and publish-subscribe pattern.

Diagram in Figure 25 shows how the communication flows when devices initiate a request and when EMS initiates a request. The Legacy protocol converter transforms messages and forwards them.

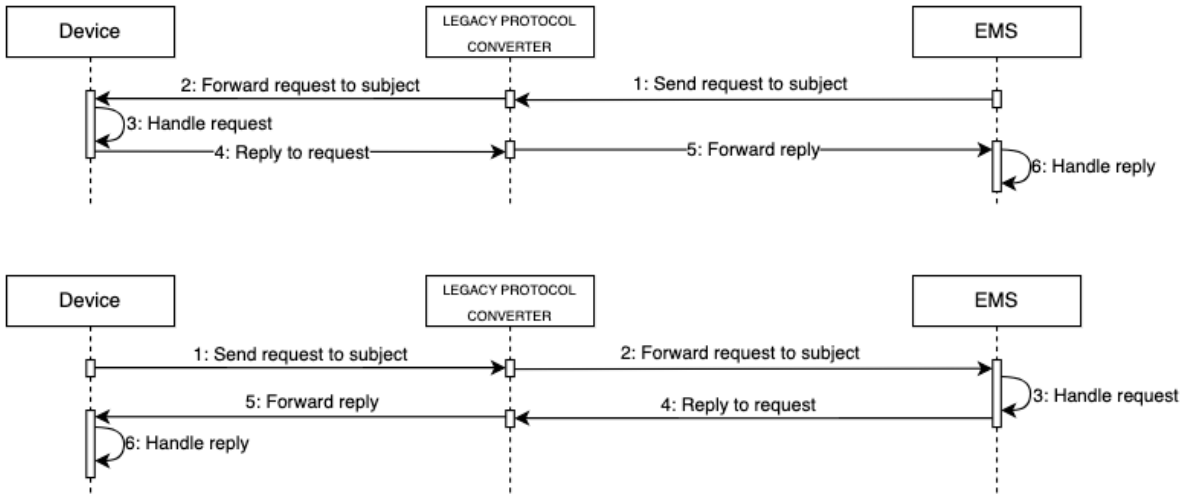

Figure 25: Sequence diagram showing request-reply pattern with legacy protocol converter.

In Figure 26, request-stream pattern is shown, where device sends multiple replies to a request.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

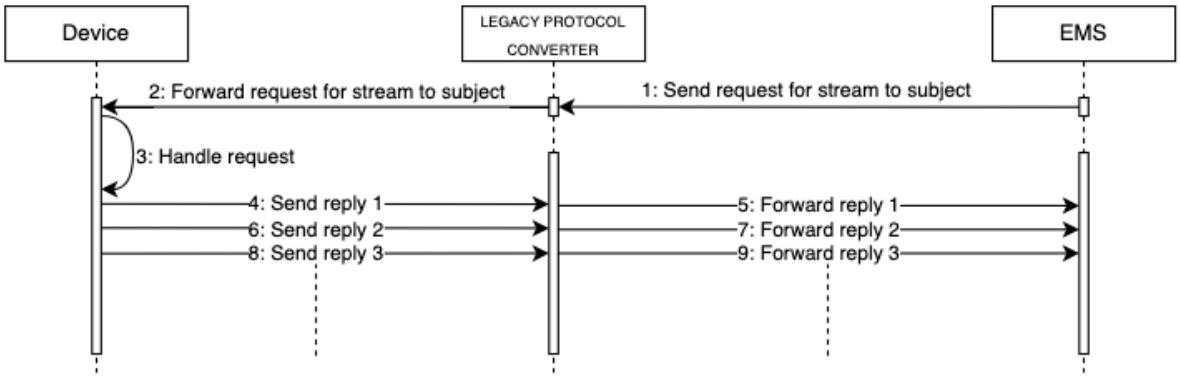

Figure 26: Sequence diagram showing request-stream pattern with legacy protocol converter.

2.2 Registration and authentication of devices

2.2.1 Registration of devices

Each device when it sends its first message, will also register with either the server or the legacy protocol converter. The Legacy protocol converter and server will keep a list of registered devices and each time a device sends a first message, it will be added to the list, alongside with the time of the message. After a certain period (configurable, e.g. 48 hours), devices that have not sent a single message within this timeframe, will be removed from the list.

2.2.2 Authentication of devices

Each device that will connect will also be authenticated. Authentication differs when connecting with NATS or with MQTT, Modbus does not support authentication so Modbus devices will not be authenticated.

2.2.2.1 Authentication with NATS

Devices will authenticate with NATS in one of the following ways:

- Token authentication
- Username/Password credentials
- TLS Certificate
- JWT

2.2.2.2 Authentication with MQTT

Device will authenticate with MQTT with Username/Password credentials.

2.3 Message exchange patterns

Message exchange patterns define how messages are sent and received between applications or components in a distributed system. Both NATS and MQTT support various message exchange patterns to facilitate communication and data flow.

2.3.1 One-way

In a one-way pattern, a message is sent from a sender to a receiver without any expectation of a response. This pattern is suitable for scenarios where the sender only needs to communicate information to the receiver without requiring any acknowledgment.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

2.3.2 Request/response

The request/response pattern involves a sender (requester) sending a message to a receiver (responder) and expecting a response back. This pattern is commonly used for synchronous communication, where the sender waits for the response before proceeding.

2.3.3 Correlation

Correlation is a pattern that enables applications to associate related messages with each other. It allows for tracking and managing conversations between components more effectively, particularly in request/response scenarios.

2.3.4 Request/data stream

Both NATS and MQTT support the request/data stream pattern, where data is streamed from a publisher to multiple subscribers. This pattern is useful for scenarios where real-time data updates or continuous streams of information are required.

2.3.5 Delivery options and subjects

Both NATS and MQTT provide various delivery options and support subjects/topics as a means of addressing messages.

Guaranteed delivery: Both NATS and MQTT offer mechanisms to ensure the reliable delivery of messages. In NATS, it can use a persistent store to ensure message durability and prevent message loss in case of failures. MQTT provides options for Quality of Service (QoS) levels, where higher QoS levels guarantee message delivery and acknowledgment.

Topics: Both NATS and MQTT use topics to categorize and route messages. A topic is a string that identifies the subject of a message, and subscribers can subscribe to specific topics of interest. When a message is published to a topic, it is delivered to all subscribers that have expressed interest in that topic.

NATS support consumers. A consumer serves as a persistent perspective on a stream, serving as a bridge for clients to access and process a specific set of messages within the stream while also maintaining a record of which messages have been successfully received and acknowledged by these clients. Consumers can follow a push-based approach, where messages are actively sent to a designated subject, or a pull-based approach, which enables clients to request batches of messages when needed. With consumers it is possible to provide at least once delivery guarantee, unlike with core NATS where it is at most once delivery guarantee.

Overall, NATS and MQTT are versatile messaging protocols that support various message exchange patterns, making them suitable for a wide range of distributed system architectures and communication requirements.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

3 Use case scenarios

The developed client/server and legacy protocol converter will be used in real-time use cases to showcase its applicability and versatility. Among InterSTORE project partners we devised a plan regarding how and where the developed tools will be put in practise. In general, there are several instances where we will implement developed tools.

3.1 Hybridization of storage systems

HESStec will develop Hybrid Distributed Energy Management Systems (HyDEMS) which will upgrade existing state-of-the-art InMS system, which primarily focuses on real-time control of different storage systems. The upgraded version will also integrate legacy protocol converter, with the goal of allowing different storage systems, used in the aligned use case, to communicate by IEEE 2030.5 over NATS with the Modbus TCP protocol used by HyDEMS. Implementation of legacy protocol will be done via Linux based hardware which is supported and already integrated in HyDEMS.

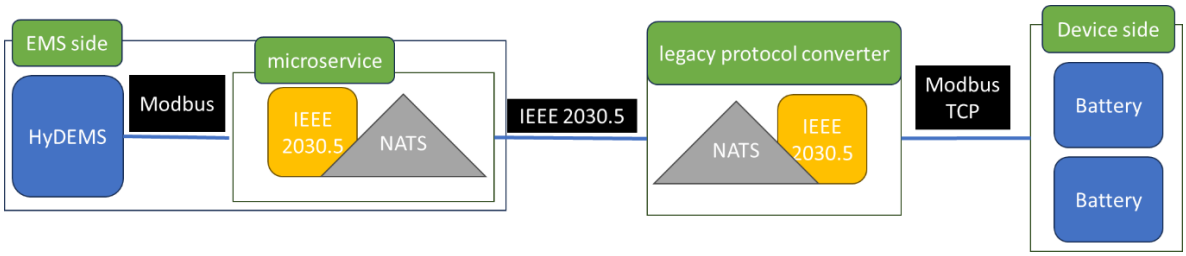

Figure 27: Integration of the IEEE 2030.5 over NATS in HESStec HyDEMS.

3.2 Integration on an inverter

Capwatt is an innovative company that promotes integrated energy solutions and tries to maximise synergies between decentralised resources. To demonstrate the applicability of our recently developed communication protocol, Capwatt will try to connect the second-life lithium battery to its system via IEEE 2030.5 over NATS, and in parallel via ModBus TCP IP. To do this, the Ingeteam inverter will, probably, have to be connected to a switch that will connect the inverter and the EMS in parallel via ModBus TCP IP and the IEEE 2030.5 protocol. To enable communication, it will be necessary to implement a legacy protocol in its LabVIEW management system, as it supports many different protocols (Modbus TCP, GPIB, CAN, LIN, FlexRay, ...), but these have yet to be verified and tested.

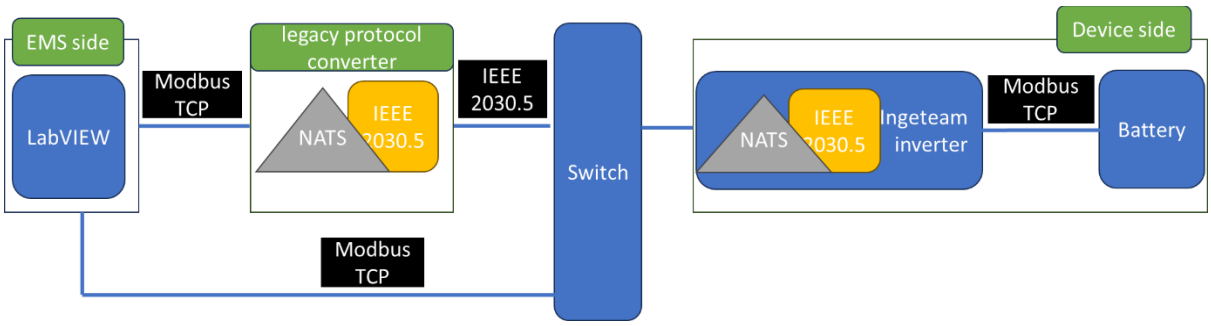

Figure 28: Integration of the IEEE 2030.5 over NATS in Capwatt's LabVIEW.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

3.3 Flexibility monetization and energy communities

CyberGrid's state-of-the-art flexibility management platform CyberNoc allows its users seamless integration of distributed assets, real-time pooling of the available resources, and access to multiple markets where auto-bidding can be used to maximize the revenue from different balancing services. In the scope of our project CyberNoc will implement the newly developed protocol IEEE 2030.5 over NATS directly in its system and use it to connect to different storage systems (batteries and perhaps heat pumps). As most of the deployed storage systems don't have an option to communicate via the new protocol, legacy protocol converters will be used to make the transition. Ideally implementation of it will be done directly to the inverters (Fronius inverters supporting Modbus TCP are in use). Alternatively, Linux based IoTmaxx Gateway GW4100 will be used to make the translation from Modbus TCP to IEEE 2030.5 over NATS.

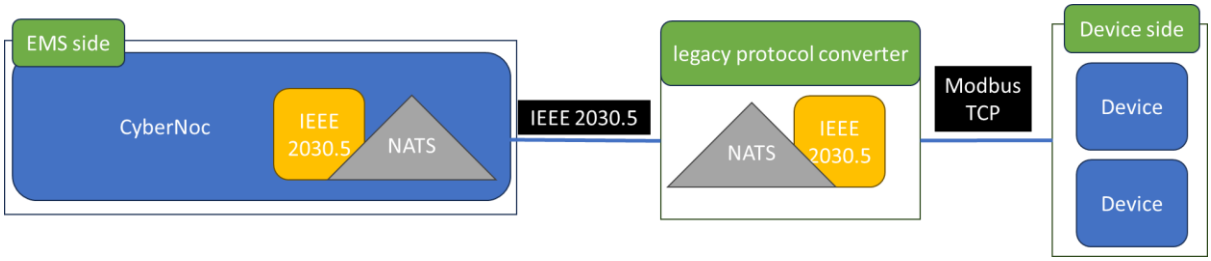

Figure 29: Integration of the IEEE 2030.5 over NATS in CyberGrid's CyberNoc.

3.4 Home management system

Forschungszentrum Jülich developed and deployed a FIWARE-based ICT platform for the integration of different assets. In InterSTORE, the ICT platform will be upgraded to support the integration of hybrid energy storage systems (HESS) and it will be used for the needs of the home management system by Eaton. The list of used HESS consists of a High-energy battery system (Tesla), a High-power battery system (Riello) and can be later expanded with the addition of a photovoltaic (PV) system and heat pumps. Use case 3 of the project aims at testing the IEEE 2030.5 over NATS integration and performance (on the battery systems) compared to current practice (FIWARE). Both battery systems support Modbus TCP and have inverters that do not support the new protocol's implementation yet. Thus, the legacy protocol converter will be implemented on the device side, by means of a Raspberry Pi 4 (RPi4). Since the ICT platform communicates via MQTT and cannot be modified, integration of the legacy protocol converter on the ICT side will be done on an RPi4 device as well.

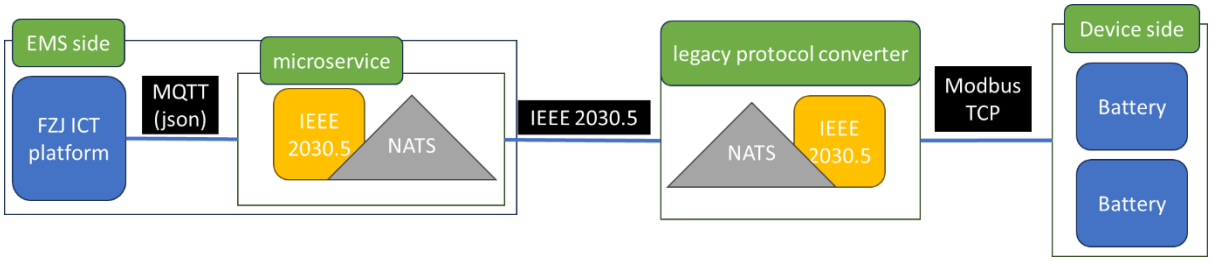

Figure 30: Integration of the IEEE 2030.5 over NATS in FZJ ICT platform.

3.5 Flexibility products management platform

Enel-X is using state-of-the-art platform, which allows the management of different types of flexibility products within the same hybridized portfolio. During the project duration the VPP platform will be further developed to address the rising rate of BESS and EV

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

installations. To optimally allocate needed energy management platform will be upgraded with IEEE 2030.5 via NATS interface. Integration will be two phased. In the first phase (on the picture bellow) legacy protocol and microservice will be implemented to connect devices and management platform. Later the EMS will be upgraded with client/server protocol directly.

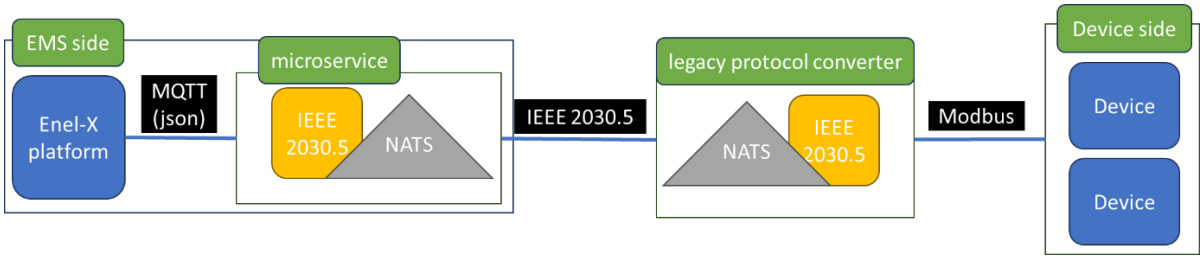

Figure 31: Integration of the IEEE 2030.5 over NATS in Enel-X VPP Flex platform.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

4 Supported protocols

4.1 Supported protocols on device (client) side

4.1.1 Modbus

Modbus is a widely adopted communication protocol in industrial automation. It facilitates the exchange of data using a master-slave architecture, supporting various physical layers like RS-232, RS-485 and Ethernet. Known for its simplicity and interoperability, Modbus finds applications in systems such as SCADA and PLCs for control and monitoring in industrial environments as well as battery and PV inverters.

In this scenario, the client side implements Modbus TCP protocol to establish communication with a legacy protocol converter. By leveraging Modbus TCP, the client can exchange messages with the converter effectively. To ensure smooth interoperability, the client needs to provide detailed descriptions of function codes and registers for the converter's usage. This information allows the legacy protocol converter to accurately convert the messages between Modbus TCP and the legacy protocol.

By utilizing Modbus TCP and providing the necessary information, the client enables seamless integration between modern communication standards and the legacy system. This ensures efficient data exchange and compatibility in industrial automation settings, allowing the client to interface with the legacy infrastructure using Modbus TCP as a bridge. (Modbus, 2023)

4.1.2 MQTT

MQTT is a lightweight messaging protocol specifically designed for efficient communication in IoT applications. It utilizes a publish-subscribe pattern, where publishers send messages to a central broker, which then distributes them to interested subscribers based on topics. MQTT is well-suited for resource-constrained devices and provides different levels of message delivery quality. It has gained widespread popularity in the IoT domain.

In this scenario, the client will utilize the MQTT protocol to facilitate message exchange with a legacy protocol converter. To establish communication, the client will subscribe and publish to predefined topic patterns. The legacy protocol converter will also subscribe and publish to these topics. Both sides must utilize Quality of Service (QoS) level 2 to ensure minimal loss of information during message transmission.

By leveraging the MQTT protocol, the client can seamlessly exchange messages with the legacy protocol converter. The client and converter will subscribe and publish to the appropriate topics, enabling bidirectional communication while maintaining a high level of message delivery reliability using QoS level 2.

Overall, MQTT's lightweight nature and its ability to support QoS levels make it a suitable choice for efficient and reliable communication between IoT devices and legacy systems through the protocol converter. (MQTT, 2023)

4.2 Supported protocols on EMS (server) side

4.2.1 NATS

NATS is a lightweight and high-performance messaging system designed for cloud-native applications. It operates on a publish-subscribe model, prioritizing fast and efficient communication. NATS is recognized for its simplicity, scalability, and low-latency messaging

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

capabilities. It has gained significant popularity in distributed systems, microservices, and IoT applications.

In this scenario, the server side will utilize NATS to facilitate communication with a legacy protocol converter while conforming to the IEEE 2030.5 specification. By leveraging NATS, the server can establish seamless integration with the converter, enabling data exchange following the prescribed standard.

The server will support various communication patterns provided by NATS, including request-reply and publish-subscribe. This flexibility allows for the implementation of different interaction modes with the legacy protocol converter. The request-reply pattern facilitates direct communication, enabling the server to make requests and receive corresponding replies. On the other hand, the publish-subscribe pattern enables the server to publish messages to topics and receive updates from subscribed topics.

By utilizing NATS and supporting multiple communication patterns, the server can efficiently communicate with the legacy protocol converter, adhering to the IEEE 2030.5 specification. This approach ensures compatibility and effective data exchange, making NATS an excellent choice for cloud-native applications that require seamless integration with legacy systems.

4.2.2 MQTT

Communication between server and legacy protocol converter is through MQTT standard with both broker and clients using Quality of Service level 2. Server will act as a broker, while legacy protocol converter will either act as a client or act as a broker, depending on the type of messages as described in paragraph 4.1.2.

4.3 Description of NATS

NATS is a cutting-edge technology that enables the functionality of modern distributed systems. In this context, connective technology plays a crucial role in handling tasks like addressing, discovery, and message exchange that drive common patterns in distributed systems. These patterns involve services/microservices, which are responsible for asking and answering questions, as well as stream processing, which involves making and processing statements.

Modern distributed systems are characterized by a growing number of interconnected components that generate vast amounts of data. To drive business value, these systems leverage both services and streams. Furthermore, they are defined by their ability to function independently of specific locations, allowing for mobility not just in frontend technologies but also in backend processes, microservices, and stream processing—all while maintaining a strong focus on security.

However, the challenges faced by current technologies designed to connect mobile front ends to static backends are becoming evident in the context of these modern systems. Existing technologies often rely on hostname (DNS) or IP and port for addressing and discovery, follow a 1:1 communication pattern, and employ various security patterns for authentication and authorization. While these technologies have been suitable for many scenarios, the evolving landscape of microservices, functions, and stream processing moving to the edge is putting their assumptions to the test.

NATS, on the other hand, manages addressing and discovery based on subjects, not limiting itself to hostname and ports. It defaults to a more versatile M:N communication pattern, which includes 1:1 communication but offers much broader capabilities. This flexibility allows

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

developers to consider the broader picture—how various moving parts can work together in production without the limitations imposed by a 1:1 system. In this context, the need for load balancers, log systems, network security models, proxies, and sidecars can be reduced or eliminated.

Another strength of NATS lies in its deployability, as it can be set up in diverse environments, including bare metal, virtual machines, containers, Kubernetes (K8S), devices, or any chosen environment. NATS works smoothly within deployment frameworks or even independently without them.

Furthermore, NATS emphasizes security and follows a secure-by-default approach, reducing the need for network perimeter security models. This aspect becomes particularly important when considering the mobilization of backend microservices and stream processors, where security often becomes a major concern.

NATS will be used in both client/server architecture and legacy protocol converter. It will act as a main way of communication between all involved devices. (NATS, 2023)

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

5 Message structures and schemas

5.1 IEEE 2030.5

5.1.1 Quick introduction

IEEE2030.5 is a communication standard developed by the Institute of Electrical and Electronics Engineers. It is designed to facilitate communication and resource management between energy companies and end-user energy devices.

The application layer with TCP/IP providing functions in the transport and Internet layers to enable utility management of the end user energy environment, including demand response, load control, time of day pricing, management of distributed generation, electric vehicles, etc. is defined in this standard. Depending on the physical layer in use (e.g., IEEE 802.15.4, IEEE 802.11, IEEE 1901, IEEE 1901.2), a variety of lower layer protocols may be involved in providing a complete solution. Generally, lower layer protocols are not discussed in this standard except where there is direct interaction with the application protocol. The mechanisms for exchanging application messages, the exact messages exchanged including error messages, and the security features used to protect the application messages are defined in this standard. With respect to the Open Systems Interconnection (OSI) network model, this standard is built using the four layer Internet stack model. The defined application profile sources elements from many existing standards, including IEC 61968 and IEC 61850

In the following chapters there are 5 structures defined from IEEE2030.5's sep.xsd file and are also represented in JSON schema. Each structure has a sample with it. This is just to showcase a few samples in XML and JSON format.

The whole JSON schema with all the structures will be developed in WP2.

(IEEE Standard for Smart Energy Profile Application Protocol, 2018)

Table 1: Structures defined in the IEEE2030.5.

Element number	Name and type
1.	DeviceCapability
2.	AbstractDevice
3.	DeviceStatus
4.	EndDevice
5.	EndDeviceList
6.	Registration
7.	SelfDevice
8.	Temperature
9.	FunctionSetAssignmentsBase
10.	FunctionSetAssignments
11.	FunctionSetAssignmentsList
12.	Condition
13.	SubscriptionBase
14.	Subscription
15.	SubscriptionList
16.	Notification

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

17.	NotificationList
18.	DERControlResponse
19.	FlowReservationResponseResponse
20.	AppliedTargetReduction
21.	DrResponse
22.	PriceResponse
23.	Response
24.	ResponseList
25.	ResponseSet
26.	ResponseSetList
27.	TextResponse
28.	Time
29.	DeviceInformation
30.	DRLCCapabilities
31.	SupportedLocale
32.	SupportedLocaleList
33.	PowerStatus
34.	PowerSourceType
35.	PEVInfo
36.	IEEE_802_15_4
37.	IPAddr
38.	IPAddrList
39.	IPInterface
40.	IPInterfaceList
41.	LLInterface
42.	LLInterfaceList
43.	loWPAN
44.	Neighbor
45.	NeighborList
46.	RPLInstance
47.	RPLInstanceList
48.	RPLSourceRoutes
49.	RPLSourceRoutesList
50.	LogEvent
51.	LogEventList
52.	Configuration
53.	PowerConfiguration
54.	PriceResponseCfg
55.	PriceResponseCfgList
56.	TimeConfiguration

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

57.	File
58.	FileList
59.	FileStatus
60.	LoadShedAvailabilityList
61.	ApplianceLoadReduction
62.	DemandResponseProgram
63.	DemandResponseProgramList
64.	DutyCycle
65.	EndDeviceControl
66.	EndDeviceControlList
67.	LoadShedAvailability
68.	Offset
69.	SetPoint
70.	TargetReduction
71.	MeterReading
72.	MeterReadingList
73.	Reading
74.	ReadingList
75.	ReadingSet
76.	ReadingSetList
77.	ReadingType
78.	UsagePoint
79.	UsagePointList
80.	ConsumptionTariffInterval
81.	ConsumptionTariffIntervalList
82.	CostKindType
83.	EnvironmentalCost
84.	RateComponent
85.	RateComponentList
86.	TariffProfile
87.	TariffProfileList
88.	TimeTariffInterval
89.	TimeTariffIntervalList
90.	MessagingProgram
91.	MessagingProgramList
92.	PriorityType
93.	TextMessage
94.	TextMessageList
95.	BillingPeriod
96.	BillingPeriodList

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

97.	BillingMeterReadingBase
98.	BillingReading
99.	BillingReadingList
100.	BillingReadingSet
101.	BillingReadingSetList
102.	Charge
103.	ChargeKind
104.	CustomerAccount
105.	CustomerAccountList
106.	CustomerAgreement
107.	CustomerAgreementList
108.	HistoricalReading
109.	HistoricalReadingList
110.	ProjectionReading
111.	ProjectionReadingList
112.	TargetReading
113.	TargetReadingList
114.	ServiceSupplier
115.	ServiceSupplierList
116.	AccountBalance
117.	AccountingUnit
118.	CreditRegister
119.	CreditRegisterList
120.	Prepayment
121.	PrepaymentList
122.	PrepayModeType
123.	PrepayOperationStatus
124.	ServiceChange
125.	SupplyInterruptionOverride
126.	SupplyInterruptionOverrideList
127.	CreditStatusType
128.	CreditTypeType
129.	CreditTypeChange
130.	ServiceStatusType
131.	RequestStatus
132.	FlowReservationRequest
133.	FlowReservationRequestList
134.	FlowReservationResponse
135.	FlowReservationResponseList
136.	DefaultDERControl

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

137.	FreqDroopType
138.	DER
139.	DERList
140.	DERSettings
141.	DERType
142.	DERAvailability
143.	DERCapability
144.	DERControlBase
145.	DERControl
146.	DERControlList
147.	DERControlType
148.	DERCurve
149.	CurrentDERProgramLink
150.	DERCurveList
151.	CurveData
152.	DERCurveType
153.	DERProgram
154.	DERProgramList
155.	DERStatus
156.	DERUnitRefType
157.	CurrentRMS
158.	FixedPointType
159.	UnsignedFixedPointType
160.	ActivePower
161.	AmpereHour
162.	ApparentPower
163.	ReactivePower
164.	PowerFactor
165.	FixedVar
166.	WattHour
167.	VoltageRMS
168.	ConnectStatusType
169.	InverterStatusType
170.	LocalControlModeStatusType
171.	ManufacturerStatusType
172.	OperationalModeStatusType
173.	StateOfChargeStatusType
174.	StorageModeStatusType
175.	AccountBalanceLink
176.	ActiveBillingPeriodListLink

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

177.	ActiveCreditRegisterListLink
178.	ActiveDERControlListLink
179.	ActiveEndDeviceControlListLink
180.	ActiveFlowReservationListLink
181.	ActiveProjectionReadingListLink
182.	ActiveSupplyInterruptionOverrideListLink
183.	ActiveTargetReadingListLink
184.	ActiveTextMessageListLink
185.	ActiveTimeTariffIntervalListLink
186.	AssociatedDERProgramListLink
187.	AssociatedUsagePointLink
188.	BillingPeriodListLink
189.	BillingReadingListLink
190.	BillingReadingSetListLink
191.	ConfigurationLink
192.	ConsumptionTariffIntervalListLink
193.	CreditRegisterListLink
194.	CustomerAccountLink
195.	CustomerAccountListLink
196.	CustomerAgreementListLink
197.	DemandResponseProgramLink
198.	DemandResponseProgramListLink
199.	DERAvailabilityLink
200.	DefaultDERControlLink
201.	DERCapabilityLink
202.	DERControlListLink
203.	DERCurveLink
204.	DERCurveListLink
205.	DERLink
206.	DERListLink
207.	DERProgramLink
208.	DERProgramListLink
209.	DERSettingsLink
210.	DERStatusLink
211.	DeviceCapabilityLink
212.	DeviceInformationLink
213.	DeviceStatusLink
214.	EndDeviceControlListLink
215.	EndDeviceLink
216.	EndDeviceListLink

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

217.	FileLink
218.	FileListLink
219.	FileStatusLink
220.	FlowReservationRequestListLink
221.	FlowReservationResponseListLink
222.	FunctionSetAssignmentsListLink
223.	HistoricalReadingListLink
224.	IPAddrListLink
225.	IPInterfaceListLink
226.	LLInterfaceListLink
227.	LoadShedAvailabilityListLink
228.	LogEventListLink
229.	MessagingProgramListLink
230.	MeterReadingLink
231.	MeterReadingListLink
232.	MirrorUsagePointListLink
233.	NeighborListLink
234.	NotificationListLink
235.	PowerStatusLink
236.	PrepaymentLink
237.	PrepaymentListLink
238.	PrepayOperationStatusLink
239.	PriceResponseCfgListLink
240.	ProjectionReadingListLink
241.	RateComponentLink
242.	RateComponentListLink
243.	ReadingLink
244.	ReadingListLink
245.	ReadingSetListLink
246.	ReadingTypeLink
247.	RegistrationLink
248.	ResponseListLink
249.	ResponseSetListLink
250.	RPLInstanceListLink
251.	RPLSourceRoutesListLink
252.	SelfDeviceLink
253.	ServiceSupplierLink
254.	SubscriptionListLink
255.	SupplyInterruptionOverrideListLink
256.	SupportedLocaleListLink

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

257.	TargetReadingListLink
258.	TariffProfileLink
259.	TariffProfileListLink
260.	TextMessageListLink
261.	TimeLink
262.	TimeTariffIntervalListLink
263.	UsagePointLink
264.	UsagePointListLink
265.	IdentifiedObject
266.	Link
267.	List
268.	ListLink
269.	Resource
270.	RespondableIdentifiedObject
271.	RespondableResource
272.	RespondableSubscribableIdentifiedObject
273.	SubscribableIdentifiedObject
274.	SubscribableList
275.	SubscribableResource
276.	Error
277.	Event
278.	EventStatus
279.	RandomizableEvent
280.	AccumulationBehaviourType
281.	ApplianceLoadReductionType
282.	CommodityType
283.	ConsumptionBlockType
284.	CurrencyCode
285.	DataQualifierType
286.	DateTimeInterval
287.	DeviceCategoryType
288.	DstRuleType
289.	FlowDirectionType
290.	KindType
291.	LocaleType
292.	mRIDType
293.	OneHourRangeType
294.	PENType
295.	PerCent
296.	PhaseCode

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

297.	PINType
298.	PowerOfTenMultiplierType
299.	PrimacyType
300.	RealEnergy
301.	RoleFlagsType
302.	ServiceKind
303.	SFDIType
304.	SignedPerCent
305.	SignedRealEnergy
306.	TimeOffsetType
307.	TimeType
308.	TOUType
309.	UnitType
310.	UnitValueType
311.	UomType
312.	VersionType
313.	MirrorMeterReading
314.	MirrorMeterReadingList
315.	MeterReadingBase
316.	MirrorReadingSet
317.	MirrorUsagePoint
318.	MirrorUsagePointList
319.	ReadingBase
320.	ReadingSetBase
321.	UsagePointBase

5.1.2 Abstract Device

The AbstractDevice schema includes fields and attributes that describe basic information about a device without going into the specifics of a particular device category. It serves as a template that can be extended and specialized to represent various types of devices, such as meters, inverters, appliances, and more. It's a way to provide common attributes for devices while allowing for flexibility in representing different device types.

The main purpose of the AbstractDevice schema is to capture essential information that can be shared across different devices in a standardized way. This promotes interoperability and uniformity in the representation of devices within a smart energy management system.

In the following four subchapters, AbstractDevice schema is represented in XML format with its sample and in JSON format also with its sample.

5.1.2.1 IEEE2030.5 XML schema

In the following XML, we can observe the structure for AbstractDevice as defined in the sep.xsd file from IEEE2030.5.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
<xs:complextype xmlns:xs="http://www.w3.org/2001/XMLSchema"
name="AbstractDevice">
  <xs:annotation>
    <xs:documentation>The EndDevice providing the resources available
within the DeviceCapabilities.</xs:documentation>
  </xs:annotation>
  <xs:complexcontent>
    <xs:extension base="SubscribableResource">
      <xs:sequence>
        <xs:element name="ConfigurationLink" type="ConfigurationLink"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="DERListLink" type="DERListLink"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="DeviceInformationLink"
type="DeviceInformationLink" minOccurs="0" maxOccurs="1"/>
        <xs:element name="DeviceStatusLink" type="DeviceStatusLink"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="FileStatusLink" type="FileStatusLink"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="IPInterfaceListLink"
type="IPInterfaceListLink" minOccurs="0" maxOccurs="1"/>
        <xs:element name="lFDI" minOccurs="0" maxOccurs="1"
type="HexBinary160">
          <xs:annotation>
            <xs:documentation>Long form of device identifier. See the
Security section for additional details.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="LoadShedAvailabilityListLink"
type="LoadShedAvailabilityListLink" minOccurs="0" maxOccurs="1"/>
        <xs:element name="loadShedDeviceCategory" minOccurs="0"
maxOccurs="1" type="DeviceCategoryType">
          <xs:annotation>
            <xs:documentation>This field is for use in devices that
can shed load. If you are a device that does not respond to
EndDeviceControls (for instance, an ESI), this field should not have
any bits set.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="LogEventListLink" type="LogEventListLink"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="PowerStatusLink" type="PowerStatusLink"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="sFDI" minOccurs="1" maxOccurs="1"
type="SFDIType">
          <xs:annotation>
            <xs:documentation>Short form of device identifier, WITH
the checksum digit. See the Security section for additional
details.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexcontent>
</xs:complextype>
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

5.1.2.2 Sample XML

In the following XML, we can observe the sample for AbstractDevice.

```
<abstractdevice>
  <configurationlink>
    <href>http://example.com/configuration</href>
  </configurationlink>
  <derlistlink>
    <href>http://example.com/derlist</href>
  </derlistlink>
  <deviceinformationlink>
    <href>http://example.com/deviceinfo</href>
  </deviceinformationlink>
  <devicestatuslink>
    <href>http://example.com/devicestatus</href>
  </devicestatuslink>
  <filestatuslink>
    <href>http://example.com/filestatus</href>
  </filestatuslink>
  <ipinterfacelistlink>
    <href>http://example.com/ipinterfaces</href>
  </ipinterfacelistlink>
     <lfdi>0123456789ABCDEF0123456789ABCDEF01234567</lfdi>
  <loadshedavailabilitylistlink>
    <href>http://example.com/loadshedavailability</href>
  </loadshedavailabilitylistlink>
  <loadsheddevicecategory>0001</loadsheddevicecategory>
  <logeventlistlink>
    <href>http://example.com/logevents</href>
  </logeventlistlink>
  <powerstatuslink>
    <href>http://example.com/powerstatus</href>
  </powerstatuslink>
  <sfdi>1234567890ABCDEF</sfdi>
</abstractdevice>
```

5.1.2.3 JSON schema

In the following JSON, we can observe the structure for AbstractDevice. It was obtained from the XML schema.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "Link": {
      "type": "object",
      "properties": {
        "href": {
          "type": "string",
          "format": "uri"
        }
      },
      "required": ["href"],
      "additionalProperties": false
    },
    "ListLink": {
      "type": "object",
      "properties": {
        "href": {
          "type": "string",
          "format": "uri"
        },
        "all": {
          "type": "integer",
          "minimum": 0
        }
      },
      "required": ["href"],
      "additionalProperties": false
    }
  },
  "title": "AbstractDevice",
  "type": "object",
  "properties": {
    "ConfigurationLink": {
      "$ref": "#/definitions/Link"
    },
    "DERListLink": {
      "$ref": "#/definitions/ListLink"
    },
    "DeviceInformationLink": {
      "$ref": "#/definitions/Link"
    },
    "DeviceStatusLink": {
      "$ref": "#/definitions/Link"
    },
    "FileStatusLink": {
      "$ref": "#/definitions/Link"
    },
    "IPInterfaceListLink": {
      "$ref": "#/definitions/ListLink"
    },
    "lFDI": {
      "type": "string",
      "pattern": "^[0-9A-F]{40}$"
    },
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
    "LoadShedAvailabilityListLink": {
      "$ref": "#/definitions/ListLink"
    },
    "loadShedDeviceCategory": {
      "type": "string",
      "pattern": "^[0-9A-F]{4}$"
    },
    "LogEventListLink": {
      "$ref": "#/definitions/ListLink"
    },
    "PowerStatusLink": {
      "$ref": "#/definitions/Link"
    },
    "sFDI": {
      "type": "string",
      "pattern": "^[0-9A-F]{16}$"
    }
  },
  "required": ["sFDI"],
  "additionalProperties": false
}
```

5.1.2.4 Sample JSON

In the following JSON, we can observe the sample for AbstractDevice.

```
{
  "ConfigurationLink": {
    "href": "http://example.com/configuration"
  },
  "DERListLink": {
    "href": "http://example.com/derlist"
  },
  "DeviceInformationLink": {
    "href": "http://example.com/deviceinfo"
  },
  "DeviceStatusLink": {
    "href": "http://example.com/devicestatus"
  },
  "FileStatusLink": {
    "href": "http://example.com/filestatus"
  },
  "IPInterfaceListLink": {
    "href": "http://example.com/ipinterfaces"
  },
  "lFDI": "0123456789ABCDEF0123456789ABCDEF01234567",
  "LoadShedAvailabilityListLink": {
    "href": "http://example.com/loadshedavailability"
  },
  "loadShedDeviceCategory": "0001",
  "LogEventListLink": {
    "href": "http://example.com/logevents"
  },
  "PowerStatusLink": {
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
    "href": "http://example.com/powerstatus"
  },
  "sFDI": "1234567890ABCDEF"
}
```

5.1.3 Event

The Event complex type is used to represent information about events within a smart energy management system. Events could include various occurrences such as changes in device status, measurements reaching certain thresholds, alarms, and other significant incidents. The Event complex type provides a structured way to convey event-related data and metadata.

In the following four subchapters, Event schema is represented in XML format with its sample and in JSON format also with its sample.

5.1.3.1 IEEE2030.5 XML schema

In the following XML, we can observe the structure for Event as defined in the sep.xsd file from IEEE2030.5.

```
<xs:complextype name="Event">
  <xs:annotation>
    <xs:documentation>An Event indicates information that applies to
a particular period of time. Events SHALL be executed relative to the
time of the server, as described in the Time function set section
11.1.</xs:documentation>
  </xs:annotation>
  <xs:complexcontent>
    <xs:extension base="RespondableSubscribableIdentifiedObject">
      <xs:sequence>
        <xs:element name="creationTime" minOccurs="1" maxOccurs="1"
type="TimeType">
          <xs:annotation>
            <xs:documentation>The time at which the Event was
created.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="EventStatus" type="EventStatus"
minOccurs="1" maxOccurs="1"/>
        <xs:element name="interval" minOccurs="1" maxOccurs="1"
type="DateTimeInterval">
          <xs:annotation>
            <xs:documentation>The period during which the Event
applies.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexcontent>
</xs:complextype>
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

5.1.3.2 Sample XML

In the following XML, we can observe the sample for Event.

```
<event>
  <creationtime>1677700000</creationtime>
  <eventstatus>
    <currentstatus>1</currentstatus>
    <datetime>1677701000</datetime>
    <potentiallysuperseded>false</potentiallysuperseded>
  </eventstatus>
  <interval>
    <duration>3600</duration>
    <start>1677700000</start>
  </interval>
</event>
```

5.1.3.3 JSON schema

In the following JSON, we can observe the structure for Event. It was obtained from the XML schema.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "TimeType": {
      "type": "integer",
      "description": "Time is a signed 64 bit value representing the
number of seconds since 0 hours, 0 minutes, 0 seconds, on the 1st of
January, 1970, in UTC, not counting leap seconds."
    },
    "EventStatus": {
      "type": "object",
      "properties": {
        "currentStatus": {
          "type": "integer",
          "enum": [0, 1, 2, 3, 4],
          "description": "Field representing the current status
type."
        },
        "dateTime": { "$ref": "#/definitions/TimeType" },
        "potentiallySuperseded": { "type": "boolean" },
        "potentiallySupersededTime": { "$ref":
"#/definitions/TimeType" },
        "reason": {
          "type": "string",
          "maxLength": 192,
          "description": "The Reason attribute allows a Service
provider to provide a textual explanation     of the status."
        }
      },
      "required": ["currentStatus", "dateTime",
"potentiallySuperseded"],
      "additionalProperties": false
    },
    "DateTimeInterval": {
      "type": "object",
      "properties": {
        "duration": { "type": "integer", "minimum": 0 },
        "start": { "$ref": "#/definitions/TimeType" }
      },
      "required": ["duration", "start"],
      "additionalProperties": false
    }
  },
  "title": "Event",
  "type": "object",
  "properties": {
    "creationTime": { "$ref": "#/definitions/TimeType" },
    "EventStatus": { "$ref": "#/definitions/EventStatus" },
    "interval": { "$ref": "#/definitions/DateTimeInterval" }
  },
  "required": ["creationTime", "EventStatus", "interval"],
  "additionalProperties": false
}
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

5.1.3.4 Sample JSON

In the following JSON, we can observe the sample for Event.

```
{
  "creationTime": 1677700000,
  "EventStatus": {
    "currentStatus": 1,
    "dateTime": 1677701000,
    "potentiallySuperseded": false
  },
  "interval": {
    "duration": 3600,
    "start": 1677700000
  }
}
```

5.1.4 TimeConfiguration

The TimeConfiguration defines the time settings and configurations used within a smart energy management system. It provides standardized elements to communicate information related to time synchronization and management between devices, applications, and services within the smart energy ecosystem.

In the following four subchapters, TimeConfiguration schema is represented in XML format with its sample and in JSON format also with its sample.

5.1.4.1 IEEE2030.5 XML schema

In the following XML, we can observe the structure for TimeConfiguration as defined in the sep.xsd file from IEEE2030.5.

```
<xs:complextype name="TimeConfiguration">
  <xs:annotation>
    <xs:documentation>Contains attributes related to the
configuration of the time service.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="dstEndRule" minOccurs="1" maxOccurs="1"
type="DstRuleType">
      <xs:annotation>
        <xs:documentation>Rule to calculate end of daylight savings
time in the current year. Result of dstEndRule must be greater than
result of dstStartRule.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="dstOffset" minOccurs="1" maxOccurs="1"
type="TimeOffsetType">
      <xs:annotation>
        <xs:documentation>Daylight savings time offset from local
standard time.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="dstStartRule" minOccurs="1" maxOccurs="1"
type="DstRuleType">
      <xs:annotation>
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
        <xs:documentation>Rule to calculate start of daylight savings
time in the current year. Result of dstEndRule must be greater than
result of dstStartRule.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="tzOffset" minOccurs="1" maxOccurs="1"
type="TimeOffsetType">
      <xs:annotation>
        <xs:documentation>Local time zone offset from UTCTime. Does
not include any daylight savings time offsets.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complextype>
```

5.1.4.2 Sample XML

In the following XML, we can observe the sample for TimeConfiguration.

```
<timeconfiguration>
  <dstendrule>02061F03</dstendrule>
  <dstoffset>3600</dstoffset>
  <dststartrule>02E10703</dststartrule>
  <tzoffset>-18000</tzoffset>
</timeconfiguration>
```

5.1.4.3 JSON schema

In the following JSON, we can observe the structure for TimeConfiguration. It was obtained from the XML schema.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "TimeConfiguration",
  "type": "object",
  "properties": {
    "dstEndRule": {
      "type": "string",
      "pattern": "^[0-9A-Fa-f]{8}$"
    },
    "dstOffset": {
      "type": "integer"
    },
    "dstStartRule": {
      "type": "string",
      "pattern": "^[0-9A-Fa-f]{8}$"
    },
    "tzOffset": {
      "type": "integer"
    }
  },
  "required": ["dstEndRule", "dstOffset", "dstStartRule",
"tzOffset"],
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
  "additionalProperties": false
}
```

5.1.4.4 Sample JSON

In the following JSON, we can observe the sample for TimeConfiguration.

```
{
  "dstEndRule": "02061F03",
  "dstOffset": 3600,
  "dstStartRule": "02E10703",
  "tzOffset": -18000
}
```

5.1.5 ServiceChange

The ServiceChange structure specifies a change to the service status within a smart energy management system. It's used to communicate alterations in the operational state of a service, such as starting or stopping a particular service. The structure provides details about the new status and the time at which the change is intended to take effect.

In the following four subchapters, ServiceChange schema is represented in XML format with its sample and in JSON format also with its sample.

5.1.5.1 IEEE2030.5 XML schema

In the following XML, we can observe the structure for ServiceChange as defined in the sep.xsd file from IEEE2030.5.

```
<xs:complextype name="ServiceChange">
  <xs:annotation>
    <xs:documentation>Specifies a change to the service
status.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="newStatus" minOccurs="1" maxOccurs="1"
type="ServiceStatusType">
      <xs:annotation>
        <xs:documentation>The new service status, to take effect at
the time specified by startTime</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="startTime" minOccurs="1" maxOccurs="1"
type="TimeType">
      <xs:annotation>
        <xs:documentation>The date/time when the change is to take
effect.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complextype>
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

5.1.5.2 Sample XML

In the following XML, we can observe the sample for ServiceChange.

```
<servicechange>
  <newstatus>Connected</newstatus>
  <starttime>1681555200</starttime>
</servicechange>
```

5.1.5.3 JSON schema

In the following JSON, we can observe the structure for ServiceChange. It was obtained from the XML schema.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "ServiceChange",
  "type": "object",
  "properties": {
    "newStatus": {
      "$ref": "#/definitions/ServiceStatusType"
    },
    "startTime": {
      "$ref": "#/definitions/TimeType"
    }
  },
  "required": ["newStatus", "startTime"],
  "definitions": {
    "ServiceStatusType": {
      "type": "string",
      "enum": ["Connected", "Disconnected", "Armed for Connect", "Armed
for Disconnect", "No Contactor", "Load Limited"]
    },
    "TimeType": {
      "type": "integer",
      "format": "int64"
    }
  },
  "additionalProperties": false
}
```

5.1.5.4 Sample JSON

In the following JSON, we can observe the sample for ServiceChange.

```
{
  "newStatus": "Connected",
  "startTime": 1681555200
}
```

5.1.6 Error

The Error complex type is used to convey information about errors that occur when a request cannot be completed successfully within a smart energy management system. The Error

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

complex type provides structured details about the nature of the error, facilitating better diagnostics and understanding of the reason behind the failure.

In the following four subchapters, Error schema is represented in XML format with its sample and in JSON format also with its sample.

5.1.6.1 IEEE2030.5 XML schema

In the following XML, we can observe the structure for Error as defined in the sep.xsd file from IEEE2030.5.

```
<xs:complextype name="Error">
  <xs:annotation>
    <xs:documentation>Contains information about the nature of an
error if a request could not be completed
successfully.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="maxRetryDuration" minOccurs="0" maxOccurs="1"
type="UInt16">
      <xs:annotation>
        <xs:documentation>Contains the number of seconds the client
SHOULD wait before retrying the request.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="reasonCode" minOccurs="1" maxOccurs="1"
type="UInt16">
      <xs:annotation>
        <xs:documentation>Code indicating the reason for failure. 0 -
Invalid request format
1 - Invalid request values (e.g. invalid threshold values)
2 - Resource limit reached
3 - Conditional subscription field not supported
4 - Maximum request frequency exceeded
All other values reserved</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complextype>
```

5.1.6.2 Sample XML

In the following XML, we can observe the sample for Error.

```
<error>
  <maxretryduration>60</maxretryduration>
  <reasoncode>2</reasoncode>
</error>
```

5.1.6.3 JSON schema

In the following JSON, we can observe the structure for Error. It was obtained from the XML schema.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Error",
  "type": "object",
  "properties": {
    "maxRetryDuration": {
      "type": "integer",
      "minimum": 0
    },
    "reasonCode": {
      "type": "integer",
      "enum": [0, 1, 2, 3, 4]
    }
  },
  "required": ["reasonCode"],
  "additionalProperties": false
}
```

5.1.6.4 Sample JSON

In the following JSON, we can observe the sample for Error.

```
{
  "maxRetryDuration": 60,
  "reasonCode": 2
}
```

5.2 Supported message structures and formats

5.2.1 Supported message structures and formats in client/server

Both client and server will support JSON/XML message structure and format. Additionally, a header with additional information about device, in type of JSON/XML field will be added to the message.

5.2.2 Supported message structures and formats in legacy protocol converter

Legacy protocol converter will support JSON/XML message structure and format. Additionally, a header with additional information about the device, in type of JSON/XML field will be added to the message. The legacy protocol converter will also be able to receive and send Modbus messages from/to device. When communicating with the server, device ID will be sent in the header.

5.3 Fault and exception messages

Structure of fault and exception messages is in JSON format and is following:

```
{
    "error": "error description",
    "error_code": 2023,
    "ieee2030.5 error": error from Error
structure defined by IEEE2030.5
}
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

6 Message transformation and configuration

6.1 Transformation framework

This part is necessary only for legacy protocol converter as it will be available to existing devices with multiple formats of messages.

The transformation will be done as defined in the configuration file in the YAML format by the user. User must map every message, that is sent from/to device, to the correct IEEE2030.5 structure. It is expected that users will have knowledge about the IEEE2030.5.

Each message to NATS/IEEE2030.5 can be sent in XML or JSON format, depending on which the user specified.

6.2 Configuration options

The configuration file will be in YAML format. Each user will have to specify mappings from their custom messages.

General structure of the configuration file is following:

```
---
lpc:
  transformations:
    - name: string
      description: string
      deviceMqttTopic: string
      modBusInFC: string
      modBusOutFC: string
      natsSubject: string
      fromDeviceFormat: XML/JSON/Modbus
      forwardFormat: XML/JSON
      outgoingMapping: ''
      incomingMapping: ''
```

Within the *transformations* section, you can include multiple transformations as it is structured as a list.

Each transformation must include the following elements: *name*, *description*, *deviceMqttTopic* or modBusInFC and *modBusOutFC*, *natsSubject*, *fromDeviceFormat*, *forwardFormat*, *outgoingMapping*, and *incomingMapping*.

The *name* and *description* fields should provide a brief name and description of the transformation to enhance understanding for others.

Depending on the connection type from the legacy protocol converter to the device, a transformation must include either *deviceMqttTopic*, which specifies the MQTT topic over which this message will be sent, or *modBusInFC*, which identifies the messages to be transformed based on the incoming function code.

The *natsSubject* attribute indicates the subject to which the legacy protocol converter should forward this message.

It is essential to specify the format of both incoming and outgoing messages. For incoming messages, format options include XML, JSON, or Modbus. For outgoing messages, the format options are XML or JSON. The attribute *fromDeviceFormat* can be omitted if attributes *modBusInFC* and *modBusOutFC* are present. Both function codes are decimal.

Each field's mapping is performed using the following format:

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

XML:

```
<mapping>
    <path type=""></path>
    <values>[]</values>
    <pattern></pattern>
</mapping>
```

JSON:

```
"mapping": {
    "path": "",
    "type": "",
    "values": [],
    "pattern": ""
}
```

The *path* attribute instructs the mapper about the location of the value within the JSON object or XPath. The *type* attribute specifies the data type of the value, such as datetime, integer, string, date, etc.

The *values* and *pattern* attributes are used based on the specific use case. If a value can be represented by multiple different values, then *values* must be defined to indicate which IEEE2030.5 value this corresponds to.

The *pattern* attribute is exclusively used with values of type date and datetime. It specifies the pattern of the outgoing value, ensuring it can be correctly interpreted. If it is used in the incoming message, then it will be mapped to this patttern and sent to the device.

$timestamp keyword is reserved and it indicates that here it should be filled with the timestamp of now.

6.3 Transforming MQTT to NATS/IEEE2030.5

The transformation framework will support incoming messages in both XML and JSON formats. XML message transformations will be carried out using XPath, while JSON message transformations will involve accessing JSON objects, following standard practices.

An example of an incoming message in XML format, which will be used to demonstrate the transformation to the Event structure, is as follows:

```
<customevent>
      <datetime>28-08-2023 12:00:35</date>
      <status>active</status>
      <start>28-08-2023</start>
      <duration>900</duration>
</customevent>
```

For legacy protocol converter to transform this to the Event structure, following configuration can be used:

```
---
lpc:
  transformations:
    - name: Transformation to IEEE2030.5 Event in XML format
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
      description: Transformation that maps from custom event to
IEEE2030.5 Event structure in XML format
      deviceMqttTopic: device2.event
      natsSubject: device2.event
      fromDeviceFormat: XML
      forwardFormat: XML
      outgoingMapping:
        '<event>
          <creationtime>$timestamp</creationtime>
          <eventstatus>
            <currentstatus>
              <mapping>
                <path type="integer">/customevent/status</path>
                <values>["scheduled", "active", "cancelled",
"cancelled_with_r", "superseded"]</values>
              </mapping>
            </currentstatus>
            <datetime>
              <mapping>
                <path type="datetime">/customevent/datetime</path>
                <pattern>DD-MM-YYYY HH:mm:ss</values>
              </mapping>
            </datetime>
            <potentiallysuperseded></potentiallysuperseded>
          </eventstatus>
          <interval>
            <duration>
              <mapping>
                <path type="integer">/customevent/duration</path>
              </mapping>
            </duration>
            <start>
              <mapping>
                <path type="date">/customevent/start</path>
                <pattern>DD-MM-YYYY</pattern>
              </mapping>
            </start>
          </interval>
        </event>'
      incomingMapping:
        '<customevent>
          <date>
            <mapping>
                <path type="datetime">/event/eventstatus/datetime</path>
                <pattern>DD-MM-YYYY HH:mm:ss</pattern>
            </mapping>
          </date>
          <status>
            <mapping>
                <path
type="integer">/event/eventstatus/currentstatus</path>
                <values>["scheduled", "active", "cancelled",
"cancelled_with_r", "superseded"]</values>
            </mapping>
          </status>
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
            <start>
              <mapping>
                  <path type="date">/event/interval/start</path>
                  <pattern>DD-MM-YYYY</pattern>
              </mapping>
            </start>
            <duration>
              <mapping>
                  <path type="integer">/event/interval/duration</path>
              </mapping>
            </duration>
          </customevent>'
```

We can observe a transformation process from *customevent* to *Event* and then back from *Event* to *customevent*. Each path contains the XPath to extract the value from *customevent*. XPath is used here because the incoming message is in XML format.

To transform the *datetime* attribute from *customevent*, we define the pattern of the incoming date using the *pattern* attribute. This allows us to perform the transformation correctly.

For the *status* attribute, it's essential to specify the values that this attribute can assume. These values should align with the IEEE2030.5 possible event statuses, maintaining the same order. The *status* type is integer, as per the enumerated enum in the IEEE2030.5 specifications. This mapping will correspond to the index in the array. The IEEE2030.5 values it must map to are: 0 = Scheduled, 1 = Active, 2 = Cancelled, 3 = Cancelled with Randomization and 4 = Superseded.

The *duration* attribute of *customevent* is mapped as is, without the need for a *pattern* or *values* attribute since none are present.

For transforming from JSON or to JSON is very similar, here it is shown how to just transform attribute *datetime* from *customevent* from XML format to JSON and from JSON to XML:

```
…
"EventStatus": {
…
    "dateTime": {
        "mapping": {
            "path": "/customevent/datetime",
            "type": "datetime",
            "pattern": "DD-MM-YYYY HH:mm:ss"
        }
    }
…
}
…
```

And this is how to transform from JSON back to XML.

```
<customevent>
      <date>
          <mapping>
              <path type="date">EventStatus.dateTime</path>
              <pattern>DD-MM-YYYY HH:mm:ss</pattern>
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
            </mapping>
        </date>
...
</customevent>
```

6.4 Transforming Modbus to NATS/IEEE2030.5

When forwarding messages from device and sending messages to device, users must specify which bytes of the Modbus message map to which field of the IEEE2030.5 structure. When legacy protocol converter sends the message back to the device, it will automatically append device ID and *modBusOutFC* from the configuration file.

For outgoing messages users must define which bytes map to which value in the IEEE2030.5 structure. As for the incomingMessages, users must define to which byte do values from IEEE2030.5 map.

```
---
lpc:
  transformations:
    - name: Modbus transformation to IEEE2030.5 Event in JSON
format
      description: Transformation that maps from Modbus to
IEEE2030.5 Event structure in XML format
      modBusInFC: 03
      modBusOutFC: 15
      natsSubject: device2.event
      forwardFormat: XML
      outgoingMapping:
        '<event>
          <creationtime>$timestamp</creationtime>
          <eventstatus>
            <currentstatus>
              <mapping>
                <path type="integer">[4-5]</path>
              </mapping>
            </currentstatus>
            <datetime>
              <mapping>
                <path type="datetime">[5-6]</path>
              </mapping>
            </datetime>
            <potentiallysuperseded></potentiallysuperseded>
          </eventstatus>
          <interval>
            <duration>
              <mapping>
                <path type="integer">[6-7]</path>
              </mapping>
            </duration>
            <start>
              <mapping>
                <path type="date">[7-8]</path>
              </mapping>
            </start>
          </interval>
        </event>'
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
      incomingMapping:
        '000A04{
          "mapping": {
            "path": "/event/eventstatus/currentstatus",
            "type": "integer"
          }
        },
        {
          "mapping": {
            "path": "/event/eventstatus/datetime",
            "type": "long"
          }
        },
        {
          "mapping": {
            "path": "/event/integer/duration",
            "type": "integer"
          }
        },
        {
          "mapping": {
            "path": "/event/integer/start",
            "type": "long"
          }
        }'
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

7 Software architecture and Development Methodology

7.1 Software architecture description

Software architecture of client/server and legacy protocol converter can be seen in Figure 32.

In client/server architecture, the client library, which will be a separate Java package, will be included directly into the client firmware. An additional Java implementation will be provided, but this can also be implemented in a variety of programming languages.

Into the EMS, either the Java library will be included, or the Java library will be packaged as a separate microservice, that will communicate with EMS over API.

On Figure 32 high-level software architecture for the client/server can be seen.

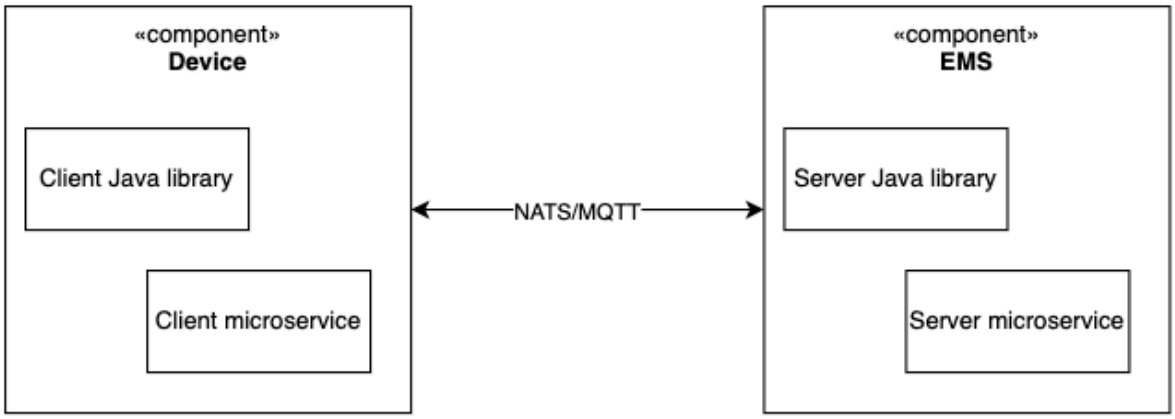

Figure 32: High-level software architecture for the client/server.

In legacy protocol converter architecture, the legacy protocol converter microservice will be included inside the container, virtual machine or IoT device.

On Figure 33, high-level software architecture for the legacy protocol converter can be seen.

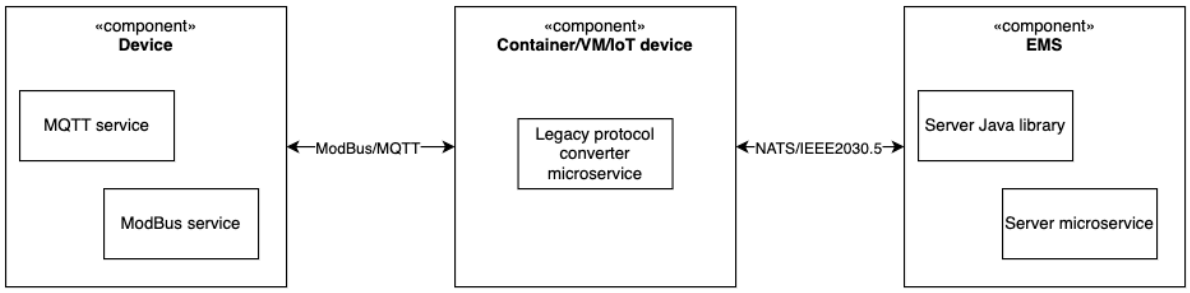

Figure 33: High-level Software architecture for the legacy protocol converter.

7.2 Build and deployment options

Flexibility is important when it comes to deploying solutions. That's why each of the solutions has the option to be deployed as:

- Preconfigured Docker image
- Custom build Docker image
- Prebuilt JAR file
- Custom built JAR file
- Java classes using a JVM

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

- For client/server, Java library can be added to existing Device or EMS system

This allows users to choose the deployment method that best suits their needs, whether they prefer the convenience of a containerized environment or the flexibility of a standalone JAR file. Client, server, and legacy protocol converter can all be deployed as Docker containers and JAR files. Additionally, users can create a custom build from the existing project, allowing them to tailor the solutions to meet their specific requirements.

7.3 Container support on different platforms

In addition to providing flexibility in deployment methods, the solutions also offer support for various platforms and architectures. The containers are designed to run on a wide range of devices, including those with ARM processors. This means that users can deploy the solutions on everything from small, low-power edge devices to powerful servers. The ability to run the containers on devices with ARM processors is particularly useful for organizations that require edge computing capabilities. By deploying the solutions on devices like smart sensors, IoT gateways, and other specialized hardware, users can take advantage of local processing power and reduce the need for data transmission to the cloud. Supporting multiple platforms and architectures also makes it easier for users to integrate our solutions into their existing infrastructure.

7.4 Microservice architecture

Client, server and legacy protocol converter will follow microservice architecture. The client/server and the legacy protocol converter will be developed using standard Java SE in a lightweight, microservice-oriented approach. A Microprofile-compliant microservice framework will be used.

7.5 Programming language and NATS libraries

Project will use Java version 21 (LTS) with OpenJDK 21. It will use NATS library from NATS available here: https://github.com/nats-io/nats.java

7.6 Security

All communication over NATS and MQTT will be secured using TLS and SSL.

7.7 Development methodology

In this project, the Agile approach will be used. It is a software development methodology that emphasizes flexibility, collaboration, and customer-centricity to create high-quality software in a more iterative and adaptive manner. Unlike traditional waterfall methodologies that follow a linear sequence of phases, Agile methodologies embrace change and prioritize delivering functional increments of the software product at shorter intervals. This allows for quicker feedback, continuous improvement, and the ability to respond effectively to evolving requirements.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

8 Software and test procedures requirements

8.1 Beyond state of the Art (from the GA)

The technical advances are: (a) protocol based on the IEEE 2030.5 standard or similar, supported by the latest trends and practical experiences from USA and Australia and aligned with EU InterConnect H2020 project (b) use of connective technology that powers modern distributed systems (e.g., NATS) allowing for effortless M:N connectivity, wide deployment, real-time data exchange and strong security; (c) implementation of connected data Associated with document Ref. Ares(2022)8566949 - 09/12/2022 InterSTORE: Interoperable open-source tools to enable hybridisation, utilisation, and monetisation of storage flexibility 11 spaces to increase user awareness and acceptance of hybrid solutions for the proposed use cases by valorizing data.

8.2 Description of task T1.3 (from the GA)

This task will specify the tools to replicate, adapt, and improve interoperable open-source software to integrate hybrid energy storage systems (HESS). The specification of the tools will foresee monitoring, control and management of the assets and seamless contact with hierarchical structures, such as aggregators and system operators. The specifications will address the following: i) interoperable client/server for distributed energy storage; ii) legacy systems protocol converter; and iii) testing procedures and software tools. The task will focus its attention on the available resources on IEEE2030.5 (such as in GitHub and support documentation). In this task the documentation of the tools will be prepared as a basis for WP2 implementation. Aspects, such as technology transfer to organizations and equipment manufactures for easiness to installation will be addressed as well. It will propose evaluation metrics for the tool's development, implementation and performance. The tools will be categorized by final equipment use and (client) and system operator or aggregator use (Server) or other similar hierarchical structure. T1.3 will foresee the requirement to integrate legacy systems, which will be fully developed and realized in WP2.

8.3 Implementation approach for a generalized interface (from the GA)

The definition of the aforementioned standard leaves still quite some flexibility on how the actual software could be implemented. On the other hand there are available open-source initiatives that can be considered a good starting point such as OpenFMB. OpenFMB has been implemented with the main purpose to integrate a set of already available standards such as CIM and IEC61850. Other relevant open-source initiatives such as OpenLEADR, FledgePower both coming from the Linux Foundation Energy are also relevant in this context. Some standards are also already quite relevant in the sector such as MODBUS. Furthermore, significant work has been performed within the project INTERCONNECT, particularly at the level of data modeling. To maximize the impact, InterSTORE will map its requirements to this set of available solutions creating a middleware that starting from enriching data modeling for hybrid storage is flexible enough to interface a variety of data protocol as envisioned with the FledgePower approach.

With this context in mind, the specification of testing procedures and software tools will be based on the available resources, which can be roughly grouped into three categories, which are explained in the following subsections.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

9 Proposed methodology for testing the interoperability software

The need for architecture arises because the IEEE 2030.5 design is based on a client-server model. Due to this, it is imperative to address both the client (Test Device) and the server (EMS), which are outlined in the architecture. This architecture is divided into two subsections: 9.1.1 and 9.1.2. The former represents the Device (IEEE 2030.5 client), while the latter describes the EMS, where the server houses the IEEE 2030.5 resources.

It's important to note that the architecture will be subject to flexibility and change because the objectives to be achieved are grounded in innovative practices. These practices refer specifically to the incorporation of a publish-subscribe model. Consequently, the testing will address not only the client-server model but also an event-driven architecture.

To summarize, there may be new components added or existing ones removed if they seem redundant when in the project it will be finally fixed.

Section 9.2 describes the CSIP smart inverter profile. This was selected because the specifications outlined in Work Package 1.3 are to be implemented in Work Package 2.3. This package explicitly states that the CSIP smart inverter profile can serve as an example or guideline, to be followed in conjunction with SunSpec testing procedures. For this reason, it's crucial to mention some of the core functionalities that need testing in IEEE 2030.5.

Section 9.3 illustrates what the outcomes should resemble. The payload, formatted in XML, must be validated or verified to ensure it behaves according to the expected results. While Section 9.2 specifies the core function sets that require testing, Section 9.3 details the potential outcomes and describes how data is exchanged between the client and server.

Transitioning to Section 9.4, this part elucidates the testing practices to be adopted, referencing common methodologies prevalent in the field. Each test strategy is interconnected with other testing methods, either directly or indirectly, to fulfill the testing requirements comprehensively.

Finally, Section 9.5 outlines the specific testing procedures, detailing how we assess certain features, function sets, or resources within an IEEE 2030.5 framework.

9.1 Testing architecture

One of the most important functionalities of the testing tools is payload validation. This is very relevant in the context of IEEE 2030.5 for both client and server. This focus makes the testing tools independent on the architecture and corresponding transport technology of the client and server tools. In this way, many solutions based on the IEEE 2030.5 standards can be tested, not only the ones developed in the InterSTORE project. Furthermore, the architecture and transport rely, in many cases and in the InterSTORE case, in already established tools and protocols like NATS, which are assumed to be already tested for supporting different features including SSL security, data integrity, etc. This means that the most important part for the development of IEEE 2030.5 applications for DER interoperability systems is the communication between the EMS and DER devices to achieve scalability, interoperability, security and flexibility.

A significant detail involves ensuring that resources or features are registered on a server, a process highlighted in the SunSpec documentation. This registration enables EMS to maintain a record of IEEE 2030.5 devices, streamlining management and coordination. Given the broad and complex nature of testing IEEE 2030.5 devices, the InterSTORE approach

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

initiates mock testing, which entails simulating the actual device's functionalities. This strategy ensures the device operates as anticipated in real-world scenarios.

In the realm of the Java programming language, tools like Cucumber and Mockito are instrumental in facilitating this mocking and end to end testing functionality. Mocking transcends mere testing, extending to encompass the integration of the client (IEEE 2030.5 device) and server (DER server or EMS), enhancing the efficacy and coherence of the system interactions.

One notable feature of the IEEE 2030.5 protocol is its support for both event-driven and request-response architectures. This dual compatibility makes a hybrid architecture model an optimal choice for implementing the testing strategy. The event-driven architecture primarily hinges on conditions and timing, often referred to as state changes, triggers, or updates. In this context, the concepts of publishing, subscribing, and notifications are interchangeably employed between message brokers of the publisher and subscriber.

Conversely, the request-response architecture is predominantly utilized within the client-server model. As previously indicated, the proposed testing procedure leans on the CSIP (Common Smart Inverter Profile) specification, with its corresponding testing procedures elaborated in the SunSpec documentation. The InterSTORE testing procedures will cater to both request-response and event-driven architectures, ensuring a comprehensive evaluation. In the scenario involving request-response, the test will involve a mocked End Device sending a request using the NATS request-response technique. The responder in this instance will be a Mock Server, akin to the utility server outlined in the project's CSIP, while the requester is portrayed as a mocked End Device. This setup is depicted in the subsequent Figure 34.

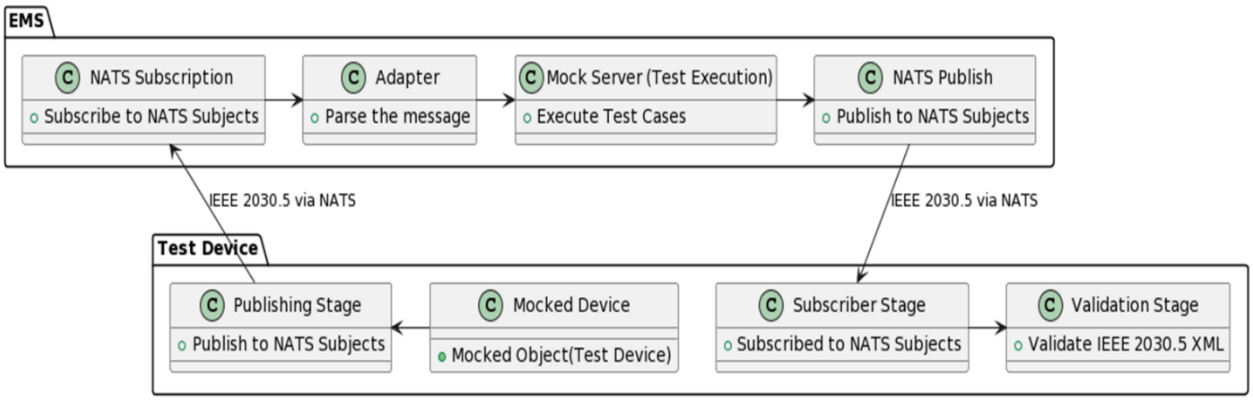

Figure 34: Interoperability Test tool Architecture.

Even though the IEEE 2030.5 Protocol can be addressed with a hybrid model architecture, the NATS message technology supports the Request Reply pattern using its core communication mechanism – publish and subscribe. The next subsections describe the components of the testing architecture in more detail.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

9.1.1 Test Device Overview

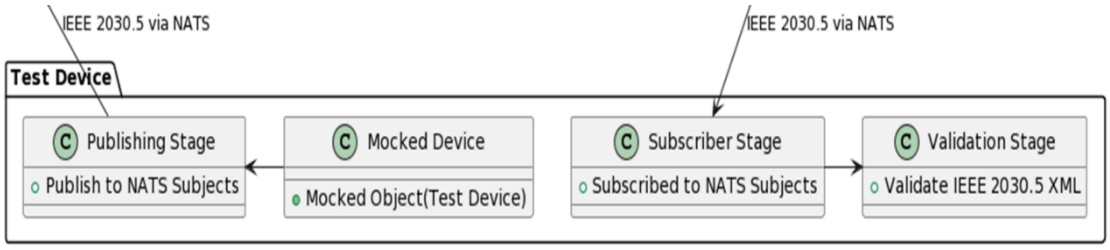

Figure 35: Interoperability Test tool (Client side) Architecture.

Mocked Object: This object represents the Test Device, which can be mocked using various tools, depending on the chosen programming language. For instance, in Java, tools like JUnit and Mockito are commonly used, while the pytest framework is available for Python. The testing flow involves the Test Device publishing queries to the Mock Server via the NATS system, after which the Mock Server, located within the EMS, responds to the queries.

The Test Device has numerous function sets (as per IEEE 2030.5) that need testing. Therefore, it initiates a request to verify whether a specific function set is correctly implemented in the Mock Server (as in IEEE 2030.5, all device capabilities are registered within the Mock Server). The request comprises a URI, an HTTP method, and headers (if applicable). This payload is then published under a NATS subject, with the critical requirement being that the message adheres to the IEEE 2030.5 specifications.

Testing requests are conducted with stubbed data from the Test Device, varying based on the specific test being performed. The essence of this process is to ensure that each function set operates in strict accordance with the standards and expectations set forth by the IEEE 2030.5 protocol.

9.1.2 EMS Overview

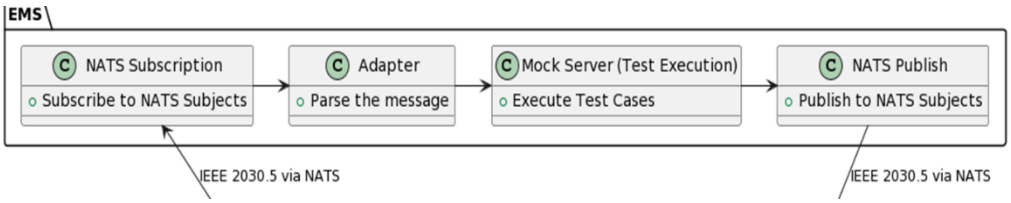

Figure 36: Interoperability Test tool (Server side) Architecture.

NATS Adapter: The EMS is subscribed to the Subject. A NATS server is assumed to be set up to route the traffic (which is not in the above figure but part of the NATS publish subscribe system). After that, the adapter parses the message and will choose the corresponding test case for the incoming message under a given Subject.

Mock Server: The mock server will run the mock test directed by the adapter and would publish the response to the same Subject, and the Test Device would subscribe to the Subject to receive the response and check it against the expected stubbed response data through validation.

9.2 CSIP Smart Inverter Profile in IEEE 2030.5

The CSIP smart invertor profile describes the core functionalities to test in the context of IEEE 2030.5. In IEEE 2030.5, a resource is a piece of information that a server exposes. These resources are used to represent aspects of a physical asset such as a smart inverter, attributes relating to the control of those assets (e.g., Volt-VAr curve), and general

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

constructs for organizing those assets. IEEE 2030.5 resources are defined in the IEEE 2030.5 XML schema and access methods are defined in the Web Application Description Language (WADL). The schema is generally organized by Function Sets, a logical grouping of resources that cooperate to implement IEEE 2030.5 features. IEEE 2030.5 provides a rich set of Function Sets (e.g. Demand Response Load Control, Pricing, Messaging, Metering, etc.) to support a variety of use cases.

The common functionalities to test are described below, according to the CSIP Smart Inverter Profile.

9.2.1 Time

The utility server uses the Time function set to distribute the current time to clients. Time is expressed in Coordinated Universal Time (UTC). Server event timing is based on this time resource.

9.2.2 Device Capability

The utility server uses the Device Capability resource to enumerate the function sets it supports. Clients use this function set to discover the location (URL) of the enumerated function sets.

9.2.3 End Device

The EndDevice function set provides interfaces to exchange information related to specific client or EndDevice.

9.2.4 Function Set Assignments (FSA)

The FunctionSetAssignments function set provides the mechanism to convey the grouping assignments of each DER. Grouping with FSAs can be implemented in diverse ways.

9.2.5 Distributed Energy Resource (DER)

The DER function set provides an interface to manage Distributed Energy Resources (DER). It is the primary function set for issuing DER controls.

9.3 CSIP IEEE 2030.5 Implementation

9.3.1 Device Capability

The DeviceCapability resource is the starting point for discovering resources on the server. It provides links to the entry point of function sets supported by the server.

```
<DeviceCapability href="/sep2/dcap" xmlns="urn:ieee:std:2030.5:ns">
  <ResponseSetListLink href="/sep2/rsps" all="0"/>
  <TimeLink href="/sep2/tm"/>
  <UsagePointListLink href="/sep2/upt" all="0"/>
  <EndDeviceListLink href="/sep2/edev" all="3"/>
  <MirrorUsagePointListLink href="/sep2/mup" all="0"/>
</DeviceCapability>
```

Figure 37: Example of DeviceCapability.

9.3.2 End Device

End Device (e.g. Smart Inverter) gets access to EndDevices through an EndDeviceListLink that is available via the server's DeviceCapability resource. The utility server should return a custom EndDeviceList for each device making the request. If the querying device is a DER, the server should return an EndDeviceList consisting of a single entry – the EndDevice instance of the requesting DER.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
<EndDeviceList href="/sep2/edev" subscribable="1" pollRate="86400" all="1" results="1"
xmlns="urn:ieee:std:2030.5:ns">
  <EndDevice href="/sep2/edev/1">
    <DERListLink href="/sep2/edev/1/der" all="0"/>
    <lFDI>12a4a4b406ad102e7421019135ffa2805235a21c</lFDI>
    <LogEventListLink href="/sep2/edev/1/log" all="0"/>
    <sFDI>050044792964</sFDI>
    <changedTime>1514836800</changedTime>
    <enabled>1</enabled>
    <FunctionSetAssignmentsListLink href="/sep2/edev/1/fsa" all="1"/>
    <RegistrationLink href="/sep2/edev/1/rg"/>
  </EndDevice>
</EndDeviceList>
```

Figure 38: Example of EndDevice.

9.3.3 Function set Assignment

The bellow response is the Function Set Assignment associated with the End Device. Function Set Assignments are generic assignments upon function sets, the assignments can be seen as configuration or kind of arrangement for associated programs or features.

```
<FunctionSetAssignmentsList all="2" href="/edev/3/fsal" results="2"
subscribable="0" xmlns="urn:ieee:std:2030.5:ns">
    <FunctionSetAssignments href="/edev/3/fsal/0" subscribable="0">
        <DemandResponseProgramListLink all="2" href="/edev/3/fsal/0/drpl"/>
        <MessagingProgramListLink all="1" href="/edev/3/fsal/0/msgl"/>
        <mRID>0ED30F5A0000</mRID>
        <description>FunctionSetAssignment Suave</description>
    </FunctionSetAssignments>
    <FunctionSetAssignments href="/edev/3/fsal/1" subscribable="0">
        <MessagingProgramListLink all="1" href="/edev/3/fsal/1/msgl"/>
        <mRID>0ED30F5A0001</mRID>
        <description>FunctionSetAssignment Guttural</description>
    </FunctionSetAssignments>
</FunctionSetAssignmentsList>
```

Figure 39: Example of a FunctionSetAssignment.

9.3.4 Distributed Energy Resource (DER)

A DER function set is essentially a collection of specific capabilities or functionalities that a DER device can perform. These can include functionalities like real-time measurement reporting, reactive power control, demand response, or voltage regulation, among other things. These function sets define what kind of operations a particular DER can support and are designed to ensure interoperability within a smart grid environment. It's designed to independent of other entities such as End Device and FunctionSetAssignments and both can inherit DER.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
<DERProgramList all="1" href="/derp" results="1" xmlns="urn:ieee:std:2030.5:ns">
    <DERProgram href="/derp/0">
        <mRID>01BE7A7E57</mRID>
        <description>Example DER Program</description>
        <DERControlListLink all="2" href="/derp/0/derc"/>
        <primacy>2</primacy>
    </DERProgram>
</DERProgramList>
```

Figure 40: Example of a DER.

9.4 Testing Methodology

9.4.1 Mock Testing

In many scenarios, real-time data is unavailable to fulfil testing conditions. In these cases, mock external dependencies can be employed as placeholders to mimic the behaviour of actual objects. For example, consider an email service tasked with sending confirmation messages upon user registration. During testing, it's impractical to send actual emails every time a test case runs. Therefore, this service should be mocked during the user registration testing phase. This approach ensures the system is tested under conditions that simulate the expected outcome (i.e., sending an email) without executing the actual operation, thus maintaining efficiency and preventing unnecessary actions during the testing process.

How Mock Testing fits in IEEE 2030.5 Context

Both the End Device and utility Server shall be mocked, with mocking aiming to emulate the behaviour of the real device or object. In a mocked test, a "virtual device" is created with the same functionalities as the real one. It allows to check if certain methods are called, with the right arguments, in the right sequence, and so on. In essence the mocking allows to check how the object under test interacts with its dependencies (i.e., did it make the right calls, in the right order, with the right data).

9.4.2 Stubbing Testing

Stubbing is primarily employed to regulate the data or state returned by a method, thereby facilitating the testing of the component utilizing this stubbed method or object. It enables the isolation of the system under test from external dependencies by offering predetermined responses. Typically, stubbing is indifferent to whether a method gets called; its main function is to ensure a specified output is returned when the method is indeed invoked.

How Stubbing Testing fits in IEEE 2030.5 Context

Utility servers frequently interact with multiple EndDevices to orchestrate demand-response events or inquire about the status of energy resources. Stubs can be instrumental in simulating the anticipated states these end devices would return, thereby enabling an emphasis on the analysis and decision-making processes within the utility server. End devices, such as Smart Inverters, are required to comply with communication protocols stipulated by IEEE 2030.5. Consequently, a stub can be configured to imitate the utility server's varied responses to assorted requests originating from the end device. This strategy permits concentrated testing on the end device's responses to a spectrum of valid or invalid feedback.

9.4.3 Integration Testing

Integration testing examines the interactions between two or more components, ensuring that various blocks of code work together as expected. This process is crucial when there's a communication channel (such as APIs) or data exchange between different parts of the

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

code. The specific feature responsible for these operations must undergo thorough testing to confirm its ability to seamlessly integrate multiple components. For example, consider a scenario where you need to retrieve data from a database to execute your business logic. Here, two distinct software components must integrate effectively. The medium or channel (be it an API or a similar construct) facilitating this integration must be rigorously tested. This crucial phase of the testing process is known as integration testing.

How Integration Test fits in IEEE 2030.5 Context

Integrated testing involves establishing a connection between various components, such as the End Device and the Utility Server. In this scenario, the mocked End Device initiates a request to the mocked Utility Server, which then executes the corresponding test case. Subsequently, a response is sent back to the End Device for verification. In essence, the interaction between the mocked End Device and Utility Server is facilitated in this manner, ensuring that the components integrate and function cohesively.

9.4.4 End To End Testing

The purpose of end-to-end testing is to verify the flow of a message that complies with IEEE 2030.5 specifications. This process involves both a client and a server, with the client initiating the message flow through a NATS system.

a. In the test scenario, the client is represented as a mock object with attributes such as a URI, an HTTP method, and potentially a payload.
b. This mocked object is encapsulated within a NATS message and published from the device.
c. To execute the above step, a connection to the NATS server is established, followed by publishing the message to a specific subject.
d. Subsequently, the subscriber in the EMS receives the message and extracts its contents, which may include the methods such as Get, Post, Put, Delete, URI, and any payload. These contents are directed towards a specific server resource.
e. Upon identifying the resource and the required operation, the resources are represented as stubs in inbuilt data structures like arrays or HashMaps. Then, the server (which could be a mock server like WireMock or a real server) performs the operation.
f. The outcome of this operation, often in XML format (especially in the case of a GET operation), is then published back from the EMS under the same subject for the subscriber present on the device side.
g. Finally, the received message is validated to ensure the output aligns with the expected results.

All the above steps have to be automated using common software automating open-source tools like cucumber, can trigger the actions one by one to complete the flow of testing, from one end to other end. For example, the mock server has to be up before the testing start as well as the NATS server connect operation has to be up before the flow of testing start i.e. those will be considered as the initial set ups of the flow. The order of the flow will depend on priority order of the test components to invoke, like mentioned above, servers should start first.

9.5 How to test the IEEE 2030.5 Client

Testing the IEEE 2030.5 client consists of different steps. First, we need to specify the purpose of the test, the setup, the detailed procedure, and define the pass and fail criteria of the test.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

Device capability is one of the resources that has to be present in the IEEE 2030.5 server. For this reason, we describe the related testing process, as follows:

Purpose

The device capability test verifies that the IEEE 2030.5 Client can retrieve the Device Capability (DCAP) resource from an IEEE 2030.5 server and use the resource information provided.

Setup

Verify that a DeviceCapability resource exists on the Utility Server and include at least one resource in the DeviceCapability resource.

Procedure

1. Perform a GET operation on DeviceCapability resource from the Server
2. Process the retrieved DeviceCapability resource and verify there is at least one resource in the DeviceCapability.
3. Perform a GET operation on the found resource in step 2 and process the response payload from the Server.

Pass/Fail Criteria

1. The IEEE 2030.5 Client requested and received the DeviceCapability resource from the Server. The Server responded with 200 OK or other status to represent success and returned a conformant payload for its DeviceCapability.
2. The IEEE 2030.5 Client found at least one resource included in the DeviceCapability. The Server included at least one resource link in the returned DeviceCapability to the Client.
3. The IEEE 2030.5 Client successfully received the payload of the found resource from the DeviceCapability resource. The Server responded with 200 OK or other status to represent success and returned a conformant payload for its resource.

The above-mentioned steps are presented in request response way, but the implementation will incorporate the publish and subscribe mechanism. For simplicity, the example was described in a similar way to a request/response architecture.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

10 Open-source access on GitHub

10.1 GitHub repository

The source code for the Client/Server, Legacy protocol converter and testing procedures will be available on GitHub. We have created a repository: https://github.com/Horizont-Europe-Interstore

Within the repository there are currently four projects:

- Interoperable client/server for distributed energy storage
- Legacy system protocol converter
- Testing procedures and software tools
- Interoperable data spaces framework

The repository is currently private (by invitation only), but will be made public when the initial version is developed.

A screenshot of the repository is shown in the figure below.

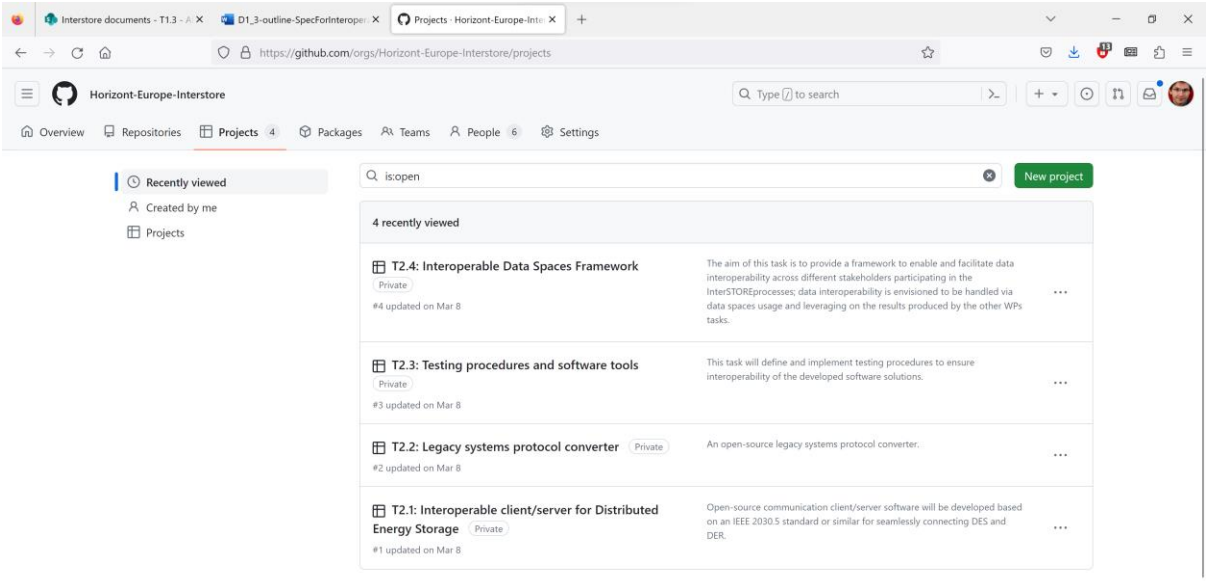

Figure 41: Open-source repository on GitHub.

10.2 Open-source license models

Project partners are currently discussing the possible OpenSource license models (as described in the following sub-chapters) and which to use for the Client/Server, Legacy protocol converter and testing tools source code. This decision needs to be made before the GitHub repository is made public, sometime in March 2024.

10.2.1 MIT license

The MIT License (Massachusetts Institute of Technology License) is an open-source software license widely used in the software development community. It is known for its simplicity and permissiveness, making it one of the most popular licenses for open-source projects. Here's a description of the key aspects of the MIT License:

1. Permission: The MIT License grants users permission to use, modify, distribute, and sublicense the software, both in source code and binary form, without any

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

restrictions. This means that you can freely use, modify, and distribute software under this license.

2. No Warranty: The license includes a disclaimer stating that the software is provided "as is," without any warranties or guarantees. Users are using the software at their own risk, and the original authors or copyright holders are not liable for any damages or issues arising from its use.

3. License Notice: When you distribute or include the MIT-licensed code in your project, you must include a copy of the MIT License text and copyright notice in your project's documentation or source code. This notice typically includes the original copyright holder's name and the license text.

4. Copyright Notice: The license includes a copyright notice, which typically states that the original copyright holders own the copyright to the software. However, this copyright notice doesn't restrict the use of the software as long as the license terms are followed.

10.2.2 Apache License 2.0

The Apache License is a widely used open-source software license known for its balance between promoting open collaboration and protecting contributors and users. It is often used for projects hosted by the Apache Software Foundation (ASF), but it can also be applied to other projects. Here's a description of the key aspects of the Apache License, version 2.0, which is one of the most common versions:

1. Permissive and Business-Friendly: The Apache License is considered a permissive license, similar to the MIT License. It allows users to freely use, modify, distribute, and sublicense the software, both in source code and binary form, without imposing stringent restrictions. This makes it business-friendly and suitable for commercial applications.

2. Patent Grant: One unique feature of the Apache License is its patent grant clause. Contributors grant a patent license to anyone who uses, modifies, or distributes the software under the license. This provision helps protect users from potential patent claims related to the software.

3. Attribution: While the license doesn't require the prominent display of a copyright notice (as some other licenses do), it does require that a NOTICE file be included in the distribution of the software. This file includes attribution to the original authors and any third-party software dependencies that are redistributed with the project.

4. Modifications and Derivatives: Users are free to modify the software and create derivative works. However, they must clearly indicate any changes made to the original source code when distributing modified versions.

5. No Warranty: Similar to the MIT License, the Apache License includes a disclaimer that the software is provided "as is" without any warranties or guarantees. Users assume all risks when using the software.

6. Compatibility: The Apache License is known for its compatibility with other open-source licenses. This means that you can often include Apache-licensed code in projects with different licenses without significant conflicts.

7. Copyleft-like Provisions: While the Apache License is permissive, it includes provisions that address the use of trademarks associated with the software and the use of the software's name in derived works. These provisions help protect the integrity of the project's branding.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

(Apache, 2023)

10.2.3 GNU GPL license

The GNU General Public License (GPL) is a widely used open-source software license that was created by the Free Software Foundation (FSF). It is known for its strong copyleft provisions, which aim to ensure that software remains free and open throughout its lifecycle. There are several versions of the GPL, with version 3 being the most recent and commonly used. Here's an overview of the key characteristics of the GPL:

1. Copyleft: The GPL is often referred to as a "copyleft" license because it uses copyright law to protect the freedom of the software. When a program is released under the GPL, anyone who receives it also receives the rights to view, modify, and distribute the source code. If you modify and distribute the software, you must make your modified source code available under the same GPL terms. This ensures that derivative works also remain open and free.
2. Free Software Philosophy: The GPL is rooted in the philosophy of free software, as defined by the Free Software Foundation. It emphasizes the importance of software being both "free as in freedom" (open source) and "free as in beer" (no cost). Users have the freedom to run, study, modify, and distribute the software.
3. Distribution Obligations: If you distribute GPL-licensed software, whether in binary or source code form, you are obligated to provide recipients with a copy of the GPL license, the complete corresponding source code, and information on how to access and obtain the source code. This ensures that recipients have the same rights you had when you received the software.
4. Compatibility with Other GPL Versions: The GPL allows for compatibility with different versions of the license. For example, you can often combine code released under GPL version 3 with code released under GPL version 2, but the resulting work will be subject to the terms of GPL version 3.
5. No Additional Restrictions: The GPL does not permit adding any further restrictions on the software beyond those defined by the GPL itself. This means you can't impose additional limitations on users beyond what the GPL allows.
6. Commercial Use: The GPL does not restrict commercial use of GPL-licensed software. Companies can use, modify, and distribute GPL-licensed software in their products, but they must still comply with the GPL's copyleft provisions when distributing it.
7. No Warranty: Like many open-source licenses, the GPL includes a disclaimer stating that the software is provided without any warranties. Users are encouraged to use the software at their own risk.
8. Enforcement: The Free Software Foundation and other organizations actively enforce the terms of the GPL. They may take legal action against individuals or entities that violate the license by not providing source code when required.

Overall, the GPL is a powerful and influential open-source license that aims to protect and promote the principles of software freedom. It has been instrumental in fostering a vibrant and collaborative open-source software ecosystem. However, its strong copyleft provisions may not be suitable for all projects, especially those that aim for more permissive licensing. Developers and organizations should carefully consider the implications of the GPL before choosing it as the license for their software.

(GNU, 2025)

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

10.2.4 Berkeley Software Distribution (BSD)

The Berkeley Software Distribution License, is a permissive open-source software license. It originated at the University of California, Berkeley, and has been used for various software projects, including the Berkeley Unix operating system (from which it gets its name) and many other software applications and libraries.

The BSD License is characterized by its permissiveness and minimal restrictions on how the software can be used, modified, and distributed. Here are some key points about the BSD License:

1. Freedom to Use: The BSD License allows anyone to use the software for any purpose, whether it's for personal, academic, commercial, or proprietary use.
2. Freedom to Modify: You are free to modify the source code of the software and adapt it to your needs without any restrictions.
3. Freedom to Distribute: You can distribute both the original software and your modified versions, either as open source or as part of proprietary software, without being required to release the source code of your modifications.
4. Attribution: The BSD License typically requires that you include a copy of the original copyright notice and the license text in your redistributions, whether they are binary distributions or source code distributions.
5. No Warranty: The license usually includes a disclaimer stating that the software is provided "as is" without any warranties or guarantees of fitness for a particular purpose. Users and distributors are responsible for any risks associated with using the software.

There are several variants of the BSD License, including the 2-Clause BSD License and the 3-Clause BSD License. The main difference between these variants is the presence of an advertising clause in the original 4-Clause BSD License, which required that any advertising materials that mentioned the use of the software must include an acknowledgment of its origin. The 3-Clause BSD License and 2-Clause BSD License omit this advertising clause, making them simpler and more widely used.

Overall, the BSD License is known for its flexibility and is often chosen for projects where the primary goal is to maximize the potential for reuse and integration into both open-source and proprietary software systems. However, it's essential to carefully read and understand the specific terms of the BSD License used for a particular software project, as there may be variations or additional conditions beyond the standard clauses.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

11 Conclusion

In this specification we have provided an in-depth description and specification of the interoperable client/server for distributed energy storage and the legacy systems protocol converter.

Based on this specification, in WP2, open-sourced implementation of client/server and legacy protocol converter will be developed. They will provide support for IEEE2030.5 communication between devices and EMS systems in original XML format, as well as IEEE2030.5 in JSON format.

They will provide support for next generation NATS messaging as a communication protocol between devices and EMS systems. This will enable message-driven, loosely coupled and scalable communication platform, which will provide out-of-the-box support for computer cloud environments.

Legacy protocol converter will provide support for ModBus and MQTT and provide a transformation and configuration framework, which will allow simple and fully configurable transformation of messages between ModBus, MQTT, NATS, IEEE2030.5 XML and JSON messages.

We have specified the message exchange patterns, which the client/server and legacy protocol converter will provide support for, including one-way, request/response and data streaming patterns. We have also specified correlation, delivery options, subjects and registration and authentication of devices. We have also defined fault and exception signalling.

We have also provided the description of software architecture. Both the client/server and legacy protocol converter follow the microservice architecture. They will be implemented in Java using open source OpenJDK and provided as Docker containers, pre-built Java JAR archives, or custom-built configurations for specific use cases. All software artifacts will be available on GitHub.

The client/server and the legacy protocol converter, as defined in this specification, will be developed within WP2 and used on several use cases within the InterSTORE project, including hybridization of storage systems, integration on an inverter, flexibility monetization and energy communities, home management system, and flexibility products management platform.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

12 REFERENCES

IEEE Standard for Smart Energy Profile Application Protocol. (2018). *IEEE Std 2030.5-2018 (Revision of IEEE Std 2030.5-2013)*, 1-361. doi: 10.1109/IEEESTD.2018.8608044.

Modbus. (2023, September 25). Retrieved from Mobus: https://modbus.org/

MQTT. (2023, September 25). Retrieved from MQTT: https://mqtt.org/mqtt-specification/

GNU. (2025, September 25). Retrieved from GPL: https://www.gnu.org/licenses/gpl-3.0.html

Apache. (2023, September 25). Retrieved from Apache: https://www.apache.org/licenses/LICENSE-2.0

NATS. (2023, September 25). Retrieved from Documentation: https://docs.nats.io/

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

13 LIST OF TABLES

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

14 LIST OF FIGURES

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.