**Document title:** Arrowhead fPVN Deliverable D2.2

| **Version** | **Status** | **Date** |
|---|---|---|
| 1.0 | final | 2025-01-19 |
| **Author** | **Contact** | |
| Pal Varga | pvarga@tmit.bme.hu | |

# Deliverable D2.2
# Second version of fPVN microservice architecture – Arrowhead v5.0

Work package leader: Pal Varga
pvarga@tmit.bme.hu

Karl-Johan Gramner
karl-johan.gramner@bnearit.se

Editor: Pal Varga
pvarga@tmit.bme.hu

## Abstract
This document constitutes deliverable D2.2 of the Arrowhead fPVN project.

WP2 will develop the microservice paradigm-based platform for flexible production value networks. WP2 addresses the maturing of released Eclipse Arrowhead core services, provisioning of communication interoperability, autonomic identification of data-model non-interoperability properties, dynamic instantiation of data-model translation services and improvement of micro-service management based on policy and rule-based management.

Chips JU project 101111977 - Arrowhead FPVN
Project Coordinator: Professor Jerker Delsing | Luleå University of Technology

EUROPEAN PARTNERSHIP

# Table of contents

# 1. Introduction

## 1.1 The elements of Arrowhead fPVN deliverable D2.2

The deliverable D2.1 of Arrowhead fPVN consists of the following documents.

1) Deliverable D2.2 Main document (Arrowhead_fPVN_D2.2_v1.0)
2) Authentication Core System – SysD v5.0.0 (ah_authentication_sysd.pdf)
3) system-discovery – SD (system-discovery_sd.pdf)
4) service-registry-management – SD (service-registry-management_sd.pdf)
5) service-discovery – SD (service-discovery_sd.pdf)
6) device-discovery – SD (device-discovery_sd.pdf)

Furthermore, the Roadmap discussions and results can be found on github:

https://github.com/eclipse-arrowhead/roadmap/tree/main/5.0%20Draft

## 1.2 Overview of WP2 aims and tasks

WP2 aims to build, mature and extend a microserves/SOA-based platform to support flexible production value networks in heterogeneous and complex settings. It consists of four tasks, namely:

- Task 2.1 Maturing of the Arrowhead Framework
- Task 2.2 Autonomous handling of data access and non-interoperable properties
- Task 2.3 Microservice design methodologies for Industry5.0
- Task 2.4 Governance of microservice-based architectures

Their current advances are the following.

## 1.3 The recent proceedings of Task2.1

- **Framework Enhancements and Version Releases:** Successfully created the initial version of two necessary libraries on which all the core and support systems can depend on [1].
- **Roadmap Development for v5.0:** Through monthly meetings, improvements to the reference architecture of Arrowhead Framework is being discussed focusing on strategic enhancements for v5.0. This includes the development of the General System of Systems Description (GSoSD) and System Description (SysD) documents for critical systems like Service Registry, Authorization, and Service Orchestration.
- **Core System Upgrades:** Successfully created the Service Registry core system v5.0 [1].
- **Stakeholder Engagement and Commercial Strategy:** Conducted significant meetings in Vienna and Luleå, setting the stage for commercializing Arrowhead Framework components and strategies.

There were no significant deviations reported from the original plan for these 6 moths. No major new risks have been identified but it remains a challenge to get use-case adoption of the Arrowhead Framework to fully verify that the framework meets industry needs.

The task has made substantial progress in advancing the Eclipse Arrowhead framework, incorporating key updates and strategic planning to ensure its leadership in IoT systems integration and service orchestration. We sustained emphasis on strong implementation and active stakeholder engagement, as this is essential for maintaining progress and mitigating potential challenges.

## 1.4 The recent proceedings of Task2.2

Task T2.2 aims to develop concepts and solutions for managing data in non-interoperable SOA environments and addressing security and privacy concerns in microservice architectures. The task involves examining updates to the Eclipse Arrowhead architecture and its core systems to facilitate the dynamic instantiation of translation services or interoperability adaptors.

In order to decide on a provider system, the orchestration is supposed to aim for the best possible option, even considering QoS. Hence, QoS capabilities between the consumer and provider needs to be monitored. In this period, the QoS monitoring specification has been created, and the first proof-of-concept has been implemented.

In this period, stakeholders have designed and developed translators in WP3 for which WP2 integrated and AI-based one with the framework. Moreover, to support autonomous operation, the Arrowhead co-pilot was demonstrated and further improved [5].

A focused evaluation of the Eclipse Arrowhead architecture has been carried out in this period, to identify necessary updates for enabling dynamic services and addressing non-interoperable properties. Based on the assessment findings, the development of interoperability solutions is essential, with ongoing discussions in the roadmap group meetings and between sessions. As a solution, we must create scalable and adaptable adaptors or translation services in collaboration with WP4 to support diverse environments.

For autonomous CPSoS, implementing robust security measures and privacy safeguards is of key importance. This involves integrating advanced cryptographic techniques and comprehensive access control mechanisms to ensure data integrity across both local and global fPVN clouds. The elaboration of related concepts has commenced during this period.

## 1.5 The recent proceedings of Task2.3

Task T2.3 is dedicated to investigating cyber-physical systems-of-systems (CPSoS) and industry-driven design patterns within microservice-based architectures. The aim is to create conceptual tools and design strategies that facilitate the integration of these elements into specific use-cases. This task also evaluates the role of human interaction within microservices/SOA-based System-of-System architectures. It strives to advance the notion that microservices are vital in the data exchange processes characteristic of Industry 5.0. The responsibilities of this task include developing microservices tailored for Industry 5.0 CPSoS architectures, assembling a design methodology guide for implementing microservices in industrial settings, and offering technological support and expertise for industrial pilot projects.

In the previous period, requirements have been collected from WP3, and collaborations with the use cases involving WP3 and WP4 are going on successfully.

One of the main objectives of Task 2.3 is to design and develop design patterns for microservice-based architectures in CPSoS for Industry 5.0. The initial efforts have been focused in the following areas:

(i) Microservices for constructing CPSoS architectures in Industry 5.0 are being developed as part of the Eclipse Arrowhead v5.0 roadmap. The partners have begun designing microservices that meet the specific needs of CPSoS architectures, aiming to provide fundamental components that facilitate seamless integration and data exchange within complex industrial systems.

(ii) The creation of the Arrowhead Domain Specific Language (DSL) is a newly risen subtask, necessitated by the requirements. The SysML profile which constitutes the DSL has in a first round successfully been integrated into the Eclipse Papyrus tool. The DSL as implemented in Papyrus has been used in a course for last year Master students and professionals (LTU, D7065E). They have very successfully used the Arrowhead DSL to create architecture black box and partly white box modelling of real world use cases. The student feedback is very positive and encouraging for further development of the DSL as an engineering tool for professional use.

Apart from bug fixing, the plan for further development addresses additional phases in the Eclipse Arrowhead engineering process. Additionally, the integration into Papyrus has to be updated and extended to reduce the number of engineering step necessary to create local clouds and systems of local clouds.

(iii) As part of the roadmap activities, targeted discussions are ongoing on elaborating the Arrowhead fPVN cookbook. This is supposed to detail design methodologies for microservices-based CPSoS - aiming to offer practical advice to developers and architects in creating microservice-based solutions. The definition of the GSoSD already marks a significant step in this direction.

(iv) The definition of the reference GSoSD for the v5.0 of the recommended core systems is almost finished. This document outlines the fifth generation of the Arrowhead Core systems, describing the high-level architecture and the functionality these systems provide, crucial for most Arrowhead application scenarios. It abstractly details the architectural elements without endorsing specific implementations or technologies.

Finally, Task 2.3 keeps providing comprehensive technological support to the industrial pilots using microservices. This includes technical assistance, training, and continuous consultation. Communication channels have been established to facilitate this support activity.

## 1.6 The recent proceedings of Task2.4

Task 2.4 is this period focused on governance methods for both design-time and run-time stages, aligning with the current engineering status of most use cases, which remain in the design phase. Particular attention has been given to developing a system enabling the Arrowhead organization to publish official machine-readable specifications for implementation by various solutions and tools. At runtime, this system will allow engineers to access a live Arrowhead cloud without requiring coding or configuration. While the system is ready for demonstration, additional refinement is needed before it can be made available to project partners. Providing verifiable identities for service interfaces, deployed systems, and vendors serves as a foundation for other governance methods, policy enforcement, and broader management capabilities.

No major deviations from the plan have been noted.

# 2. WP2 Objectives

WP2 aims to fulfil the General Objective #2 of the Arrowhead fPVN project, namely creating and maturing a microservices/SOA platform that enables of dynamic deployment and autonomous utilization of information translation in PVNs.

The main objectives of WP2 are:

- Maturing of released Eclipse Arrowhead core services to TRL7-8.
- Defining an innovative microservices based architecture, meeting a comprehensive assessment of performance KPIs.
- Defining a set of solutions for middleware virtualization
- Provisioning of communication interoperability through new, updated and extended protocol translators and legacy technology adaptors targeting TRL6-7
- Provisioning of autonomic identification of data model non-interoperability properties
- Provisioning of dynamic instantiation of data model translation services (from WP4)
- Provisioning of improved microservices management based on policy and rule-based approach.

| Objectives | Work towards achieving objectives |
|---|---|
| Microservices/SOA enabling of dynamic deployment and autonomous utilization of information translation in PVNs | Eclipse Arrowhead v5.0 roadmap is solidified. The GSoSD definitions for v5.0 is an active document, serving as the basis of v5.0-related design and development for microservice creation and refinements, and CPSoS creations. WP2 also worked together with the technical work packages WP3 and WP4 in order to create a proper interoperability platform, allowing translations and adaptions for various major digital data models and languages.<br><br>Maturing of released Eclipse Arrowhead core services to TRL7-8 is ongoing as the Arrowhead reference implementation gets updated in GitHub.<br><br>Defining an innovative microservices based architecture, as Eclipse Arrowhead v5.0, first concept definition is set by the GSoSD document and related documents – part of D2.1.<br><br>Provisioning of communication interoperability through new, updated and extended protocol translators together with WP3 and WP4. The architectural connectors for such adaptors (even including AI-powered systems) are provided by WP2, best practices are explained and disseminated in Arrowhead bi-weekly meetings, workshops, and the Wiki.<br><br>The first version of the Arrowhead Co-Pilot is operational, and can be used in friendly environments. |

# 3. Recent results on maturing the Eclipse Arrowhead framework

## 3.1 Specification

In this period, roadmap meetings were continued to enhance the initial specifications. During this work, we have further detailed the conceptual changes presented in the previous deliverable. The results of these discussions are additional documents: the system descriptor of the new Authentication core system and four service descriptions for services provided by the Service Registry core system.

Furthermore, the conceptualization of Quality of Service support in the Arrowhead v5.0 has also been started and resulted in a proof-of-concept description.

AITIA actively participated in the joint Deep Tech workshop – involving Arrowhead fPVN and the AIMS5.0 project –, in Budapest 2024 November. During the event, the coming changes were presented and a short demonstration was showed about the current state of Service Registry 5.0.

## 3.2 Reference Implementation

During this time, the development of the reference implementation has been started. We are using Java 21 programming language with Spring Boot framework. There are some parts that we could reuse from the previous Arrowhead version, but most of the implementation is written from scratch, because of major conceptual changes and the obsolescence of the existing source code.

In *development phase 1* we produced the initial version of two necessary libraries on which all the core and support systems can depend on. In *development phase 2* we created the Service Registry core system, however the testing and documenting is still an ongoing task. Related source codes are accessible on the following links:
https://github.com/Aitia-IIOT/ah5-common-java-spring/tree/development
https://github.com/Aitia-IIOT/ah5-core-java-spring/tree/development

Along with the system developments a new website is under construction as well to provide documentations, tutorials, examples and other useful information about the Arrowhead 5.0. reference implementation. The current state is accessible here:
https://aitia-iiot.github.io/ah5-docs-java-spring

In the reference implementation all system follow (or will follow) the same multi-layered design (see the figure below).

Every system has endpoints (**Service API**) that supports one or more protocols (**Protocol Handlers**). A request then must go through various filters (**Authentication**, **Authorization** and **system-specific**) to check access permissions and do preliminary tasks. The next step is the input **Validation** and **Normalization**. Then task-specific **Service Logic** is executed using the valid and normalized input parameters. If the task requires to store or retrieve data, the **Data Store** layer is involved as well.

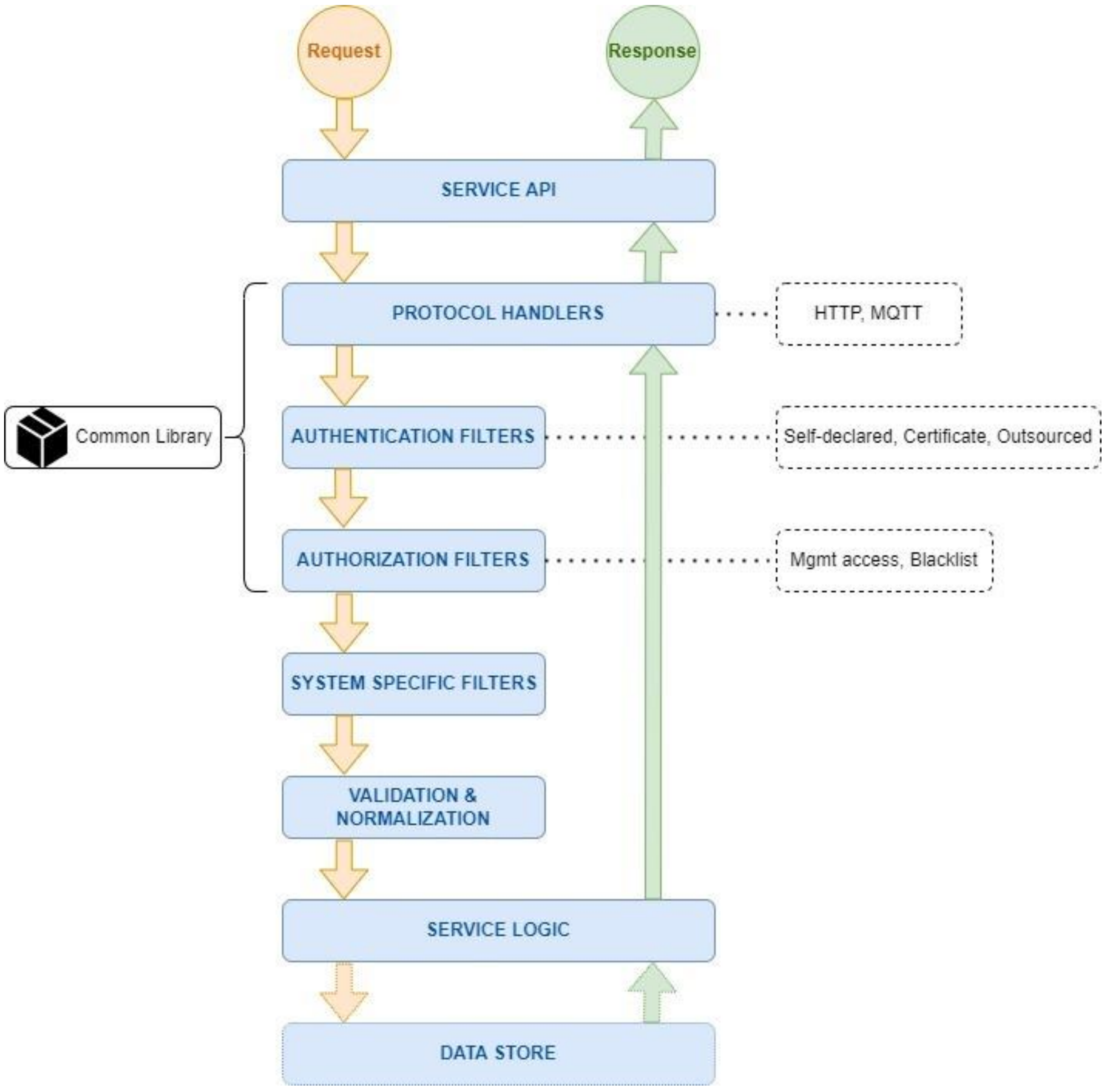

**Figure 3.1 General System Design**

The advantage of this approach is that the source code is easier to understand. Another benefit is that some of the layers are always the same regardless of the actual system, so we can extract those into a common library. Protocol Handlers, Authentication, Authorization filters are good examples for this. Also, commonly used validators and normalizers can also be offered by the library.

***arrowhead-data-transfer-objects library***

This library contains all the necessary data structures that can be used for communication between core, support and application systems.

***arrowhead-common-utils library***

As we mentioned before, this library contains all the data structures, algorithms and mechanisms that can be used in the implementation of multiple systems.

A lot of the planned systems use databases to store their data, so we provide a basic database support. Logging is also important to all systems: the library contains support to store log entries in a database, and also a service to retrieve them using various filters.

There is a system initialization mechanism that runs at startup and executes common tasks (system and service registration to the Service Registry for example) and also provides opportunity for the systems to run their custom tasks as well. Similarly, a shutdown mechanism is also available.

The library offers some data structures to represent systems, service instances, interfaces and operations. Some of these are general, but there are protocol-specific structures as well (currently HTTP and MQTT is supported).

We have introduced a mechanism to support the communication via MQTT. With this feature, in the actual system implementation the developer only has to provide one class per service with one method for each operation. In these methods the only real task is parsing the payload and passing it to the service logic layer.

As mentioned before, we provide various filters for endpoints that are using HTTP or MQTT protocols. The Authentication filters support three authentication strategies: self-declared, when the requester identifies itself (HTTP, MQTT); certificate, when the requester provides an X.509 certificate (HTTP is available, MQTT is not implemented yet), and outsourced, when we use the Authorization system to check the token the requester provides (not implemented yet). The Authorization filters will use the Authorization core and the Blacklist support systems to check authorization (not implemented yet). Some development filters are also implemented that offer detailed logging.

The Arrowhead core and support systems often consume other system’s services typically using HTTP protocol, so a convenient HTTP client is also available, as well as a service collector that helps to get access information of the needed services by contacting the Service Registry or Service Orchestration systems.

Metadata representation in Arrowhead 5.0 is much more sophisticated than it was in the previous versions: complex data structures with embedded objects and lists instead of simple key-value pairs. With this feature, the metadata requirement matching is also more challenging with multi-level keys and different operations. The library contains the necessary matcher that supports more than two dozen possible operations.

We offer support for handling pages in the services and also validators and normalizers for commonly used input parameters, such as address, name, page or log filters.

Arrowhead 5.0 introduces interface templates that describe how an actual interface should look like by specifying mandatory and optional properties. Validators can be assigned to every property. The library offers an extendable interface validator algorithm with various validators. The library also contains default security configurations for HTTP protocol, exception classes, an Open API-based browser UI support (Swagger) for systems, and various utility methods.

***arrowhead-serviceregistry system***

The Service Registry core system provides four services: *device-discovery*, *system-discovery*, *service-discovery* and *service-registry-management*. It will also provide the *monitor* service when the necessary specification is available. Consumers can choose between HTTP and MQTT interfaces.

The Service Registry represents multiple connected entities, such as *devices* (hardware components which are capable to host systems), *systems* (applications which are capable of providing and/or consuming services), *service definitions* (unique identifiers of services), *interface templates* (describe the structure of actual interfaces) and *service instances* (records that describe how systems provide services).

These entities and their connections are stored in a relational database (MySQL). This database has no connection to any other Arrowhead system. We use unique names for identify entities inside the Local Cloud.

The Service Registry implementation and package structure follows the General System Design. It realizes the Service API layer (using both HTTP and MQTT), system-specific filters, custom validation (one method per operation) and normalization (one method per operation), Service Logic layer and Data Store Layer.

The Service Registry is designed to be highly customizable. There are a lot of configuration possibilities available to adjust the system’s behavior. Operators can choose

- whether MQTT interface support should be enabled or not;
- whether the Service Registry should work independently or with a Service Orchestration system. Also, a list can be specified for systems that can always have direct access (without orchestration);
- whether the service instance responses should contain detailed system and device information or not;
- whether or not it is allowed to register services without existing interface templates;
- between the three authentication strategies: self-declared, certificate, and outsourced;
- whether the management endpoints should be authorized or not;
- if authorization is activated, which authorization strategy should be used (only system operator can use management endpoints or additionally listed systems can use them or beside these the Authorization system should be contacted as well);
- whether the Blacklist support system should be used during authorization;

# 4. QoS monitoring specification for Arrowhead endpoints

## 4.1 Mandatory monitoring capabilities

Each Arrowhead endpoint has an IP connection and an OS-level TCP/IP protocol stack. Using the functionality of these protocols, we designed QoS tests that can be executed on all AH endpoints over IP connections regardless of their hardware and software architecture.

**Protocol-based tests**

1. TCP SYN + TCP SYN/ACK + TCP RST response time test
2. High UDP port test: closed port connection attempt, to which the AH endpoint responds with an ICMP port unreachable message, thus measuring the response time during the message exchange
3. ICMP Echo test

During testing, a priority fallback mechanism is used; the test runs until the first successful measurement. So, if the TCP test is successful, the process does not switch to UDP or ICMP.

The test measurement is executed iteratively in a pre-defined number of steps. The number of steps is a configuration parameter of the QoS Manager. The purpose of iterative measurement is to allow the QoS Manager to calculate statistics from the results: minimum, average, and maximum round trip delay (RTT) and delay variation (PDV). In addition, if the iteration number is configured to a sufficiently high value, a realistic packet loss rate can be calculated from the tests.

The mandatory monitoring capability does not require specific protocols or message formats.

## 4.2 Augmented monitoring capabilities

The mandatory monitoring capabilities are only indirectly capable of providing load feedback on the internal resources (CPU, network adapter, memory) of the AH endpoint. Software support at the endpoint providing the service is required for a more granular, detailed endpoint monitoring. The concept is that if the operator of an AH service wants to extract detailed QoS metrics from the participating endpoints, support for this should be set up as a requirement for the endpoint operator (i.e., the provider). *Accordingly, the framework provides an easy-to-deploy endpoint monitoring application and a deployment guide to augment the Arrowhead endpoints' monitoring capabilities.*

The support consists of two components:

- Implementation of a dedicated control protocol for communication with the QoS manager
- Implementation of endpoint measurements

The AH endpoint provides Augmented QoS tests on a pre-defined TCP port.

The available augmented tests are divided into two groups:
- Mandatory test
- Optional tests

Adaptability and upgradability can be guaranteed by defining a specific protocol and message types between the QoS manager and the AH endpoint to control the augmented monitoring functions.

| Control task | Execution phases |
|---|---|
| *Execution of a periodic measurement* | 1. Negotiation phase: the AH endpoint reports to the QoS manager which optional tests it supports and which are currently enabled.<br>2. Organization phase: the QoS manager organizes the test sequence by sending the dynamic parameters to the client<br>3. Execution phase: the QoS manager starts the test sequence based on the information provided in Phase 1 and 2<br>4. Collection phase: the QoS manager collects the measurement results and calculates/updates the QoS statistics<br>5. Close measurement session |
| *Collecting periodic (measurement capability) reports* | 1. The QoS manager sends a QoS capability query message to the selected AH endpoint.<br>2. The AH endpoint specifies the measurement types it supports and currently allows in a capability report message.<br>3. The QoS manager updates the QoS capabilities for the endpoint in the central database. |

| Mandatory measurements | | |
|---|---|---|
| CPU load: 5-min min/avg/max of<br>○ User %<br>○ System %<br>○ Wait IO time %<br>○ Total % | Memory usage<br>○ Free memory % | AH network interface load: 5-min min/avg/max of<br>○ Egress load %<br>○ Ingress load %<br>○ Egress packet drop + error %<br>○ Ingress packet drop + error % |

| **Optional measurements** | | |
|---|---|---|
| CPU load: 5-min min/avg/max of<br>○ Run queue length<br>○ Number of interrupts | Memory usage<br>○ Swap usage % | AH network interface load: 5-min min/avg/max of<br>○ TCP retransmission % |

Measurement types and subtypes are uniquely identified by hierarchical, OID-like unique identifiers. They are labeled as dotted decimals. The OID list can be flexibly extended in the future thanks to the hierarchy. The OID definitions and their explanations will be collected in a single document or Wiki page publicly accessible to the Arrowhead users.

Format: x.y.z (where x is the resource group, y is the subgroup identifier, and z is the measured object identifier)

| **Group name (x)** | **OID prefix** |
|---|---|
| CPU | 1.y |
| Memory | 2.y |
| Network interface | 3.y |

| **CPU sub-group (y)** | **Type (M/O)** | **OID** |
|---|---|---|
| User space load % | mandatory | 1.1.z |
| System space load % | mandatory | 1.2.z |
| Wait IO time % | mandatory | 1.3.z |
| Total CPU load % | mandatory | 1.4.z |
| Run process queue length | optional | 1.5.z |
| Number of interrupts per second | optional | 1.6.z |

| **Memory sub-group (y)** | **Type (M/O)** | **OID** |
|---|---|---|
| Free % | mandatory | 2.1.z |
| Swap % | optional | 2.2.z |

| Network sub-group (y) | Type (M/O) | OID |
|---|---|---|
| Egress load % | mandatory | 3.1.z |
| Ingress load % | mandatory | 3.2.z |
| Egress packet drop + error % | mandatory | 3.3.z |
| Ingress packet drop + error % | mandatory | 3.4.z |
| TCP retransmission % | optional | 3.5.z |

| Statistics object (z) | OID |
|---|---|
| Minimum | x.y.1 |
| Mean | x.y.2 |
| Maximum | x.y.3 |
| Current | x.y.4 |

**Control protocol specification**

- Protocol name: **AH-EMCP** (Arrowhead Endpoint Monitoring Control Protocol)
- Transport Protocol: TCP
- Message format: JSON

| AH-EMCP messages | |
|---|---|
| Capability QUERY | Request endpoint monitoring capabilities |
| Capability RESPONSE | Reporting endpoint monitoring capabilities |
| Measurement SETUP | Preparing a measurement: providing the requested measurement and related parameters |
| Measurement SETUP ACK | Acknowledgement of measurement preparation |
| Measurement START | Start measurement |
| Measurement GET results | Request results |
| Measurement POST results | Submit results |
| Measurement FINISH | Close measurement |

## Message schemes

### Capability QUERY

```
[
  {
  "header" : {
    "ProtocolVersion" : 1, // interger
    "MsgType" : "CapQuery", // string
    "MsgSeq" : 1, // integer
    "MgrIP" : "1.2.3.4", // IPv4 or v6 address as string
    "MgrPort" : 9000  // Collector TCP port
    }
  }
]
```

### Capability RESPONSE

```
[
  {
  "header" : {
    "ProtocolVersion" : 1,  // interger
    "MsgType" : "CapResponse", // string
    "MsgSeq" : 1,  // integer
    "EndpointIP" : "5.6.7.8",   // IPv4 or v6 address as string
    }
  "body" : {
    "MeasTypesSupported" : [OID_1, OID_2, OID_3, OID_4, OID_5, OID_6, OID_7,
OID_8],  // list of object IDs as strings, e.g., "1.2.3" or "1.2."
    "MeasTypesActivated" : [OID_2, OID_3, OID_5, OID_6, OID_8]  // list of object
IDs as strings, e.g., "2.1.4" or "1.2."
    }
  }
]
```

### Measurement SETUP

```
[
  {
  "header" : {
    "ProtocolVersion" : 1,  // interger
    "MsgType" : "MeasSetup", // string
    "MsgSeq" : 1,  // integer
    "MgrIP" : "5.6.7.8",    // IP address as string
    "MgrPort" : 9000  // Collector TCP port
    }
  "body" : {
    "MeasTypesRequested" : [OID_1, OID_2, OID_3, …],  // list of object IDs as
strings, e.g., "1.2.3" or "1.2."
    "MeasParameters" : [OID_1 : [ ], OID_2 : [ ], OID_3 : [ ], …]  // list of
object IDs as strings, e.g., "2.1.3" : [1, 10, …]"
    }
  }
]
```

## Measurement SETUP ACK

```
[
  {
  "header" : {
    "ProtocolVersion" : 1,  // interger
    "MsgType" : "MeasSetupAck", // string
    "MsgSeq" : 1,  // integer
    "EndpointIP" : "5.6.7.8",   // IPv4 or v6 address as string
    "ResponseCode" : 200  // integer
    }
  }
]
```

## Measurement START

```
[
  {
  "header" : {
    "ProtocolVersion" : 1,  // interger
    "MsgType" : "MeasStart", // string
    "MsgSeq" : 1,  // integer
    }
  }
]
```

## Measurement GET Results

```
[
  {
  "header" : {
    "ProtocolVersion" : 1,  // interger
    "MsgType" : "MeasGetResults", // string
    "MsgSeq" : 1,  // integer
    "MgrIP" : "1.2.3.4", // IPv4 or v6 address as string
    "MgrPort" : 9000  // Collector TCP port
    }
  }
]
```

## Measurement POST Results

```
[
  {
  "header" : {
    "ProtocolVersion" : 1,  // interger
    "MsgType" : "MeasPostResults", // string
    "MsgSeq" : 1,  // integer
    "EndpointIP" : "5.6.7.8",   // IPv4 or v6 address as string
    }
  "body" : {
    "MeasResults" : [OID_x.y_1 : [1,2,3,4], OID_x.y_2 : [1,2,3,4], OID_x.y_3 :
[1,2,3,4],  , …]  // list of object IDs and meas values as strings, e.g., "2.1 :
[1, 2, 3, 4]"
    }
  }
]
```

## Measurement FINISH

```
[
  {
  "header" : {
    "ProtocolVersion" : 1,  // interger
    "MsgType" : "MeasFinish", // string
    "MsgSeq" : 1,  // integer
    "EndpointIP" : "5.6.7.8",   // IPv4 or v6 address as string
    }
  }
]
```

# 5. Integration results in use-cases

## 1) SAE (WP7, UC1.7) – Interoperable intelligent management of production lines: Towards Model-based Enterprise

In the recent period, AITIA has investigated the AI-related translation possibilities to standardize the SAE's custom EC data set based on the IPC 2581 standard's requirement.
The scope of the use case is to automatically map between a custom XML format to a standardized, IPC 2581-related format.

To perform the mapping procedure, AITIA proposed a two-step method. The first step is to create a mapping definition table using the RAG (Retrieval-Augmented Generation) method.

The second step is to apply the mapping definition and standard-related knowledge for the automatic mapping. Figure 5.1. represents the mapping definition creation procedure, and Figure 5.2. presents the translation components.

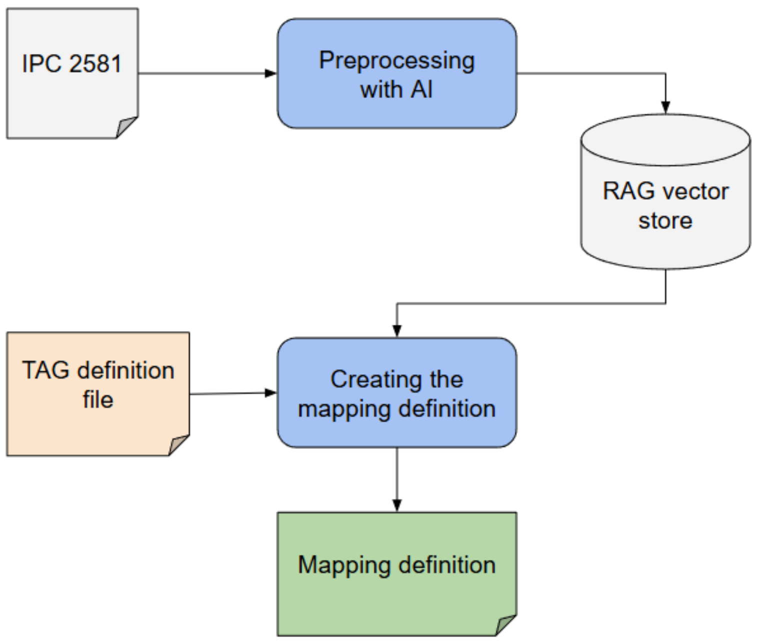

Figure 5.1 – AI-based model to create the mapping definition file

The TAG definition file contains the original, custom XML TAG names with a short description. Table 5.1. presents an example content.

| Name of the EC TAG | Description |
|---|---|
| EPPL Family | Electronic Component's Family (Resistors, Capacitors, Diodes,...) |
| EPPL Group | Type of material of the component |
| Component Number | ID used by the manufacturer |

Table 5.1 – Example content from the TAG definition file

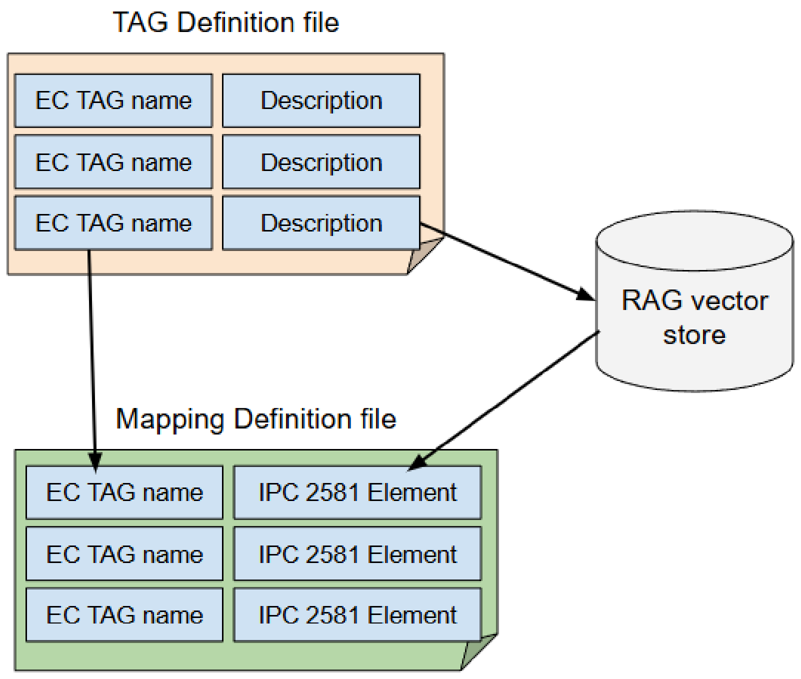

Figure 5.2 – Creating the mapping definitions

To create the mapping definition table, AITIA has applied a predefined prompt, and iteratively retrieved possible IPC 2581 element names from the RAG vector store. To find the relevant elements and subcomponents, AITIA applied the TAG definition's description in each iterative step (see Figure 5.3.).

| No. | EC custom TAG | IPC 2581 element (Option-1) | IPC 2581 element (Option-2) |
|---|---|---|---|
| 1 | EPPL_Family | Component/refDes | Component/packageRef |
| 2 | EPPL_Group | BomItem/matDes | Component/matDes |
| 3 | Quality_type | Compliance | Technology |
| 4 | Description | BomItem/refDes | Ecad |
| 5 | SPN | AvlMpn/name | Component/part |

Table 5.2 – Example content from the mapping definition file

Based on the mapping definition table, the standardization of the custom XML format could be performed. The translation could be solved with predefined functions, or an LLM (Large Language Model) could be also applied to generate XML content with the standardized element names.

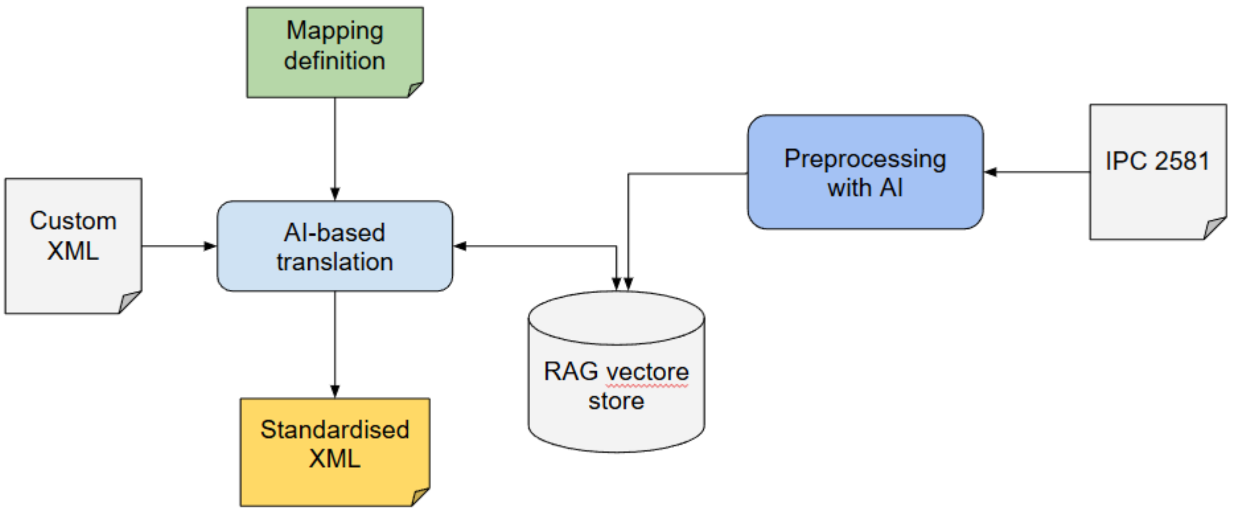

Figure 5.3 – Mapping between custom and standardized models

To apply the previous steps through the Arrowhead framework, AITIA defined a concept regarding the Translation services. To integrate the translator which solves the SAE's use case, it will be further developed as a Data Translation Provider. The Arrowhead translation concept is presented by Figure 5.4.

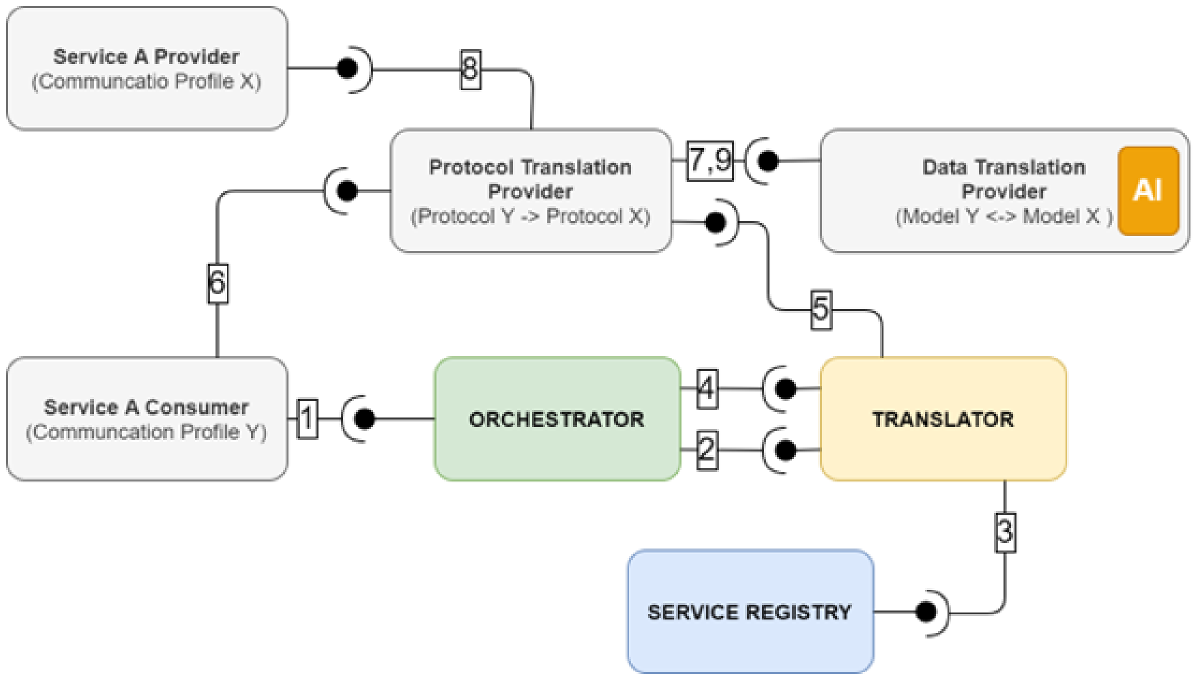
Service A Provider (Communcatio Profile X)
8
Protocol Translation Provider (Protocol Y -> Protocol X)
7,9
Data Translation Provider (Model Y <-> Model X )
AI
6
5
Service A Consumer (Communcation Profile Y)
1
ORCHESTRATOR
4
2
TRANSLATOR
3
SERVICE REGISTRY

Figure 5.4 – Extending the Arrowhead framework with Translation services

## 2) AVL (WP6, UC1.6) – Automotive Battery Innovation fPVN

### 2.1 Battery development - Test lab

The scope of this use case is to collect and label incoming sensor measurement data, and automatically fill out a custom format-based table based on a predefined mapping definition file. The mapping definition file contains the name of the output key, and a short text description about the content. Based on the description, the content from the central data lake should be retrieved and formatted regarding the requirements.

Figure 5.5 presents the proposed model from AVL.

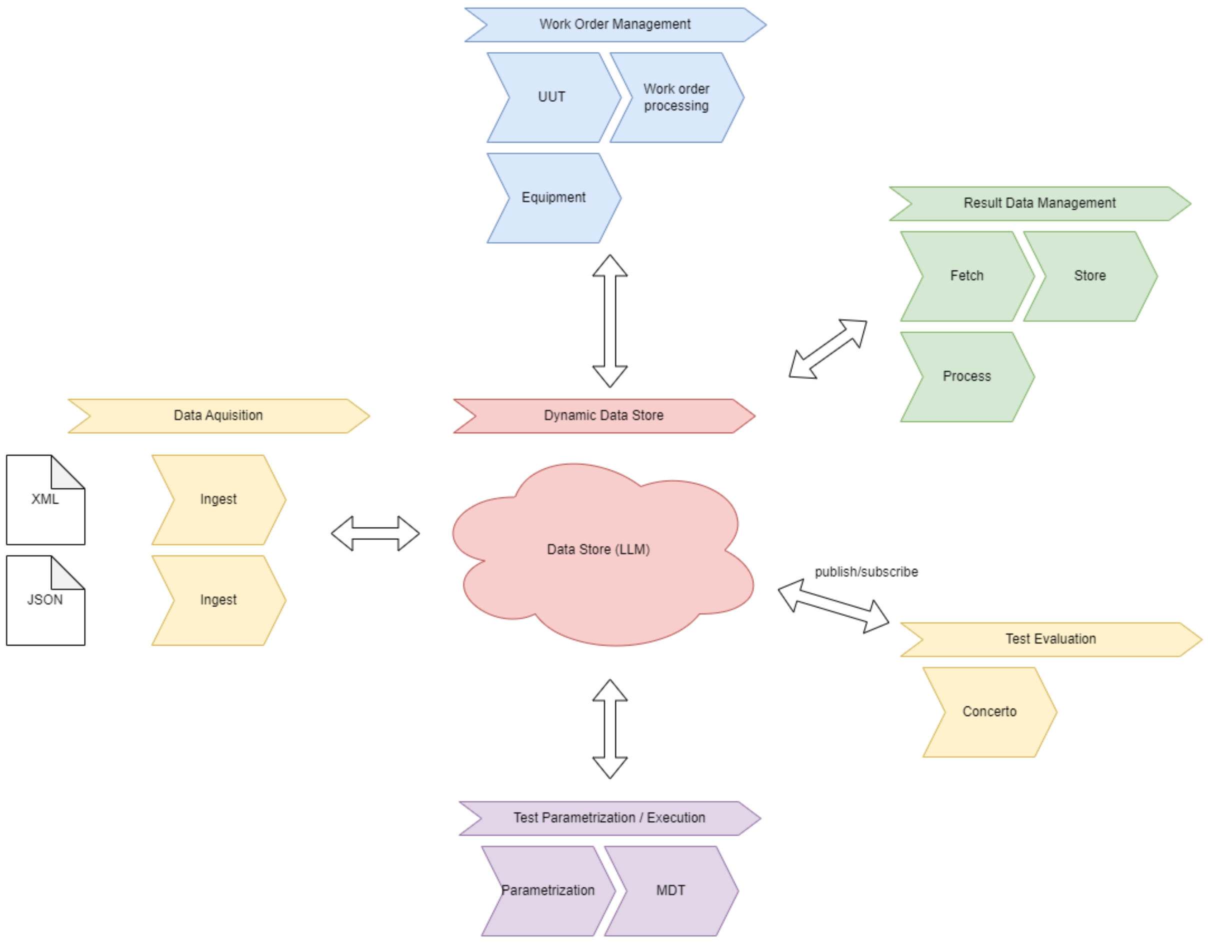

Figure 5.5 – Measurement data processing pipeline

Based on the aims of the Arrowhead framework's microservice architecture, the subsystems (presented by Figure 5.5) could be separated into a Provider – Consumer model. Each subsystem (e.g., Result Data Management) could apply a specific provider, which retrieves the relevant information from the Data Store, and generates the required content (see Figure 5.6.).

This provider could be also an AI-based model, or a specific translator as well.

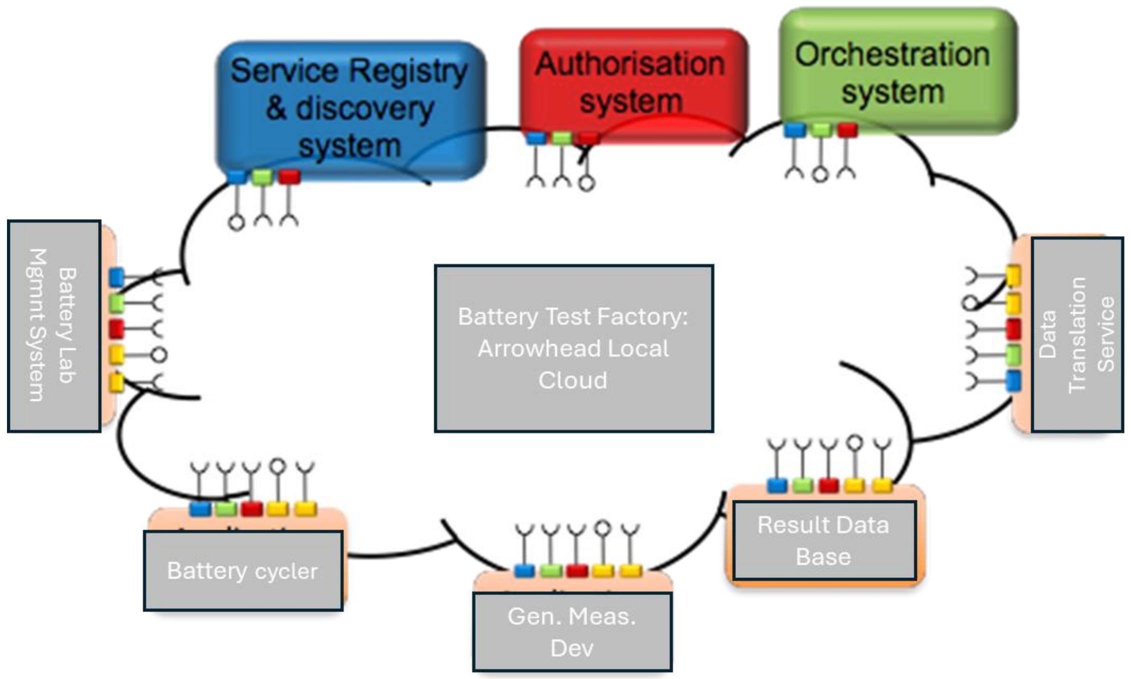

Figure 5.6 – Arrowhead microservice concept regarding UC 1.6-Testlab

## 2.2 Battery end-of-life treatment

The objective of this use case is to efficiently re-purpose/recycle the batteries by having an accurate and reliable determination process of their SoX (State of Health, … Sate of X) and to improve the existing SoX by combining inspection technologies with estimation methods. Applying micro service paradigm is a practical solution to achieve these objectives, therefore Arrowhead FW integration is planned in order to manage, orchestrate and coordinate each device within a local cloud. Also, existing databases and eventual data sharing via battery pass database are planned to be integrated via adapters. (See Figure 5.7.)

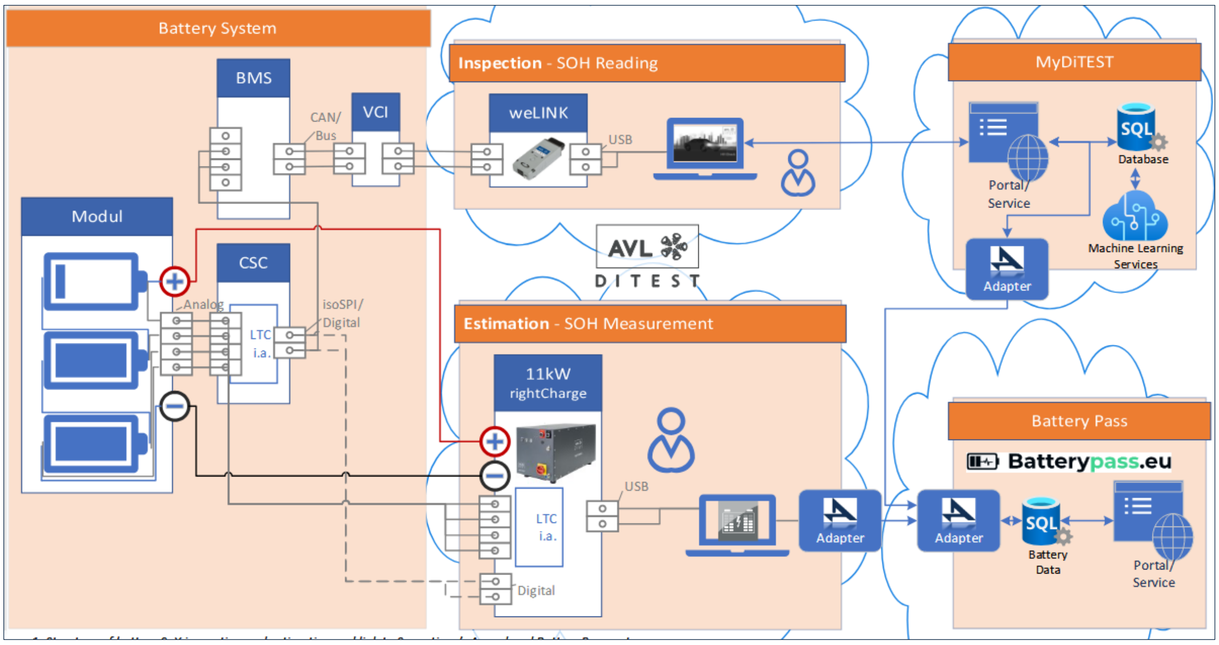

Figure 5.7 – Arrowhead microservice concept regarding UC 1.6-EoL Treatment

# 6. The Arrowhead Copilots

The target of the Arrowhead copilot is to enable the design and management of Arrowhead-supported Cyber-Physical System of Systems through natural language interfaces while analyzing its status and monitoring data using the potentially multimodal output of Copilots. The concept is briefly summarized in this chapter – and it is further elaborated in [5].

This target has been achieved by conceptualizing three stages of Copilots within the Arrowhead Engineering Process, each with increasing complexity and added value.
These phases are shown by Figure 6.1.

1. **Arrowhead Expert Copilot**
   A chat-based entry point into the design and engineering of Arrowhead-based CPSoS architectures that leverages existing documentation and publications within the ecosystem. This Copilot can address design and integration-related queries and could be embedded in the Arrowhead Framework Wiki as an inline chatbot. Its intended audience includes all Wiki visitors seeking guidance or information about the Arrowhead ecosystem.

2. **Arrowhead Management Copilot**
   This Copilot interacts with various Arrowhead Core Systems within a Local Cloud deployment. It is designed to analyze, understand, and potentially manage CPSoS operations through the Arrowhead governing middleware. This tool can be integrated as a widget within the Arrowhead Management Tool GUI, targeting authenticated Local Cloud (SoS) operators as its primary users.

3. **Arrowhead Design Copilot**
   Integrated with the engineering toolchain, this Copilot supports the design of SoS deployments, industrial automation processes, and infrastructure. For example, it can facilitate SysML modeling with tools like Eclipse Papyrus. It is envisioned as part of the Arrowhead Engineering Toolchain, working alongside design tools for CPSoS. The intended users are SoS engineers involved in system design and development.

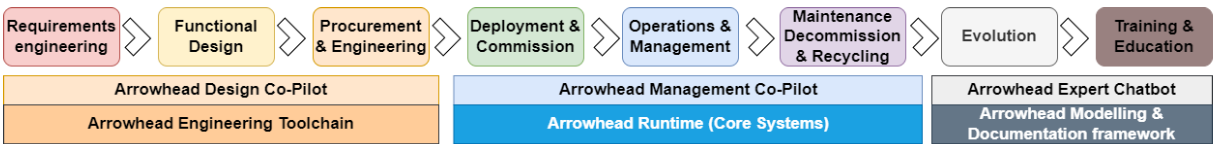
Requirements engineering
Functional Design
Procurement & Engineering
Deployment & Commission
Operations & Management
Maintenance Decommission & Recycling
Evolution
Training & Education
Arrowhead Design Co-Pilot
Arrowhead Engineering Toolchain
Arrowhead Management Co-Pilot
Arrowhead Runtime (Core Systems)
Arrowhead Expert Chatbot
Arrowhead Modelling & Documentation framework

Figure 6.1 - Overview of the Arrowhead Engineering Process (AEP) supported by Co-pilots

The Arrowhead Expert Copilot is specifically useful for training, education, and extracting and explaining expert knowledge from Arrowhead-related documentation. Its structure is shown by Figure 6.2.

This Expert Copilot is able to answer reasonable questions like *"What are the restrictions on system naming in Arrowhead?"*, or *"How can manually generate certificates for a new application system, if I do not have Onboarding Controller?"*

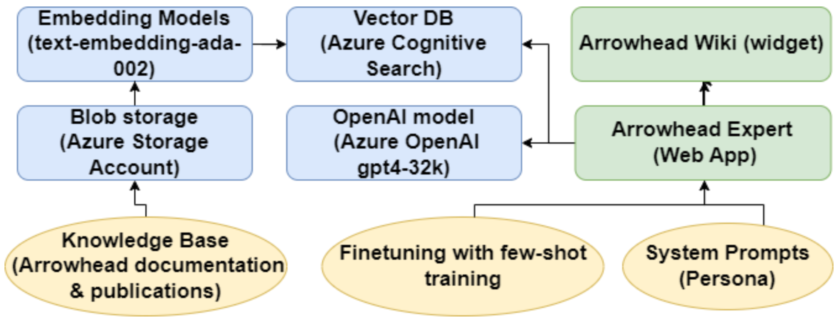
Embedding Models (text-embedding-ada-002)
Vector DB (Azure Cognitive Search)
Arrowhead Wiki (widget)
Blob storage (Azure Storage Account)
OpenAI model (Azure OpenAI gpt4-32k)
Arrowhead Expert (Web App)
Knowledge Base (Arrowhead documentation & publications)
Finetuning with few-shot training
System Prompts (Persona)

Figure 6.2 – The main elements of the Arrowhead Expert Copilot [5]

The Arrowhead Management Copilot is able to plan and execute management and operation tasks real-time, in the working Arrowhead Local Couds (for which it has access to). Its structure is shown by Figure 6.3.

This Copilot is able to answer simple questions like *"What are the currently registered services in the Arrowhead Service Registry?"*, but it can also chain API calls together in sequential plans, and execute them. An example of such complicated tasks, could be: *"Please add an authorization rule between systems P and C, for service S in the Arrowhead intra-cloud Authorization System, but please register system X as the provider of service S in the Arrowhead Service Registry first"*.

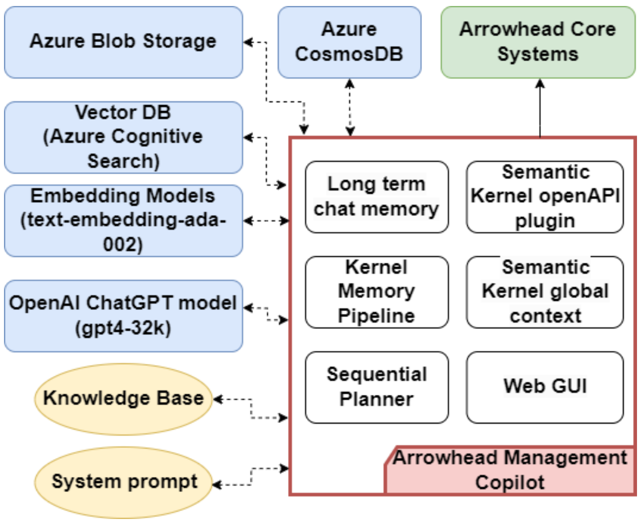
Azure Blob Storage
Azure CosmosDB
Arrowhead Core Systems
Vector DB (Azure Cognitive Search)
Embedding Models (text-embedding-ada-002)
OpenAI ChatGPT model (gpt4-32k)
Knowledge Base
System prompt
Long term chat memory
Semantic Kernel openAPI plugin
Kernel Memory Pipeline
Semantic Kernel global context
Sequential Planner
Web GUI
Arrowhead Management Copilot

Figure 6.3 – The main elements of the Arrowhead Management Copilot [5]

# 7. Collaboration with other technical WPs

WP2 continued its regular collaboration meetings with WP3 and WP4 to complete the requirements about the integration of the T3.1 – T3.3, and of the T4.1 – T4.3. This work results in the microservice-based AH Eclipse architecture. The requirements cover the following topics:

- Ensure Syntactic, Semantic and Pragmatic Interoperability between systems.
- Ensure interoperability between major standards.
- Ensure the integration possibilities of AI-based Translator modules.
- Ensure the integration possibilities of Model-based Translator modules.

Based on the T3.1 survey results, the participants reported more than 27 major industrial data models, and more than 10 data representation formats. The various WPs of the Arrowhead fPVN project is now working on translators for these data representations.

The reported formats could be separated into three groups:

1. The major of the representation formats are XML and JSON, including predefined or standardized schemas.
2. EXPRESS language-based models.
3. Model-based representation (e.g., UML).

The current result of the collaboration between Work Packages, is that the Arrowhead Eclipse core systems are started to get designed to provide the following features:

- Translation subsystems based on predefined schemas. The schemas should be uploaded by a provider or consumer, and should be handled as metainformation by the AH system components.
- Schema-related orchestration.
- Automatic recognition of the interoperability problems, and possible solutions with translator instantiation.
- Extending the orchestration parameters with context-specific information to support the Pragmatic Interoperability aspect.

# 8. Conclusion

This is the second deliverable – D2.2 – of the WP2 “Microservices” work package of the Arrowhead fPVN project.

One of the major results of the last year of WP2 is that the partners defined and created the v5.0 of the Authorisation core system, the QoS monitoring concept, and many essential core services, as described in this document. This work is the result of the roadmap discussion elaboration and definition for Eclipse Arrowhead v5.0, including enhancing the core systems, defining microservice development, and CPSoS creation. The Arrowhead copilots (design, management and expert phases) has also been conceptualized and created as a proof of concept in this period. Additionally, WP2 collaborated with WP3 and WP4 to establish a robust interoperability platform, enabling translations and adaptations for various digital data models and languages.

As part of maturing the Arrowhead framework, the stakeholders addressed bug fixes, vulnerability fixes, further detailed the Eclipse Arrowhead v5.0 roadmap tasks, created reference definitions for v5.0 of recommended core systems, and services – taking into account microservice-related requirements for industry5.0 needs.

As an originally unplanned result, WP2 conceptualized the Arrowhead Copilot, provided expert knowledge to its RAG knowledge base, and created the initial, working proof of concept for the Arrowhead Expert Copilot and the Arrowhead Management Copilot.

By working together with WP3 and WP4, the WP2 partners are implementing the basic concepts for the identification of non-interoperable properties, as well as the QoS monitoring concept elements.

# 9. Appendixes

Besides this “Deliverable D2.1 Main document (Arrowhead_fPVN_D2.1_v0.5)” the other document of this deliverable are the following:

1) Authentication Core System – SysD v5.0.0 (ah_authentication_sysd.pdf)
2) system-discovery – SD (system-discovery_sd.pdf)
3) service-registry-management – SD (service-registry-management_sd.pdf)
4) service-discovery – SD  (service-discovery_sd.pdf)
5) device-discovery – SD (device-discovery_sd.pdf)

Furthermore, the Roadmap discussions and results can be found on github:

https://github.com/eclipse-arrowhead/roadmap/tree/main/5.0%20Draft

## 10. References

[1] https://github.com/Aitia-IIOT/ah5-common-java-spring/tree/development
[2] https://github.com/Aitia-IIOT/ah5-core-java-spring/tree/development
[3] https://aitia-iiot.github.io/ah5-docs-java-spring
[4] https://github.com/eclipse-arrowhead/roadmap/issues/63
[5] Cs. Hegedűs and P. Varga, "Co-pilots for Arrowhead-based Cyber-Physical System of Systems Engineering," *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, Seoul, Korea, Republic of, 2024, pp. 1-6,
doi: 10.1109/NOMS59830.2024.10575845.

# 11. Revision history

## 11.1 Contributing and reviewing partners

| Contributions | Reviews | Participants | Representing partner |
|---|---|---|---|
| Architecture Definition | | | AITIA |
| Architecture Definition | | | BME |
| Architecture Definition | | | SINETIQ |
| Architecture Definition | | | LTU |
| Architecture discussion and review | | | EUROTECH |
| Architecture discussion and review | | | BEIA |
| Architecture discussion and review | | | LEONARDO |

## 11.2 Amendments

| No. | Date | Version | Subject of Amendments | Author |
|---|---|---|---|---|
| 1 | 18/11/2024 | 0.1 | Initial version | Pal Varga |
| 2 | 28/11/2024 | 0.2 | Core 5.0 components added | Tamas Bordi, Rajmund Bocsi |
| 3 | 29/11/2024 | 0.3 | Maturing the framework | Pal Varga |
| 4 | 27/12/2024 | 0.4 | Task contributions, WP3-4 collaborations | Pal Varga, Tamas Tothfalusi |
| 5 | 07/01/2025 | 0.5 | Use cases | Pal Varga |
| 6 | 16/01/2025 | 0.6 | Finalization for review | David Rutqvist, Pal Varga |
| 7 | 19/01/2025 | 1.0 | Review suggestions incorporated | Pal Varga |

## 11.3 Quality assurance

| No | Date | Version | Approved by |
|---|---|---|---|
| 1 | 17/01/2025 | 0.6 | Jerker Delsing |
| 2 | 19/01/2025 | 1.0 | Jerker Delsing |