Version 30-7-2025: Updated specifications for EEBUS

Request For Proposal (RFP): Residential Flexibility, Interoperability HEMS and Flexible Energy-Intensive Devices

ElaadNL and the Flexiblepower Alliance Network (FAN) invite suppliers to participate in a co-funded project to develop and test open-source software connectors that enable Home Energy Management Systems (HEMS) to communicate with flexible energy-intensive devices using standardized protocols. This RFP covers five work packages, each targeting a key protocol or integration layer:

WP	Protocol / Focus	Deliverable
1	S2 / PEBC	HEMS + device + optional cloud receiver
2	Matter 1.4	HEMS + Matter-compliant device
3	EEBUS (SHIP/SPINE)	HEMS + EEBUS-compatible device
4	Modbus converter	Secure HW+SW bridge (e.g. S2 → Modbus)
5	OCPP 2.1 proxy	Virtual proxy for HEMS ↔ CSMS/charger

All code will be published under the Apache 2.0 license and maintained in the official ElaadNL GitHub repository, with CI pipelines, documentation templates, and SonarQube-based quality monitoring in place.

Why Participate?

- Receive funding for development work with clear deliverables
- Lead the market with future-ready solutions
- Accelerate deployment with access to reference implementations and test facilities
- Matchmaking support for consortia (e.g. device + software partner)

Focus Use Cases

- Grid-capacity limiting based on DSO capacity profiles (OpenADR)
- Dynamic tariff optimization to shift or store load
- Self-consumption maximization using local PV and flexible loads

Who Can Apply?

- For WP 1–3, proposals must cover the full chain: HEMS, device, and software
- For WP 4, proposers may focus on converter development (hardware + software)
- For WP 5, HEMS-side development only is permitted (ElaadNL supplies the charger)
- Software firms must apply as part of a consortium

Submission & Contact

Submit your proposal by e-mail to Marisca Zweistra at rfp@elaad.nl no later than Friday, 19 September 2025, 17:00 CET.

Definitions and Acronyms

Aggregator	A market party translating signals from the market or grid into commands for households. (This document applies a broader definition than the market role definition of aggregator.)
BRP	Balance Responsible Party
DSO	Distribution System Operator
EEBUS	A use-case-driven interoperability protocol allowing household devices and energy systems to communicate through a standardized semantic data layer.
EV	Electrical Vehicle
FEID	Flexible energy-intensive device
HEMS	Home Energy Management System
Matter	An open-source, IP-based connectivity standard for smart home and IoT devices, enabling secure, reliable, and interoperable communication across ecosystems.
MODBUS RTU	A serial communication protocol widely used for monitoring and controlling energy devices in buildings.
MODBUS TCP	A variant of the MODBUS protocol using TCP/IP networks for communication.
OCPP	Open Charge Point Protocol; an open standard for communication between charging stations and backend systems.
OpenADR	Open Automated Demand Response, a client-server-based demand response protocol allowing energy suppliers and grid operators to remotely manage flexibility through standardized messaging.
PEBC	Post-Equivalent Base Case is a control signal used to coordinate device behavior based on standardized grid capacity profiles, typically implemented in devices or cloud-based systems as a PEBC receiver.
Partner	Selected Proposers that partake in the project
Project	The development and testing of work packages in this RFP.
Proposer	Party or parties that submit a proposal for this RFP.
RFI	Request For Information
RFP	Request For Proposal
S2	A flexibility protocol focusing on energy management across multiple devices, abstracting the communication between devices and the HEMS, enabling devices to collaborate independently of their communication language. More information at S2standard.org
SHIP	Smart Home Interoperability Protocol is a communication protocol that enables standardized data exchange and control between energy devices and a HEMS.
SPINE	Smart Premises Interoperable Neutral-message Exchange is a semantic data model and message framework used by SHIP and EEBUS to ensure consistent, interoperable communication between devices and systems in the energy domain.
TSO	Transmission System Operator

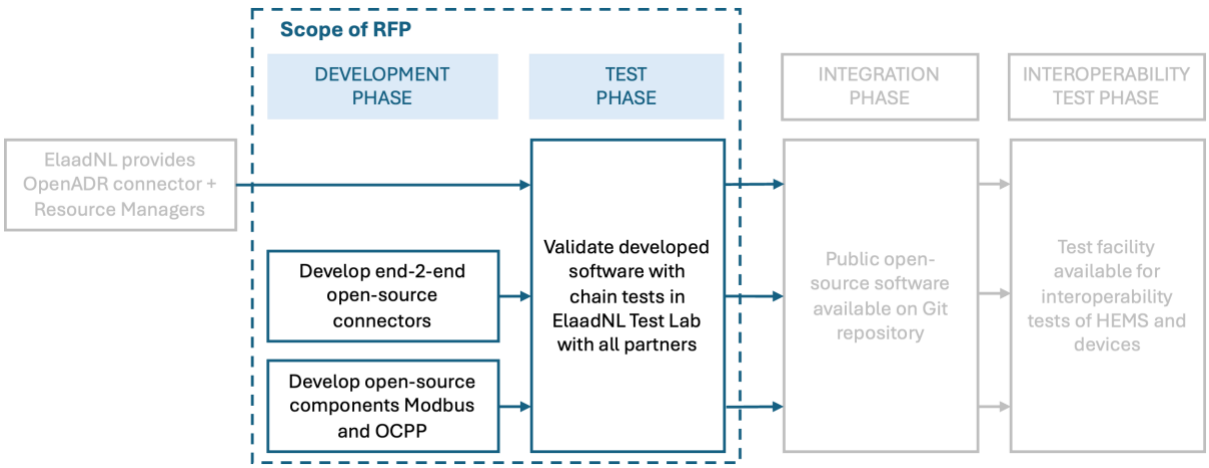
Contents

1. Introduction

The energy transition in the Netherlands is driving rapid growth in EVs, heat pumps, home batteries, and solar panels. These devices offer major potential for residential flexibility by shifting energy use to times of high renewable supply and reducing grid congestion.

Unlocking this potential requires interoperable Home Energy Management Systems (HEMS) that coordinate and optimize device operation. HEMS enable both automated responses to external signals (e.g. price or grid capacity) and smart scheduling based on household preferences. More background information on our research and previous Request for Information can be found in Appendix A.

This Request for Proposal (RFP), initiated by ElaadNL and Flexiblepower Alliance Network (FAN) under the Dutch National Grid Congestion Action Program, invites proposals for developing and testing interoperable HEMS components and flexible energy intensive devices. As part of our roadmap toward interoperability, this RFP focuses on the initial development of open-source software, with ElaadNL taking the lead and bearing overall responsibility for coordination and delivery.



Why participate?

- Receive funding for development work with clear deliverables
- Lead the market with future-ready solutions
- Accelerate deployment with access to reference implementations and test facilities

By contributing, companies help build a shared foundation for residential flexibility and benefiting both consumers and the grid.

2. Scope of Work

2.1 Objectives

The objective of this RFP is to develop open-source, standardized software connectors that enable seamless communication between HEMS and selected FEIDs.

These connectors must:

- **Enable interoperability** by supporting consistent communication across different devices and vendors by implementing selected protocols (S2, EEBUS and Matter).
- **Support local and cloud integration** to function both in local HEMS deployments and cloud-based HEMS, including hybrid solutions with cloud-device relays.
- **Ensure secure and reliable communication** by implementing security features as defined by each protocol, ensuring authentication, encryption, and session integrity across all components. More details are provided in Section 4.1.1.
- **Support selected use cases** that can be found in Section 2.2. Any logic and additional interfaces required to support these use cases are expected to be part of the HEMS solution and are not included in the development scope of this RFP.
- **Adopt a modular, documented, and testable architecture** that allows multiple development teams (consortia) to contribute effectively and ensure long-term maintainability. More details are provided in Section 4.1.6.

2.2 Key Use-Cases

This project focuses on three core use cases that demonstrate the value of interoperability between HEMS and connected devices:

1. **Limiting Peak Grid Demand:** The DSO forecasts grid load and sends capacity profiles to the HEMS, which then limits import/export using local flexibility. This helps prevent congestion during high-demand periods, especially in winter.
2. **Dynamic Tariff Optimization:** The HEMS responds to fluctuating electricity prices by shifting consumption to lower cost periods and storing energy (e.g. in a battery or hot water buffer). This reduces household energy costs and grid pressure during peak pricing periods.
3. **Optimizing Self-Consumption:** The HEMS maximizes the use of self-generated solar energy by intelligently distributing surplus power to batteries and/or EVs. This increases energy independence and reduces grid feed-in during local solar generation peaks.

More detailed descriptions of each use case are provided in Appendix A.4.

2.3 Project Scope and Boundaries

- **HEMS Integration Phase:** HEMS manufacturers are expected to develop one or more protocol connectors as part of this RFP. However, in the subsequent Integration Phase (outside the scope of this RFP), they will be required to support all selected protocols, either by building their own or integrating existing open-source connectors developed through this project.
- **Interoperability Test Phase:** The goal of this project is to stimulate market adoption of interoperable, plug-and-play solutions. Participating parties commit to participate in the Interoperability Test Phase where they will test and demonstrate their connectors on real devices and HEMS platforms at the ElaadNL Test Lab. An invitation to this phase will follow in late 2025 or early 2026. Parties not involved in open-source development are also welcome to participate in the next phase.
- **Protocols and Architecture:** This project builds on existing architectures, protocols, and messaging standards. It explicitly does not aim to create new frameworks, but rather to implement selected protocols effectively and with interoperability in mind.
- **Hardware:** Development of hardware or basic device-side flexibility functions is out of scope. The focus is purely on software integration via connectors, except for WP4 (see Section 2.4 and 6).
- **Use Case Logic:** The internal logic of HEMS platforms, including how they optimize or prioritize different use cases, is considered proprietary and is not in scope of this RFP. The described use cases serve only to guide the implementation of interfaces, messages, and integration testing to demonstrate interoperability.

2.4 Work Packages

This RFP project consists of five work packages, each of which Proposers can apply for. Proposers are expected to develop and deliver end-to-end solutions for the full scope of the selected work package.

- **Work Packages 1, 2, and 3** cover the implementation of the protocols S2, Matter, and EEBUS, respectively.

- **Work Package 4** involves the development of a converter that translates messages from S2, Matter, or EEBUS into Modbus RTU and Modbus TCP. Given the security requirements, the expected outcome is a combined hardware and software solution.

- **Work Package 5** focuses on the development of an OCPP proxy, enabling two systems, CPO backend and HEMS, to send and manage charging profiles. This is expected to be a virtual (non-hardware) solution.

Every work package consists of:

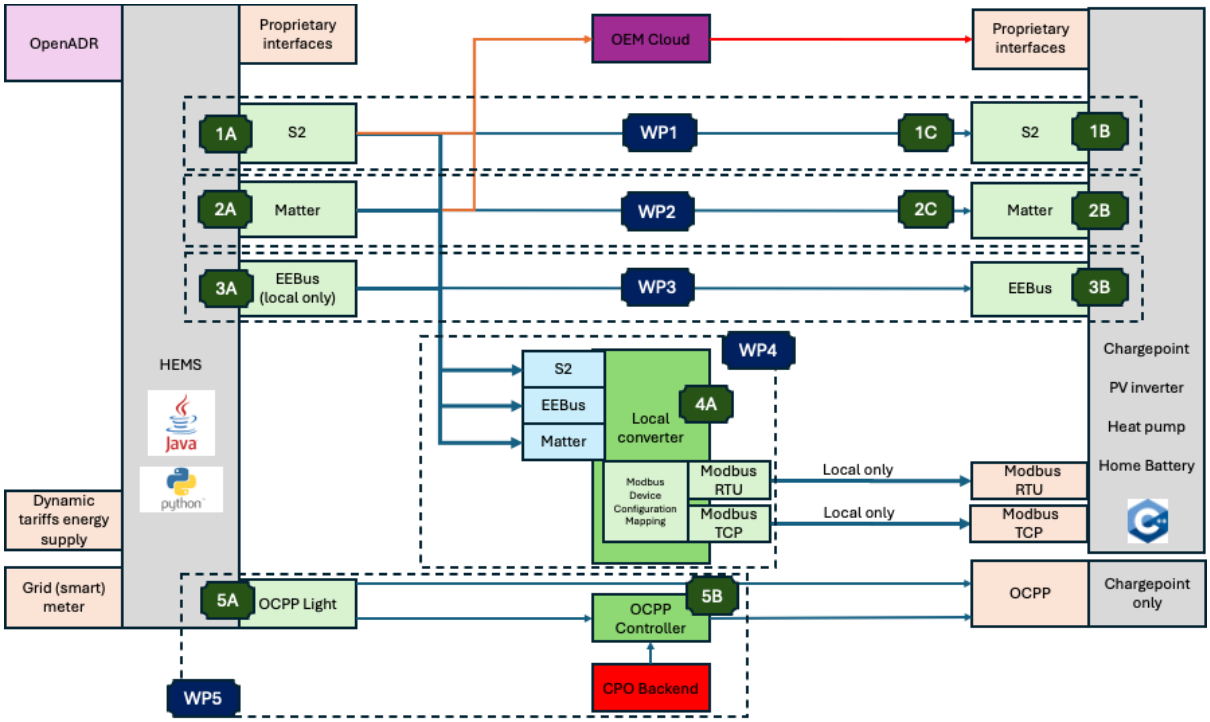
- Developing open-source software
- Implementing the connector in physical/cloud HEMS and/or physical FEID.
- Demonstrating successful communication between HEMS and device, where the HEMS sends a control action and the device executes it as intended.

		HEMS	HEMS Cloud (optional)	Device
WP1	**S2**	1A: PEBC control implementation	1C: PEBC receiver (optional)	1B: PEBC control implementation
WP2	**Matter**	2A: Matter HEMS	2C: Matter-Compatible Cloud Gateway (optional)	2B: Matter Device
WP3	**EEBUS**	3A: SHIP and SPINE		3B: SHIP and SPINE
WP4	**Modbus**			4A: Local Converter Modbus
WP5	**OCPP**	5A: OCPP "Light" Connector		5B: OCPP Controller

Detailed information on the work packages and deliverables can be found in Section 3.

3. Technical Specifications for the Deliverables

This chapter outlines the specific deliverables for each work package (WP) as defined in Section 2.4. Each WP is split into subcomponents (HEMS, Device, and where applicable HEMS Cloud). The descriptions include clear acceptance criteria that will serve as the basis for evaluation and delivery.



Above conceptual representation of IT architecture was created to clarify the different possible information flows between HEMS and FEID(s). A prerequisite for this architecture was to be technology agnostic and support local, cloud and hybrid information flows. In this figure all possible information flows are represented. Only the green coloured shapes are in scope of this RFP.

3.1 WP1 – S2 Protocol

3.1.1 WP1A – PEBC Control Implementation (HEMS-side)

Description	Develops the S2 connector on the HEMS side, implementing the PEBC control type from EN 50491-12-2. Since the S2 standard is not prescribing a transport method we choose JSON over web sockets based on s2-ws-json
Technical Requirements	1. Programming languages: Python, Java, Go, C/C++, Rust (others require review) 2. Support for message security and conformance to standard

Acceptance Criteria	1. Can send the following S2 messages: a. SelectControlType b. PEBC.Instruction (PowerEnvelope) 2. Can receive the following S2 messages: a. ResourceManagerDetails b. InstructionStatusUpdate c. PEBC.PowerConstraints d. PEBC.EnergyConstraints e. RevokeObject f. PowerMeasurement g. PowerForecast 3. Implements message encoding/decoding and lifecycle handling (Handshake(Response), SessionRequest, and ReceptionStatus messages) 4. Supports both local (direct) and remote (1C) control 5. Includes test tools and sample configuration
Documentation & Testing	1. Functional overview and integration instructions 2. Sample configurations and usage examples 3. Gherkin-style tests or equivalent CI pipelines

3.1.2 WP1B – PEBC Control Implementation (Device-side)

Description	Implements the receiving side of the PEBC protocol for devices such as EVSE, BESS, or Heatpump units.
Technical Requirements	1. Programming languages: C/C++, Rust 2. Embedded compatibility, low resource footprint. We target embedded devices capable of running embedded Linux
Acceptance Criteria	1. Can receive and respond to the following messages: a. SelectControlType b. PEBC.Instruction 2. Can send the following messages: a. ResourceManagerDetails b. PEBC.PowerConstraint c. PEBC.EnergyConstraints d. RevokeObject e. InstructionStatusUpdate f. PowerMeasurement g. PowerForecast 3. Translates control messages into local actions 4. Responds with status and telemetry Supports both local (direct) and remote (1C) control 5. Includes test tools and sample configuration
Documentation & Testing	1. Local build instructions for embedded targets 2. Logging and telemetry format description

3.1.3 WP1C – PEBC Receiver (HEMS Cloud) (Optional)

Description	Implements cloud-side receiver for PEBC messages, preserving the semantics and behaviors of the protocol.
Technical Requirements	1. Cloud-compatible language stack (Python, Java, Node.js, Go, .NET) 2. Scalable architecture with documented API
Acceptance Criteria	1. Fully compatible with HEMS-side implementation 2. Can receive and forward messages to local or OEM-managed devices 3. Maintains PEBC state tracking and integrity
Documentation & Testing	1. API reference and usage examples 2. Security and authentication mechanisms

3.2 WP2 – Matter Protocol

This work package focuses on leveraging the Matter 1.4.1 protocol to enable interoperability and flexibility control for Electric Vehicle Supply Equipment (EVSE), while laying the groundwork for broader support of other Flexible Energy-Intensive Devices (FEIDs), including Heat Pumps, Home Batteries, and Solar Panels.

Although the Matter 1.4.1 specification introduces foundational support for EVSE, it does not yet provide complete functionality for all required flexibility use cases, particularly for FEID types beyond EVSE. Therefore, this work package will also address the current limitations of the standard and define a roadmap for necessary extensions or adaptations.

The goal is twofold:

- To deliver a fully functional, Matter-based software foundation for EVSE interoperability based on existing clusters and device types; and
- To define and document the message structures, clusters, and interactions required within Matter to enable the three residential energy flexibility use cases

Respondents are expected to independently analyze the Matter 1.4.1 specification and assess its capabilities and limitations. Based on this assessment, they must develop a concrete message set that enables the use cases described above, either by using existing standard elements or by identifying and clearly documenting the necessary extensions. This includes specifying which attributes, commands, and events are to be implemented, monitored, or invoked by both HEMS and device-side nodes, in accordance with the Matter data and interaction models.

3.2.1 WP2A – Matter Implementation (HEMS-side)

Description	This work package focuses on developing a Matter controller implementation on the HEMS side that can interact with devices supporting the Energy EVSE Cluster (0x0099). The implementation shall conform to the Matter 1.4.1 specification and prepare for future support of Heat Pumps, Home Batteries, and Solar Panels.
Technical Requirements	1. Programming languages: Python, Java, Go, C/C++, Rust (others require review). 2. Adherence to Matter's secure communication model using AEAD (AES-CCM). 3. Compliance with the Matter data model and interaction model (Read/Write/Invoke/Subscribe). 4. Support for operational certificates (NOC) and discovery via DNS-SD.
Acceptance Criteria	1. Demonstrated commissioning of a Matter-compliant EVSE device. 2. Ability to issue control commands (SetTargets, EnableCharging, DisableCharging, EnableDischarging) and subscribe to telemetry (State, SupplyState, NextChargeRequiredEnergy, NextChargeTargetSoC, SessionEnergyCharged, SessionEnergyDischarged). 3. Clear demonstration of Matter Interaction Model functionality. 4. End-to-end integration with the device-side implementation via a shared Fabric. 5. Explicit mapping of Matter messages (attributes, commands, events) required to support the three use cases.
Documentation & Testing	1. Functional overview and API integration instructions. 2. Example commissioning, control, and telemetry workflows. 3. Usage instructions for integration with future FEID types including the required messages and required changes within the Matter specification

3.2.2 WP2B – Matter Implementation (Device-side)

Description	This task involves implementing a Matter node that conforms to the Energy EVSE Cluster (0x0099) as defined in Matter 1.4.1. The node will simulate or control an EVSE and expose key functionality such as charge control, telemetry, and state reporting. It must also implement optional clusters for electrical energy measurement (0x0091), electrical power measurement (0x0090), and device energy management (0x0098) to support extended flexibility use cases.

Technical Requirements	1. Programming languages: C/C++ or Rust. 2. Designed for embedded environments with low memory footprint. 3. Support for secure commissioning, certificate-based authentication (DAC, NOC), and conformance with TLV encoding. 4. Conformance to Matter's data model and message reliability mechanisms (MRP).
Acceptance Criteria	1. Successful commissioning with a Matter controller using PASE or CASE. 2. Implementation of the Energy EVSE Cluster and selected optional clusters. Correct interpretation of control commands: SetTargets, EnableCharging, DisableCharging, EnableDischarging. 3. Accurate reporting of attributes: State, SupplyState, NextChargeRequiredEnergy, NextChargeTargetSoC, SessionEnergyCharged, SessionEnergyDischarged. 4. End-to-end communication with the HEMS-side implementation.
Documentation & Testing	1. Device-side functional overview and test instructions. 2. Sample configuration files and code snippets. 3. Logs from simulated charging sessions and telemetry updates.

3.2.1 WP2C – Matter-Compatible Cloud Gateway (Optional)

Description	Develop a Matter-compatible gateway service that enables cloud-based monitoring and control of Electric Vehicle Supply Equipment (EVSE) devices connected to a local Matter fabric. Matter is not a cloud-native protocol and does not support direct device-to-cloud communication. Therefore, this work package focuses on building a secure, standards-compliant local gateway that acts as a Matter controller or commissioner, while exposing a cloud API for remote interaction. This gateway will serve as a reference architecture for bridging cloud services with Matter devices. The implementation must be explicitly designed for EVSE devices using the Matter-defined Energy EVSE Cluster (0x0099), but must also be architected with extensibility in mind, such that future support for other FEIDs (e.g. Home Batteries, Heat Pumps, Solar Panels) can be added with minimal effort.
Technical Requirements	1. Implements the Matter controller or commissioner role, conforming to Matter 1.4.1. 2. Maintains secure local sessions with Matter nodes via PASE or CASE.

	3. Language stack: Python, Java, Node.js, Go, or .NET. 4. Provides a RESTful or event-driven (e.g. MQTT/WebSocket) cloud API. 5. Maps cloud-side commands to Matter operations: Read, Write, Invoke, Subscribe. 6. Cloud communication must respect Matter's trust and fabric boundaries. 7. Must handle telemetry routing, command translation, and event propagation. 8. Designed to support modular extensions for additional FEID device types and clusters.
Acceptance Criteria	1. Establishes a secure Matter fabric and commissions at least one EVSE node (Device Type 0x0099). 2. For EVSEs, demonstrates full support for: ○ Commands: SetTargets, EnableCharging, DisableCharging, EnableDischarging ○ Attributes: State, SupplyState, NextChargeRequiredEnergy, NextChargeTargetSoC, SessionEnergyCharged, SessionEnergyDischarged 3. Provides a secure API/protocol (e.g. S2 if possible, otherwise HTTPS or MQTT) that allows external clients to send control commands and receive telemetry. 4. Ensures end-to-end security and reliability from the cloud interface to the Matter device. 5. Gateway architecture and implementation must be documented as extensible, with guidance for integrating additional device types and clusters. 6. Explicit mapping of Matter messages (attributes, commands, events) to cloud API endpoints and request formats, supporting the following three use cases:
Documentation & Testing	1. Gateway architecture, security model, and component interfaces. 2. API reference and usage examples. 3. Test logs demonstrating cloud-to-local control and telemetry. 4. Example extension scenario for a second FEID (e.g. Battery or Heat Pump). 5. Integration guidance for future cluster/device types using the same gateway pattern.

3.3 WP3 – EEBUS Protocol

This work package focuses on the implementation and validation of interoperable communication using the EEBUS protocol stack, specifically targeting the SPINE semantic layer and the SHIP secure transport protocol. The implementation must adhere to EEBUS Use Case specifications and follow the defined actor structures, scenario flows, and state machine logic to ensure interoperability across multiple vendors and device types.

Each WP below defines a targeted use case. The proposer must implement the mandatory scenarios, actors, and functions specified in the relevant EEBUS Use Case Technical Specifications. To facilitate this, filenames of the corresponding EEBUS documents are included.

3.3.1 WP3A – SHIP and SPINE Implementation (HEMS-side)

Description	Open-source implementation of SHIP and SPINE on the HEMS side, acting as Energy Guard or EMS. The implementation must support sending limits to multiple flexible devices.
Technical Requirements	1. Languages: Python, Java, Go, C/C++, Rust 2. Implement EEBUS actor role: Energy Guard / EMS 3. Support all mandatory scenarios from the following Use Cases: ○ LPC – Limitation of Power Consumption ○ LPP – Limitation of Power Production 4. Generate and send: ○ Active power limits (LPC/LPP) 5. Implement Use Case-specific SPINE flows, message structures, actor bindings, and state machines.
Reference Files	The EEBUS website offers all required use case documents. They only require you to log in and go to the Downloads page. You need the following files: • EEBus_SHIP_TS_Specification_v1.0.1.pdf • EEBus_UC_TS_LimitationOfPowerConsumption_V1.0.0_public.pdf • EEBus_UC_TS_LimitationOfPowerProduction_V1.0.0_public.pdf • EEBus_SPINE_TS_ProtocolSpecification.pdf
Acceptance Criteria	1. Demonstrate complete EMS behavior for at least 3 controllable device types. 2. Show plan negotiation, power limit enforcement, and fallback handling. 3. Run scenario-based tests aligned with Use Case specs. 4. Correct state machine transitions and SPINE interaction logic must be validated.
Documentation & Testing	1. Open-source code and installation guide 2. Use Case mapping (per supported Use Case) 3. Sequence diagrams and SPINE flows

	4. Test logs for each Use Case and scenario 5. Overview of actor-role bindings and capabilities

3.3.2 WP3B – SHIP and SPINE Implementation (Device-side)

Description	Implementation of SHIP and SPINE on a flexible device (FEID) such as EVSE, battery, or HVAC. The device must receive and respond to EMS signals as defined by the relevant EEBUS Use Cases.
Technical Requirements	1. Languages: C/C++, Rust 2. Implement EEBUS actor role: Controllable System 3. Support all mandatory scenarios from the following Use Cases: ○ LPC – Limitation of Power Consumption ○ LPP – Limitation of Power Production (if applicable) 4. Respond to: ○ Active power limits (LPC/LPP) 5. Maintain required SPINE message interfaces, state machines, and scenario flows. 6. Implement fallback behavior, heartbeat handling, and plan acceptance logic.
Reference Files	Sames as WP3A
Acceptance Criteria	1. Device responds to EMS instructions with correct timing and logic. 2. Supports all required Use Case message sequences. 3. Handles failsafe behavior and reverts under communication loss.
Documentation & Testing	1. Integration guide and message API documentation 2. Sequence charts per Use Case 3. List of supported scenarios and message types 4. (Simulated) EMS test results

3.3.3 Common Requirements (for WP3A and WP3B)

The following protocol-level requirements apply to all EEBUS implementations across WP3A and WP3B.

SHIP (Secure IP Communication)

All components must implement:

- TLS 1.2+ with mutual authentication (mTLS)
- PIN-based certificate pairing process
- mDNS-based device discovery and announcement
- Lifecycle and key management (pairing, removal, rekeying)

- Encrypted WebSocket framing as transport

SPINE (Semantic Communication)

- Only implement SPINE messages defined in selected Use Cases
- Follow all role-specific message flows and cardinality rules
- Implement required state machines per actor
- Use dynamic feature binding and subscription mechanisms
- Reject or ignore messages outside defined Use Cases

3.4 WP4 – Local Modbus Converter

3.4.1 WP4A – Local Converter for Modbus Devices

Description	A bridge that connects Modbus RTU/TCP assets to standard protocols (S2, EEBUS, or Matter). The converter must act as a virtual device: presenting itself as a Resource Manager in S2 and as an EEBUS responder using SPINE function sets, while communicating with a real asset via Modbus on the backend.
Technical Requirements	1. Suitable for embedded Linux or gateway-class devices 2. Configuration must support units, scaling, steps, procedures, etc.
Acceptance Criteria	1. Implements at least one protocol stack (e.g. S2, EEBUS and/or Matter) 2. All protocols (WP 1 t/m 3) should be able to run on the local converter 3. One protocol must be implemented using one of WP1 – 3 deliverables. 4. Configurable mapping between Modbus registers and standard fields 5. Data-driven mapping (no code changes for profile updates)
Documentation & Testing	1. Public Git repository for config/mapping files (will be created by ElaadNL upon request) 2. Example mapping profile for each supported asset type

3.5 WP5 – OCPP Integration

3.5.1 WP5A – OCPP "Light" Connector (HEMS-side)

Description	A connector that enables the HEMS to act as a minimal OCPP backend (CPMS) to communicate with local OCPP Controllers. It supports only the subset of messages needed for local energy control, and receives data from one or more downstream controllers.

Technical Requirements	1. Conforms to OCPP 1.6j, 2.0.1, and/or 2.1 depending on configuration 2. Works alongside or within the HEMS local controller
Acceptance Criteria	1. Supports sending the following OCPP messages: a. SetChargingProfile b. ClearChargingProfile 2. Supports receiving and handling the following messages: a. BootNotification b. MeterValue 3. Able to address individual charge points by ID 4. Gracefully handles status updates and failure reports
Documentation & Testing	1. Sample code that helps to integrates the OCPP connector in a HEMS

3.5.2 WP5B – OCPP Controller

Description	A local controller that emulates a virtual charging station from the HEMS perspective. It intermediates between HEMS-issued charging profiles and those from the CPO, prioritizing grid-level limits while locally balancing power.
Technical Requirements	1. Acts as a virtual EVSE from HEMS view 2. Transparent OCPP proxying with override capability 3. Maintains logs of control decisions and command origins
Acceptance Criteria	1. Supports OCPP 1.6j, 2.0.1, and/or 2.1 depending on configuration 2. Accepts and interprets HEMS-issued TxProfile or SetChargingProfile messages 3. Applies local profiles without violating active CPO profiles 4. Aggregates metering values from connected EVSE unit 5. Performs local power allocation based on available headroom 6. Ensures total site consumption remains within CPO-imposed envelope 7. Emulates one or more virtual charge points to the upstream HEMS 8. Supports seamless fallback to CPO-only control if local control fails
Documentation & Testing	1. Test cases for TxProfile reception, override resolution, and limit adherence 2. Logging format for metering and event traces

4. Development Guidelines

4.1 Non-functional Requirements

To ensure the HEMS and Connected Assets software works reliably in real-world deployments, it's not enough to focus solely on functional features. The software must also be secure, platform & architecture independent, scalable, maintainable, well-documented, and easy to update. Based on market feedback and technical sessions, we've identified the most important non-functional areas to address.

Best practices and standard workflows apply to all non-functional areas and will be enforced across all work packages.

4.1.1 Security

1. Support for low-resource devices: Must accommodate low-resource devices (e.g. embedded Modbus converters).
2. Protocol-native security first: All connectors must implement and adhere to the security features defined by the protocol they support (e.g. TLS in SHIP, digital signatures in S2). We do not impose additional security layers beyond those specified by the protocols are not required unless there is a clear operational need.
3. Secure by specification: The protocols used in this project (S2, EEBUS, etc.) already define encryption, authentication, and session integrity. Implementations must follow these specifications precisely and completely.
4. For components that include cloud communication (e.g. 4C variants), security measures such as authentication, encryption, and access control must be included in the proposal. ElaadNL will assess their adequacy during evaluation.

4.1.2 Quality

1. Acceptance criteria: Deliverables must meet the functional acceptance criteria as described in the work packages.
2. Automated testing: ElaadNL will facilitate a GitHub environment with CI minutes and support for automated pipelines.
3. Code validation and review: Follow industry standard practices including automated linting, CI builds, and peer-reviewed pull requests (internal)
4. Quality monitoring; ElaadNL will provide a shared quality assurance environment to track quality and security risks.
5. Versioning and traceability: All code must be versioned, with atomic commits and clear release notes, including changelogs to support traceability of updates and integration by third parties.
6. Code must adhere to industry best practices (e.g. PEP8 linting for Python), follow clear and consistent naming conventions, and avoid hardcoded secrets or environment-specific logic.

4.1.3 Documentation

1. Functional and integration overviews: Every connector or module must include, in English, a plain-language explanation of what it does, how it fits into the larger system, and what protocols or devices it supports. Visuals are encouraged.
2. Usage instructions with examples: Include step-by-step guides for installing, configuring, and using components. Provide realistic example configurations, input/output payloads, and expected behavior.
3. Contribution and governance guidelines: Since the code will be open-source, the repository must explain how others can contribute, raise issues, request features, or propose changes. The review and approval process must be transparent.
4. Versioning and changelogs: Documentation must reflect the current software version. Changes between versions must be documented clearly, especially when they affect functionality or compatibility. Semantic versioning must be used. Initially versions must stay within the 0.x range. The first major release will be at the end of the initial release.
5. Consistent terminology and format: To avoid confusion across different connectors and protocols, common terms (e.g. "asset", "setpoint", "control signal") should be used consistently. ElaadNL may provide a shared glossary or style guide.
6. Documentation repository and access: All documentation must be stored in the same version-controlled repositories next to the actual code (e.g. Markdown files in GitHub). No binary formats.
7. A template repository will be provided, for the aforementioned documentation.

4.1.4 Scalability

1. Modular design: Connectors should be built as separate, reusable modules that can be easily integrated into different HEMS platforms and device firmware.
2. Extensible design: Connectors should be built to allow for easy extension with additional functionality without requiring changes to the original deliverable.
3. Flexible deployment: The connectors must support deployment on both embedded platforms (e.g. microcontrollers in heat pumps or inverters) and on more powerful HEMS controllers (e.g. Linux gateways or local hubs).
4. Efficient use of system resources: This is especially important for the embedded side, where devices may have limited CPU, memory, and storage. The connector must run reliably in these constrained environments. This includes meeting all relevant security requirements as well.
5. Cloud-optional architecture: All components must support full local operation (with the exception of the WP1C cloud component).

4.1.5 Auditability & Logging

1. Log key actions: Connectors must log when a command is sent, when a response is received and when an important status changes. Each log entry must include a timestamp and a device identifier.
2. Use a clear format: Logs must follow a consistent structure that can be read and reviewed during testing and integration sessions.

3. Capture errors: Errors such as timeouts, rejected commands, or unexpected behavior must be logged in a format that supports effective analysis and debugging.

4.1.6 Maintainability

1. Keep components modular: Each connector should have a clearly defined scope. Small, focused modules are easier to test, update, and reuse across projects.
2. Minimize hidden complexity: Avoid tightly coupled logic, shortcuts, or opaque implementations that make future changes difficult. The goal is to make the connectors understandable and safe to modify.
3. Use third-party libraries sparingly. All dependencies must be well-maintained, widely used, and have permissive licenses (e.g. MIT, BSD, ISC, Apache 2.0). Avoid strong copyleft licenses such as GPL or AGPL. LGPL or MPL may be used only for clearly separated libraries.
4. Support long-term updates: Connectors should be designed to evolve as standards change. This includes the ability to add new protocol versions, fix bugs, and adapt to feedback without breaking existing installations.
5. Follow common practices: Use standard workflows for versioning, testing, and releasing. This allows others to contribute or take over maintenance when needed, even years after the initial release.
6. Prevent checking in IDE specific files (e.g. .idea, .vscode, etc.).

4.2 Open-Source

All software developed under this project must be released under the Apache License 2.0 to ensure that the software deliverables can be freely used, modified, and integrated into commercial products without legal complexity.

The goal of this project is to stimulate the market by providing open, production-ready building blocks that accelerate development and adoption. Apache 2.0 supports this by offering clear permissive terms, including explicit patent rights, while ensuring proper attribution and maintaining license integrity across implementations.

4.3 Documentation Templates & Repository

To promote consistency, openness, and long-term maintainability across all deliverables, ElaadNL provides a GitHub template as the starting point for each project repository. This template includes key files such as README.md, CONTRIBUTING.md, and SECURITY.md, ensuring that every repository starts with a clear structure for documentation, contribution, and responsible disclosure. By applying the same standards across projects, we make it easier for partners to collaborate, for third parties to adopt and build upon the work, and for the community to maintain a high level of trust in the quality and transparency of the software.

5. Project Approach & Milestones

5.1 Way of Working and Collaboration

The project follows an Agile methodology through regular sprints and iterative deliveries. Selected parties are expected to work independently where possible and align with the shared development process of the project.

A Technical Project Lead from ElaadNL will serve as the primary contact for all technical matters. A shared ticketing system will be used to submit, track, and document technical issues and inquiries. The Technical Project Lead will coordinate biweekly meetings to monitor progress, align on architecture and resolve integration or implementation issues. Collaboration should be transparent, proactive, and solution-oriented.

In parallel with the development work, the project aims to build a broader community of contributors and users for the Integration Phase. If possible, code should be shared publicly at an early stage to encourage early feedback and promote code reuse. The approach, timing, and platform for public sharing will be discussed and agreed upon with partners during the project.

5.2 ElaadNL's Role and Responsibilities

ElaadNL has a strong track record in improving technical connectivity and standardization for EV charging. It has since expanded its focus to other household assets and developed its own HEMS and device integrations in its lab environment. In this project, ElaadNL serves as coordinator and executor, without a commercial role in the HEMS or device market.

Area	Contribution
Coordination & Project Management	Drafts test scenarios, manages schedules and guides documentation; safeguards openness and standardization. Coordinates development across partners and has final authority on resolving technical issues.
Network & Ecosystem	Leverages its links with grid operators, research institutes and industry partners to involve key stakeholders. TNO, FAN and the Ministry of Economic Affairs and Climate are already engaged.
Open-source Support	Helps set up and maintain a public repository (e.g. GitHub) for connectors and tools to ensure continuity.
Test Facilities	Provides its EV & Smart Charging Test Lab for neutral demos and integration tests of devices and HEMS solutions.
Knowledge Sharing	Publishes anonymized findings to advance interoperability, respecting IP and commercially sensitive data.

5.3 Delivery Plan

Phase	Activity	Timeline	Milestones
1.	• Kickoff • ElaadNL sets up Git repo, SonarQube, and tooling per connector. • ElaadNL provides documentation, OpenADR connector and Resource Managers • Preparation time for partners to prepare project	Nov ’25	• Refined sprint planning teams • Refined delivery plan
2.	• Development in sprints including demo at end of every sprint	Dec ’25 - Mar ’26	• Code in ElaadNL repo (starting private)
3.	• Final demo of the developed software, demonstrating successful communication between HEMS and FEID across all three use cases	April ’26	• Implementation of standards validated • Usecases validated
4.	• Final documentation & handover	May ’26	• Code in ElaadNL repo (now public)

** See also the overall planning in Section 7*

5.4 Acceptance Criteria & Final Demonstration

Final acceptance of each work package is based on successful execution of end-to-end interoperability tests during the final demonstration. This includes demonstrating correct communication between the HEMS and the device using the selected protocol(s), and that control actions are executed correctly and reliably.

ElaadNL will manage the final demonstration and, upon completion, assume responsibility for ongoing coordination, maintenance, and updates. Parties are encouraged to indicate their willingness to remain involved after the project ends, e.g. by offering continued support or contributing to code maintenance on an offer basis.

<u>Practical Arrangements for Final Demonstration</u>

- Parties are expected to bring their own devices and necessary hardware to the test lab.
- The ElaadNL test lab provides electrical connections, test setups, physical space for demonstrations, and on-site support and technical expertise for setup and testing

Participants are responsible for bringing the necessary personnel and equipment to demonstrate their solution. For specific shared components such as the Modbus converter or OCPP controller, limited facilities may be made available, with details to be discussed on a case-by-case basis.

6. Participation & Eligibility

6.1 Eligible Parties and Roles

The RFP distinguish two primary categories of eligible parties:

1. Suppliers of HEMS solutions

   This includes HEMS vendors, EMS providers, manufacturers, and aggregators. These parties must demonstrate:

   - Proven experience with energy management systems
   - Integration capabilities with energy assets (e.g. batteries, inverters, smart meters, charge points)
   - Compliance with relevant standards (e.g. OpenADR, S2, Matter, EEBUS, or equivalent)

2. Suppliers of flexible energy-intensive devices (FEID)

   This includes manufacturers of batteries, inverters, EV chargers, or heat pumps. These parties must demonstrate:

   - A track record of delivering certified hardware
   - Availability of open APIs or documented protocols for integration
   - Support for secure remote communication and control

Participation per Work Package

For **Work Packages 1, 2, and 3**, Proposers must apply for the entire work package. This means the proposals must cover the full chain: a HEMS supplier, a flexible energy-intensive device supplier, and a party responsible for end-to-end software development, either in-house or through a contracted partner.

For **Work Package 4**, it is possible to apply for software development only, without a supplier; however, the proposal must include a hardware module that can be independently connected via Modbus.

For **Work Package 5**, it is possible to apply for the HEMS side only. Software development may be performed by an in-house team or a partnering software company. ElaadNL will provide a charge point device compatible with OCPP 1.6j, 2.0.1, and/or 2.1.

Note on Consortia and Matchmaking

Standalone software development companies are not eligible to apply independently. They must be part of a consortium that includes a qualifying HEMS and/or device Proposer, who will take responsibility for functional integration and field applicability.

If you are seeking a partner, you may indicate this in your proposal. ElaadNL may assist matchmaking by connecting interested parties.

6.2 Responsibilities of Selected Partners

Selected partners will be responsible for delivering their contribution in close coordination with the project. Responsibilities include, but are not limited to:

- Actively participating in planning, coordination, and progress meetings.
- Providing a single point of contact for the project.
- Delivering agreed-upon components or services according to timeline and specifications.
- Ensuring technical compatibility and integration with the described standards and architecture described in this document.
- Providing timely documentation, support, and updates during development, testing, and deployment.
- Participating in the integration phase and interoperability test phase after completion of this project.
- Committing resources for issue resolution, bug fixing, and optimization throughout the project.

7. Planning

	Description	Deadline
1.	Publication of RFP	Thu. 24 July 2025
2.	Deadline for Proposers to submit questions	Tue. 26 August 2025
3.	Written responses to Proposer questions	Mon. 1 September 2025
4.	Submission deadline for proposals	Fri. 19 September 2025
5.	Notification to all Proposers of results	Fri. 26 September 2025

Submitted questions will be answered no later than 1 September 2025.

ElaadNL reserves the right to adjust this schedule. In such cases, all Proposers will be informed accordingly. Proposals must be submitted by email to Marisca Zweistra at **rfp@elaad.nl** no later than **Friday 19 September 2025, 17:00 CET**. Late submissions will not be considered.

8. Proposal Questionnaire

1. Provide contact details and a brief summary of all parties involved in the proposal.

2. Which work package(s) are you submitting a proposal for?
   - *List the work package, protocol, and the type and model of the HEMS and/or device*

3. Describe your approach for the first sprint for preparation for the development phase.
   - *What will you do?*
   - *What needs to be set up?*
   - *What support or input do you expect from the project team?*

4. Give an estimate of the two-week-sprints you expect to need to develop the work package and describe the deliverables per sprint

5. Which programming languages do you plan to use? If known, please also list any key libraries, framework, or tools you expect to use.

6. Describe your approach how you will deal with the described non-functional requirements: Security, Quality, Documentation, Scalability, Auditability, and Maintainability (See Section 4.1).

7. Please list the key team roles and briefly describe each team member's relevant experience. (Names of members are not required)
   - *Include roles such as developer, architect, tester, etc., indicate experience level (e.g. junior, senior, years of experience), and dedicated time per week (in hours)*

8. Provide two reference projects that demonstrate your team's relevant expertise.
   - *Briefly describe the project, your role, and technologies used.*

9. Provide a cost breakdown for an all-inclusive price per phase (Section 5.3)
   - *Please include all expected costs (e.g. development, project management, testing, coordination).*

9. Evaluation & Selection

Proposers are asked to make an offer (free format) based on the criteria outlined in this document, and, in particular the questions listed in Section 8. The evaluation will be conducted by a panel consisting of representatives from ElaadNL, FAN, and selected domain experts.

	Evaluation Criteria	Weight
1.	**Technical Approach & Planning** • Description of approach to non-functionals (see Section 4.1). • Description of preparation sprint (see Section 5.3). • Description of proposed timeline with development sprints and expected deliverables per sprint (see Section 5.3).	25%
2.	**Team & Expertise** • Description of team, roles, seniority and dedicated FTEs. • Two relevant references of similar projects.	25%
3.	**Pricing & Funding** • Cost breakdown per phase described in Section 5.3. • Any in-kind contributions or co-funding (if applicable)	50%

All proposals will be evaluated based on the following criteria:

- Alignment with the project's objectives and technical requirements.
- Completeness and clarity of the answers provided.
- Quality of responses, assessed for clarity, conciseness, transparency, and practical feasibility.
- Each answer will be evaluated as follows:
    - Full points for clear, complete, and relevant answers.
    - Partial points for incomplete or partially relevant answers.
    - No points for missing or insufficient responses.
- For Pricing & Funding, prices will be benchmarked against the lowest offer submitted within the same work package.

The selection process will proceed as follows:

1. Initial screening to ensure eligibility and completeness. (Section 6)
2. Evaluation based on the criteria above resulting in a score.
3. Optional clarifications or interviews with parties, if needed.
4. Final selection and notification to Proposers (see Section 7).

10. Commercial & Legal Terms

10.1 General

- This RFP does not constitute any obligation or commitment of ElaadNL towards any Proposer. ElaadNL will select Proposers at its discretion and maintains discretionary freedom whether and with whom it will enter into an agreement.
- Termination for Convenience of any agreement after award of a WP is allowed as follows: ElaadNL may end the agreement with thirty (30) days' notice. Proposer is entitled to payment for its deliverables accepted up to the termination date.
- Any deviations from the requirements of the RFP must be clearly referenced and explained by Proposer.
- Initial proposals received after the closing date will NOT be considered.
- Where any offered deliverable differs from the requirements, it should be clearly identified by Proposer as such since ElaadNL will in general assess and compare only those proposals determined to be substantially compliant with the requirements of the RFP.
- Notwithstanding any other provision or comment herein, ElaadNL may, at its discretion, waive any non-conformity or irregularity in a submission.
- After the closure of the RFP, ElaadNL may request additional information, clarifications and/or verifications with respect to any item contained in the proposal. Proposer shall endeavor to respond as quickly as possible to any such requests.
- To assist in the assessment and comparison of proposals, ElaadNL may also seek the attendance of Proposer at meeting(s) to be held at ElaadNL' offices or other locations.
- Proposer's proposal shall lead to a contract with ElaadNL, and all commitments made by Proposer in the proposal and during the RFP process will, if acceptable to ElaadNL, be included in the agreement between ElaadNL and Proposer for such WP.
- This RFP and any awards to Proposers shall be governed by Dutch law. All disputes shall be submitted exclusively to the District Court of Gelderland, location Arnhem.

10.2 Pricing & Payment

- The project operates on a fixed-price model for each project phase, with deliverables and milestones defined in Section 5.3. If a Milestone is delayed by >10 calendar days, the parties meet within five (5) working days to agree a recovery plan. Failing agreement, ElaadNL may terminate the WP allocated to the Proposer for the remaining scope without liability beyond Section 10.4.
- 30% of the total amount will be retained until final acceptance of all deliverables following successful demonstration and documentation (see Section 5.4).
- Testing at the ElaadNL Lab is free of charge.
- All parties are responsible for their own costs, including development, travel, internal validation, and participation in test events.
- Partial payments will be issued following successful completion of agreed testing milestones and sprint demonstrations.

10.3 Intellectual Property & Licensing

- All software deliverables must be released under the Apache 2.0 License.
- Proposers must confirm their commitment to open-source development and agree to publish their contributions under this license.
- All intellectual property (IP) rights in the developed software will be transferred to ElaadNL including all source code, documentation, and related materials created under this RFP.
- ElaadNL will manage the official Git repository and ensure long-term public availability.
- The Apache 2.0 License allows for free use, modification, and integration by third parties, while preserving attribution and legal clarity.
- Proposer indemnifies ElaadNL and any future users of its deliverables against third-party claims alleging IP infringement by the software, covering reasonable legal costs and damages.

10.4 Public Availability & Handover

- Final delivery must include working, versioned code, documentation, and integration into public Git repository as specified in Section 4.3.
- Code must pass quality checks (e.g. SonarQube, CI) and be accompanied by usage examples and changelogs.
- All deliverables must remain publicly accessible after project completion.

10.5 Limitation of Liability

- All open-source contributions are provided “as is.” ElaadNL is not liable for any direct or indirect damages resulting from their use or integration.
- Proposer’s cumulative liability for direct loss is limited to the greater of 125% of the actual payments by ElaadNL.
- Proposer is not liable for indirect or consequential loss (e.g. loss of profit, loss of data) unless in case of gross negligence or willful misconduct attributable to Proposer.
- ElaadNL shall not be liable for any damages arising out of or related to this RFP or the Agreement, save to the extent such damages are the direct result of ElaadNL’s willful misconduct or gross negligence. Under no circumstances shall ElaadNL be liable for indirect or consequential loss.

10.6 Funding Contingency & Withdrawal of RFP

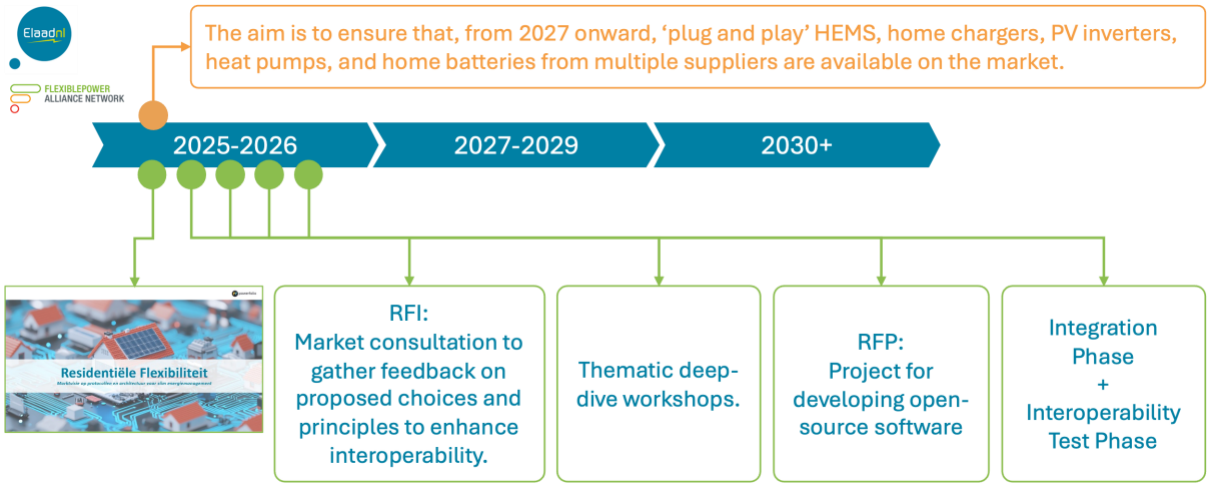
ElaadNL may, in good faith and upon written notice, withdraw this RFP or suspend/terminate the RFP and agreement after award of a WP to a Proposer, wholly or partly, if expected public or grant funding is not secured or materially reduced. In such case ElaadNL reimburses only documented, reasonable costs that: (a) relate directly to Deliverables already accepted in writing; and (b) cannot be mitigated or re-purposed. No further compensation shall be due.

Appendices

A. Project Background and Goals

A.1 Context and Motivations

A key barrier to unlocking residential flexibility is the lack of interoperability between devices. Products from different manufacturers often lack compatibility. ElaadNL and FAN aim to ensure that by 2027, a broad and appealing range of interoperable devices will be available on the market. To achieve this, a multi-year program was launched in 2024, as outlined in the figure below.



This initiative is carried out by ElaadNL on behalf of the Dutch grid operators and is part of the National Grid Congestion Action Program (LAN). Its goal is to improve the interoperability and controllability of flexible energy-intensive devices, thereby enabling residential flexibility.

The first step-mapping the protocols and architecture for residential flexibility was completed in early 2025 with the delivery of a report. This was followed by a Request for Information (RFI) and a series of thematic deep-dive workshops. Building on this groundwork, this Request for Proposal (RFP) initiates the implementation of an approach to improve the use of communication protocols for devices in and around the home.

Key considerations shaping the scope and approach of this RFP:

- **Wide variety of available protocols.** In practice, many different communication protocols are used to control and coordinate household devices, requiring a HEMS to integrate at least ten protocols to achieve basic interoperability.

- **Need for a defined protocol set.** Market feedback indicates that selecting a small, well-defined set of protocols is essential for covering communication to and within the home.

- **Proposed selection.** This RFP narrows the field to one protocol for communication to the home, four protocols between HEMS and devices, with at least one applicable to flexible energy-intensive devices, favoring internationally adopted, existing protocols.

- **Future-proofing and backward compatibility.** Many current protocols cannot expose the full flexibility potential of devices. The selected protocols are future-proof for new devices and provide a path to improve interoperability of the existing installed device base.

- **Market collaboration.** Industry stakeholders have expressed willingness to advance interoperability while safeguarding commercial interests. The approach relies on open-source development and shared test facilities.

- **Agile approach.** Market parties recommended enabling early testing and step-by-step open-source development.

- **Three core use cases.** Improved interoperability must support residential flexibility for consumers, market actors and grid-operators. Therefore, message flows and end-to-end tests will address the following:

    1. control based on available grid capacity (network objective),
    2. control based on dynamic electricity prices (market objective), and
    3. control to optimize the use of self-generated energy.

A.2 Urgency

The following developments demonstrate the urgency to have commercially available HEMS solutions that are interoperable with multiple products and brands, and that can process external signals:

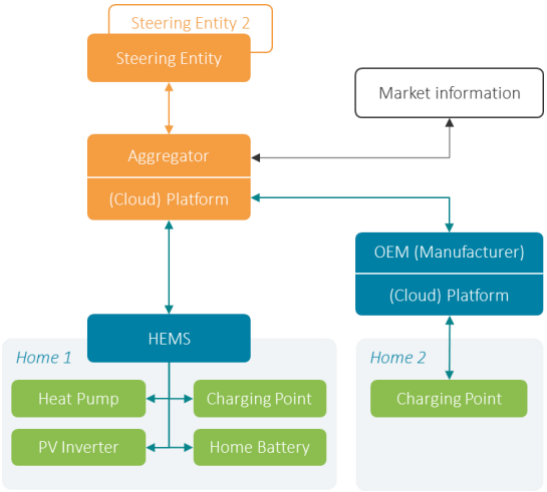
- A growing number of energy contracts with dynamic tariffs
- The discontinuation of net metering (*salderingsregeling*) and introduction of feed-in charges for self-generated power
- The need to mitigate looming congestion on the low-voltage grid
- The introduction of an alternative network-tariff system for small consumers

As noted in Section 2.2, the project focuses on three use cases that reflect the above-mentioned drivers for residential flexibility. Of course, a HEMS can support many other use cases, both now and in the future, such as optimizing comfort, monitoring energy usage, and balancing household loads.

A.3 General Architecture

This document provides detailed architectural overviews of the specifications for components to be built and tested and describes the selected protocols and devices used in different use cases. All components fit in the general architecture, which consists of:

- Three layers: steering entity, aggregator, and home.
- Steering entities: parties such as DSO, TSO and BRP, responsible for managing capacity and/or balancing load through information or control signals.
- Market-driven flexibility: an aggregator (or another market entity) receives these signals and translates them into control actions directed at households. Examples include dynamic energy tariffs and standardized signals to temporarily lower a household's capacity limits.



- In-home coordination: A HEMS coordinates the response of available flexible devices to control signals.
- HEMS implementation: functionality can be implemented through a physical device in the home, through the cloud (with each device connecting separately), or as a hybrid of both.

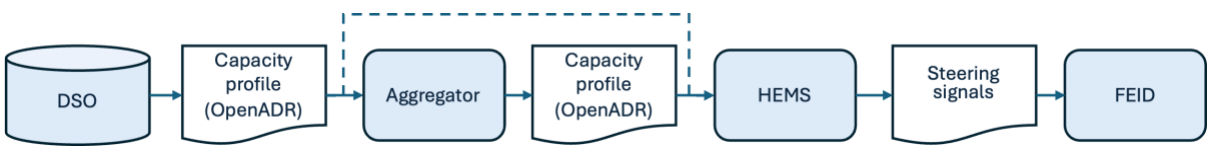
A.4 Key Use Cases

For this project, we selected three key use cases for testing. By developing open-source connectors, additional use cases can be added in the future.

1. Limiting Peak Grid Demand

The first use case focuses on limiting grid capacity. The Dutch power grid experiences constraints due to demand peaks, especially during winter. By forecasting grid load, the DSO can calculate a capacity profile that defines upper limits for both feed-in and usage. The HEMS, as the home's central controller, receives these messages and uses the available device flexibility to always keep the household connection within specified limits at all times.

In this scenario, the capacity profile is sent either (a) from the grid operator to an aggregator, and then to the HEMS; or (b) directly from the grid operator to the HEMS. In both cases, the capacity profile is sent through an OpenADR open-source connector provided by ElaadNL.



2. Dynamic Tariff Optimization

The second use case centers on control based on dynamic electricity tariffs. The HEMS can optimize a household's energy costs by, for instance, storing cheap power in a battery or as heat in a hot water buffer allowing the home to draw on its own reserves during expensive periods. The guiding principle is to avoid unnecessary consumption when prices are high and to exploit cheaper periods whenever they occur.

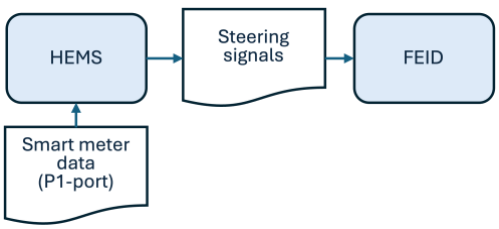
In this scenario, dynamic tariffs are provided via proprietary APIs (e.g. ENTSO-E) and communicated to the HEMS. Currently, there is no widely adopted open standard for accessing or processing dynamic tariff data. Based on feedback from the earlier RFI, market parties indicated that developing a new standard is not necessary at this stage, and proprietary solutions are currently considered sufficient.

3. Optimize the Use of Self-Generated Energy

The third use case focuses on maximizing the consumption of generated solar power in the household. When generation exceeds immediate household demand, the HEMS can distribute surplus energy intelligently by e.g. charging a home battery and/or EV to increase self-consumption.

This not only improves the household's energy efficiency and independence, but also optimizes electricity costs and reduces stress on the local grid, especially during periods of high solar generation.

In this scenario, the HEMS monitors local production and manages energy flows to optimize the use of self-generated power, based on local production data, local usage data, and/or smart grid meter data.



B. Open-Source and Interoperability Testing

B.1 Development of Open-Source Software

ElaadNL aims to develop modular open-source software that translates control signals, both from within the home and from external parties such as grid operators—into concrete control commands for devices via a HEMS. This software must be easy to integrate into commercial HEMS products, enabling market players to unlock residential flexibility quickly and efficiently.

Our proposed approach includes:

1. **Start with core flexibility functionality.** Begin with a limited set of messages from a small number of flexibility protocols (within and to the home), allowing for rapid early progress.

2. **Conduct open-source development via one or more partners.** Message sets for the selected protocols will be developed in collaboration with parties willing to publish open-source software.

3. **Provide HEMS solutions and devices for testing.** Participating partners must provide existing or newly developed HEMS solutions and devices for end-to-end testing using predefined communication protocols.

4. **Ensure active technical participation.** Technical experts and engineers are expected to join test and demo sessions (on-site or remote) to validate interoperability in practice.

5. **Jointly develop and execute test scenarios.** ElaadNL will collaborate with participants and standardization bodies to design and carry out relevant test cases.

6. **Demonstration three defined use cases.** Use cases will focus on capacity limiting, dynamic tariff optimization, and optimization of self-generated energy use.

B.2 Testing and Demonstrating Interoperability

ElaadNL aims to accelerate the market development of interoperable, customer-installable (“plug-and-play”) solutions between HEMS and connected devices. To support this, selected parties will be invited to actively test and demonstrate their products in a controlled technical environment—such as the ElaadNL TestLab. Parties not involved in open-source development are explicitly encouraged to participate in the Integration Phase.

The expectations for test and demonstration phase include:

1. **Integration Phase via public availability.** All open-source software developed in this project will be freely available, enabling other parties to implement the protocols easily and cost-effectively.

2. **Interoperability Test Phase to support scaling.** Practical insights from protocol implementation will be shared with organizations such as NEN to support the development of national (mandatory) and ideally European standards.

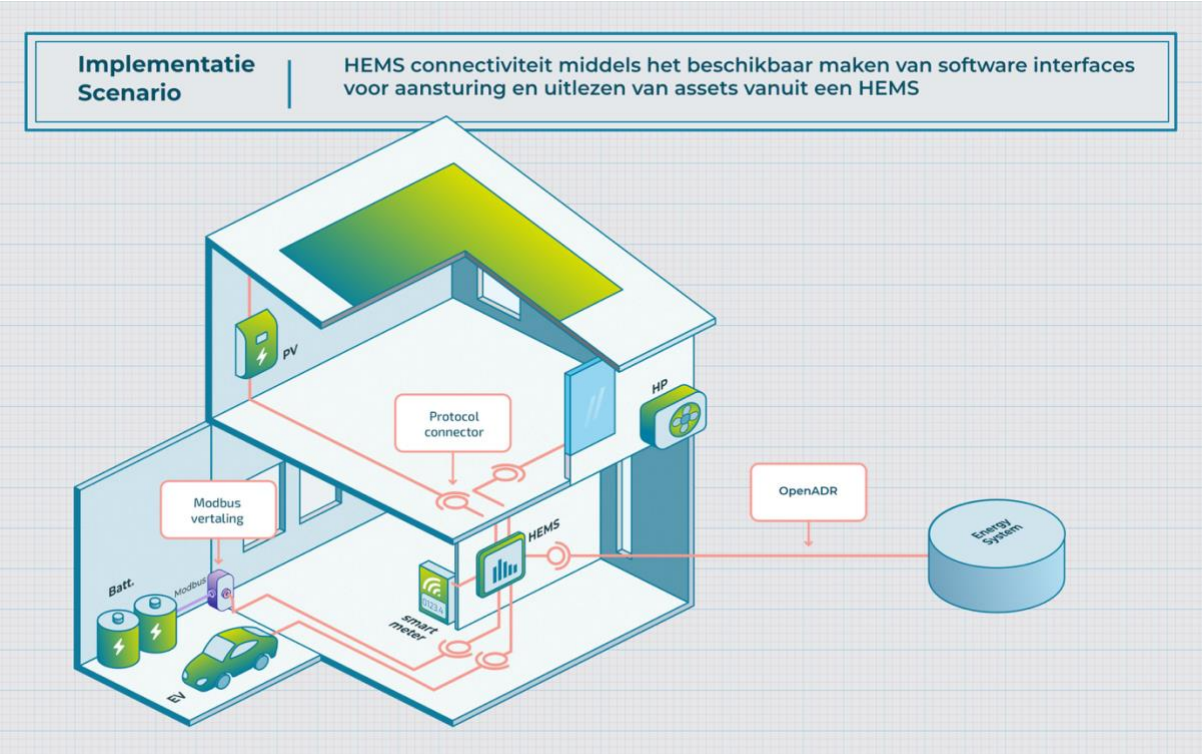
3. **Exploration of certification options.** The project will explore certification opportunities to ensure long-term compliance, potentially in collaboration with formal certification bodies.

This joint effort aims to deliver practical, and reliable implementations of interoperability that support a more flexible and stable energy system.

C. Implementation Scenarios

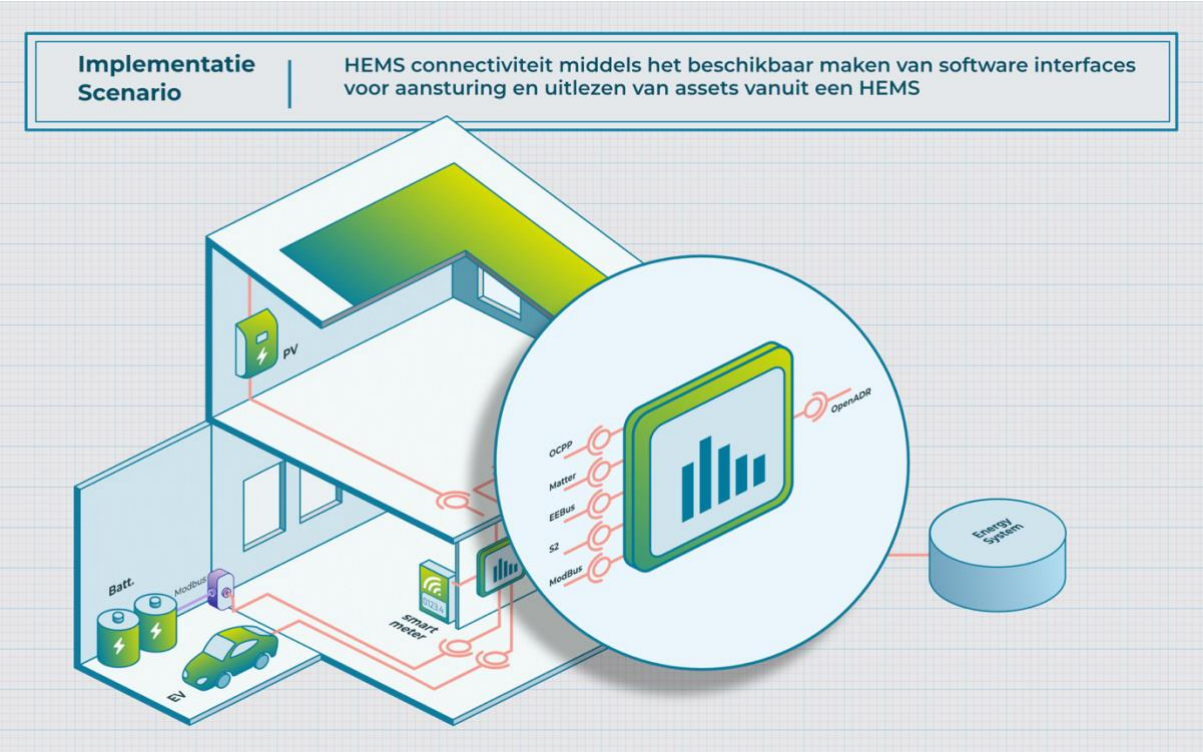
Several architectural approaches are possible for designing the control chain. No definitive architecture has been selected for the Netherlands—or for Europe—at this time. The goal of this project is to develop limited message sets for selected protocols are applicable across different architectural models.

This RFP focuses on the connectivity between the HEMS and the connected devices. For connectivity between the HEMS and the grid operator—whether directly or via an aggregator—the OpenADR 3.0 protocol will be used. ElaadNL will provide open-source software for this purpose, which HEMS vendors can integrate into their solutions. Alternatively, HEMS vendors may implement OpenADR support using their own software.



The referenced diagram illustrates functional communication between HEMS and devices. It intentionally does not differentiate between local and cloud-based control models. The HEMS serves as the central component and must support multiple protocols, as listed in Section 1.4. Devices may connect using one or more of the supported protocols.

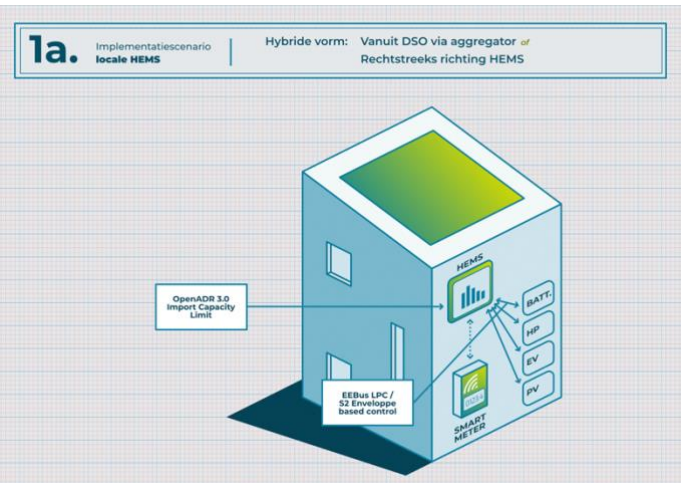
Another key function of the HEMS is translating energy system messages into device control schedules. The second diagram focuses on this specific HEMS role.



Note on Modbus The HEMS is unlikely to support Modbus directly. Due to the limitations discussed in Section D.1, devices using Modbus should instead interface with the HEMS via another supported protocol (e.g. S2, EEBUS, or Matter). Modbus is expected to be used only locally and translated at the device level, ensuring that all HEMS-facing communication follows a more secure and interoperable protocol.

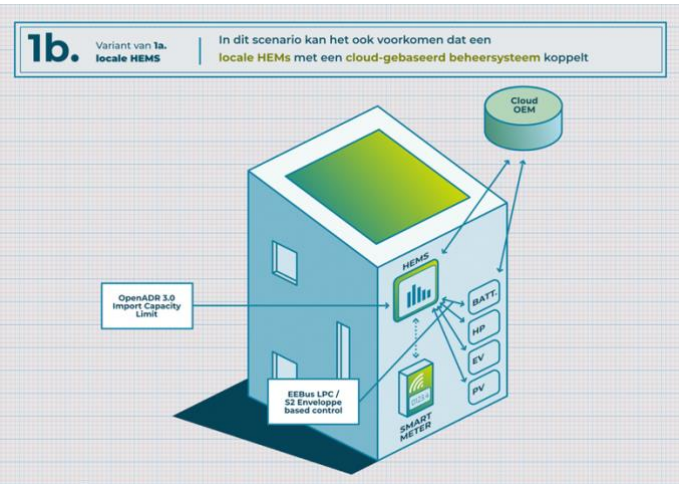
To illustrate local versus cloud flows, the following sections present several implementation variants and associated use cases.

C.1 Local HEMS



1a.

The first implementation scenario is based on a fully local HEMS setup.

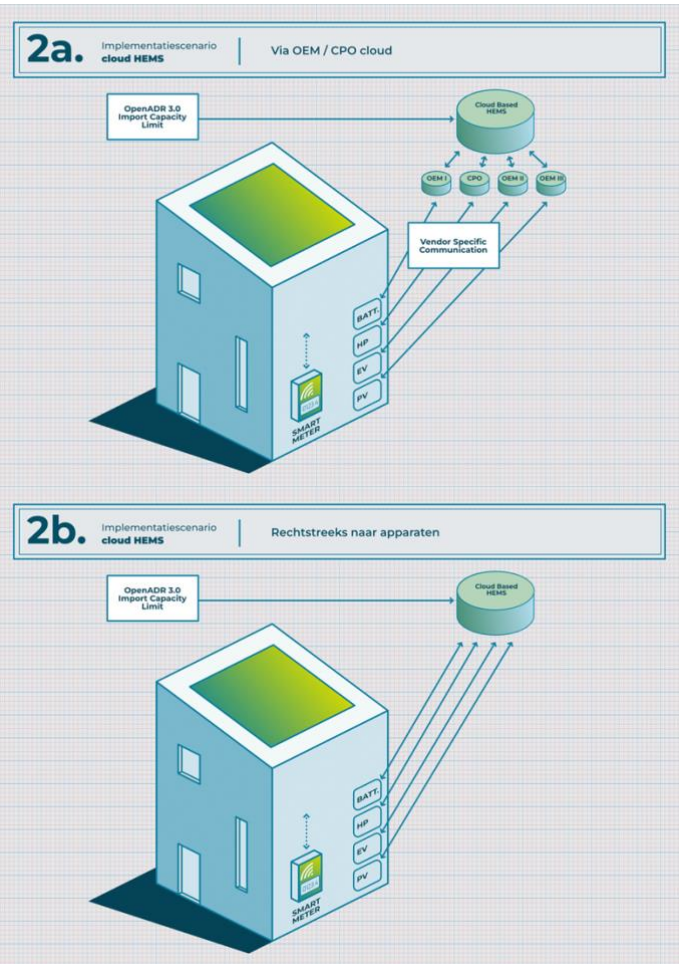


1b.

A specific variant of this involves communication from the local HEMS to a local control system managing one or more energy-intensive devices.

C.2 Cloud-based HEMS

Another implementation scenario involves a cloud-based HEMS architecture. Two control routes are defined below.



2a.

Communication from the cloud-based HEMS to a local control system that manages the devices.

2b.

Direct communication from the cloud HEMS platform to the individual device.

D. Technical Considerations

D.1 Modbus

Modbus has long been a widely accepted industrial standard for communication between devices such as sensors, control systems, and other industrial components. It is reliable, simple, and, most importantly, it works. There are two main variants: Modbus RTU, which uses serial communication, and Modbus TCP, which runs over IP networks.

Modbus TCP is increasingly popular due to its compatibility with modern networks. However, despite its technical reliability, Modbus lacks fundamental security features. Modbus TCP does not support any form of authentication. This means any device or person on the same network could, in principle, send commands to the system, posing a serious risk.

Although encryption is technically possible, it is rarely implemented in practice. As a result, communications are often readable and modifiable by anyone with network access, including hackers or malware.

This risk is amplified by the fact that an increasing number of flexible energy-intensive devices, such as EV chargers, heat pumps, and home batteries, support Modbus TCP. In a coordinated attack (e.g. as in the FrostyGoop[1] incident), malware could control multiple devices simultaneously, with potentially severe consequences for grid stability.

In theory, network segmentation (e.g. VLANs or dedicated subnets) could reduce this risk. In practice, however, such measures are rarely implemented in residential environments. Most households run everything on a single network, making it easy for malicious software to spread or abuse devices.

At the same time, Modbus is still widely supported by OEMs and manufacturers of flexible energy-intensive devices. It is therefore hard to ignore in any HEMS strategy.

Therefore, ElaadNL invites proposers to help develop secure ways to integrate Modbus into EMS/HEMS solutions in Work Package 4. Proposals may include:

- Middleware or proxy solutions that act as intermediaries between the HEMS and Modbus device, adding layers of security (e.g. authentication or filtering)
- Translation components that convert secure protocols (e.g. EEBUS or S2) into Modbus commands for legacy devices

[1] In the FrostyGoop attack, heating systems in 600 buildings were taken over via Modbus TCP. Legitimate commands were intercepted and replaced through compromised network devices such as routers and gateways, giving attackers full control over energy consumption and management in those buildings.

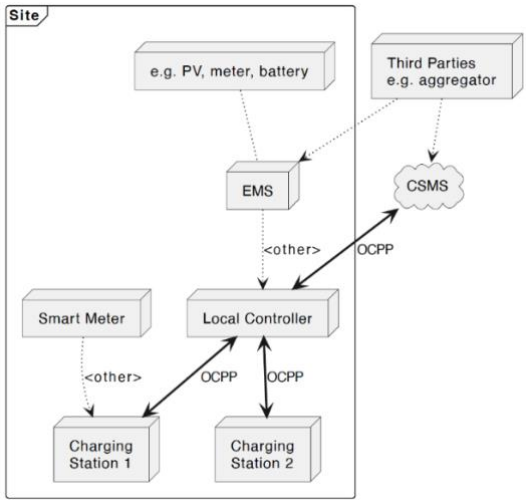
- Other solutions that support secure and practical use of Modbus in modern EMS architectures

The goal is to collaboratively develop a secure, practical, and future-proof Modbus integration within the broader HEMS ecosystem. We encourage open-source contributions or use of well-documented, existing standards wherever possible.

D.2 OCPP

For local control of EV chargers within an EMS (Energy Management System), we seek an open-source implementation of an OCPP 2.1 proxy.

This proxy should enable the injection of control signals and power profiles from the EMS into the communication between the EV charger and the CSMS (Charging Station Management System), without interfering with backend functions such as billing, authorization, or logging.



OCPP 2.1 includes a Local Controller feature in which the proxy acts as a middleware layer— it behaves like a CSMS from the perspective of the charger and like a charger from the perspective of the backend. Both connections use WebSockets with the same URI and charger ID. The proxy forwards messages in both directions and can also send OCPP messages (e.g. ChargingProfiles) directly to the charger, provided unique message IDs are used.

We are looking for an implementation that:

- Is fully compliant with OCPP 2.1, including WebSocket and JSON support
- Operates securely, using TLS server mode toward the charger and TLS client mode toward the CSMS, using custom certificates
- Supports the injection of EMS control signals and power profiles through a clear API (REST or Python interface)
- Is open-source under an Apache 2.0 license
- Includes comprehensive documentation, example code, and automated tests

The goal is to enable local control of EV chargers based on solar generation, battery status, or power limits, without disrupting the commercial backend functions (e.g. billing or management). The proxy should be lightweight, reusable, and easy to integrate into existing HEMS environments.

E. Technical Information

E.1 Cross-Protocol Terminology Glossary

Generic Term	S2 (EN 50491-12-2)	EEBus / SPINE	Matter 1.4	Notes / Usage in RfP
Home Energy Manager	**CEM** (Customer Energy Manager)	**CEM** (Central Energy Manager)	**Controller** (multi-admin Node)	Core orchestrator; receives grid signals, price signals; issues control instructions.
Device / Asset	**RM** (Resource Manager)	**SPINE Actor** (e.g. EVSE, PV, Battery)	**Node** (with Endpoints)	Controlled resource with flexibility; supports one or more control types.
Function	Control Type (PEBC, FRBC...)	Use Case + Data Point via SPINE Features/Functions	**Endpoint** (Device Type + Clusters)	Logical controllable capability (e.g. charge EV, curtail PV).
Constraint	PEBC.PowerConstraints, EnergyConstraints	LoadControlLimitListData, LoadControlEventListData / represented in SPINE LoadControl feature	DeviceEnergyManagementMode cluster (PA, CON)	Operational or grid-imposed envelope defining limits.
Instruction / Schedule	PEBC.Instruction (PowerEnvelope)	LoadControlEventListData / ScheduleListData equivalent — within LoadControl feature	**Energy Management Schedule** cluster	Optimization signal sent to device (e.g. when/how to operate).
Status / Feedback	InstructionStatusUpdate, ReceptionStatus	LoadControlStateListData, SPINE result objects	Cluster attributes (state, measurement)	Confirms instruction execution, rejects, or adjusts.
Measurement	PowerMeasurement, PowerForecast	MeasurementListData (real-time or forecast)	ElectricalMeasurement cluster	Real-time reporting; supports optimization and feedback loops.
Device Type	Declared via SystemDescription	**SPINE Actor Type** (e.g. EVSE, Battery)	**Device Type Library**	Used to express capabilities and bind control logic.
Topology	CEM ↔ RM over s2-ws-json	SHIP (transport layer) + SPINE (data model)	Fabric with Nodes, Admins, Clusters	Logical and transport-level structure for interoperability.
Transport Protocol	s2-ws-json (WebSocket/JSON)	**SHIP** over IP (WebSocket/TLS)	Matter IP stack (Thread, Wi-Fi, Ethernet)	Actual protocol stack—semantic alignment possible.
Security Domain	Session/Handshake with role/authentication	Device Binding in SHIP sessions	**Fabric** (shared credentials, ACLs)	Defines which controller can manage which device.
Energy Use Case	Control Type + Forecast + Instruction	SPINE Use Case Model + Features	Cluster interaction + Device Types	Examples: Grid capacity, dynamic price, self-consumption.
Interoperability Layer	**S2 semantic model**	**SPINE data model**	**Matter data model** (Clusters/Endpoints)	Defines data structure and intent for commands and feedback.