



Document title: Arrowhead fPVN Deliverable D2.3

Version
1.0

Author
Pal Varga

Status
final

Contact
pvarga@tmit.bme.hu

Date
2026-02-26

Deliverable D2.3

Third 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 D2.3 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.

Table of contents

1. Introduction.....	4
1.1 The elements of Arrowhead fPVN deliverable D2.3	4
1.2 Overview of WP2 aims and tasks	4
1.3 The recent proceedings of Task2.1.....	4
1.4 The recent proceedings of Task2.2.....	5
1.5 The recent proceedings of Task2.3.....	6
1.6 The recent proceedings of Task2.4.....	7
1.6.1 General.....	7
1.6.2 The Eclipse Arrowhead Working Group Charter.....	8
2. WP2 Objectives.....	10
3. Recent results on maturing the Eclipse Arrowhead framework.....	11
3.1 Specifications	11
3.1.1 DynamicServiceOrchestration Core System.....	11
3.1.2 ConsumerAuthorization Core System.....	11
3.1.1 Authentication System	11
3.1.2 Blacklist Support System	11
3.1.3 TranslationManager Support System	11
3.1.4 DeviceQoSEvaluator Support System	11
3.2 Reference Implementation	11
3.2.1 General.....	11
3.2.1 Consumer Athorization Core System.....	12
3.2.2 Dynamic Service Orchestration Core System.....	13
3.2.3 Simple Store Service Orchestration Core System	15
3.2.4 Authentication Core System.....	15
3.2.5 Blacklist Support System	17
3.2.6 Translation Manager Support system.....	18
3.2.7 Device QoS Evaluator Support System	20
4. Integration results in use-cases	22
4.1 SAE (WP7, UC1.7) – Interoperable intelligent management of production lines: Towards Model-based Enterprise	22
4.2 AVL (WP6, UC1.6) – Automotive Battery Innovation fPVN	25
4.2.1 Battery development - Test lab	25
4.2.2 Battery end-of-life treatment.....	28
4.3 UC 2.6 – Architecture Concept: Command Robustness Screening API	29
4.3.1 Architectural Overview	29

4.3.2 Application Systems and Responsibilities.....	30
4.3.3 Service-Oriented Interaction Flow.....	31
4.3.4 Orchestration and QoS-Aware Provider Selection.....	32
4.3.5 Security and Authorization	32
4.3.6 Mapping from User Workflow to Services.....	32
4.3.7 Scalability and Evolution	32
4.3.8 Relation to Arrowhead Translation Capabilities	33
4.3.9 Summary	33
4.4 Process industry (WP9, UC1.9, 2.9 and 3.9) - Integrating information systems in process industry fPVNs.....	33
5. The Arrowhead Copilots	35
6. Collaboration with other technical WPs	38
7. Conclusion	39
8. Appendixes	40
9. References	41
10. Revision history.....	42
10.1 Contributing and reviewing partners	42
10.2 Amendments	42
10.3 Quality assurance.....	42

1. Introduction

1.1 The elements of Arrowhead fPVN deliverable D2.3

The deliverable D2.3 of Arrowhead fPVN consists of the following documents.

- 1) Deliverable D2.3 Main document (Arrowhead_fPVN_D2.3_v1.0)
- 2) SYSD documents
- 3) SD documents
- 4) Arrowhead Translation presentation
- 5) Arrowhead v5.2 features highlight presentation
- 6) Arrowhead v5.1 reference implementation and documentation

<https://projects.eclipse.org/projects/iot.arrowhead/releases/5.1.0>

<https://github.com/eclipse-arrowhead/ah5-translation-manager-java-spring>

- 7) Arrowhead v5.2 reference implementation and documentation

<https://projects.eclipse.org/projects/iot.arrowhead/releases/5.2.0>

<https://github.com/eclipse-arrowhead/ah5-core-java-spring>

<https://github.com/eclipse-arrowhead/ah5-device-qos-evaluator-java-spring>

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 microservices/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:** Continued creating necessary *common* 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.x.

- **Core System Upgrades:** Successfully created Core system v5.2 for: *authentication; consumer-authorization; serviceregistry; serviceorchestration-dynamic; serviceorchestration-simple*, and Support systems v5.2 for *translation manager; blacklist* [2].
- **Stakeholder Engagement and Commercial Strategy:** Conducted significant meetings in Budapest and Lisbon on continuing efforts for commercializing Arrowhead Framework components and strategies.

There were no significant deviations reported from the original plan for this period. 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 a huge progress in advancing the Eclipse Arrowhead framework. There were plans on the roadmap but T2.1 exceeded many of them, especially with the core and support system reference implementations (described later in this deliverable).

WP2 continues on strong roadmap creation and execution in relation on concepts as well as on implementation.

1.4 The recent proceedings of Task2.2

Task T2.2 develops concepts and concrete mechanisms for data management in non-interoperable SOA environments. Besides, it is also addressing security and privacy in microservice-based architectures. The work is centered on how Eclipse Arrowhead v5.1 and v5.2 and its Core/Support Systems can be extended to support dynamic instantiation and runtime selection of interoperability components, such as translation services and adaptors. A key goal is to enable Arrowhead-based CPSoS to operate across heterogeneous interfaces and data models without forcing prior standardization. The task therefore combines architectural assessment, enabling runtime orchestration logic, and security-by-design considerations.

A major technical requirement is provider selection under operational constraints. The orchestration must select the “best” provider system, not just any compatible one. This implies QoS-aware orchestration, where QoS attributes between consumer and provider are continuously observed and used in selection decisions. During this period, the QoS monitoring specification was completed, and a first proof-of-concept implementation was delivered. This provides a foundation for closed-loop decisions in Arrowhead v5 deployments, where service choices can be adapted based on observed latency, availability, or other QoS metrics.

In parallel, T2.2 carried out a focused evaluation of the Arrowhead v5.1 and v5.2 architecture and its reference implementation (provided by task 2.1 and 2.3) to identify gaps and required updates for (i) dynamic services and (ii) operation in non-interoperable environments. These findings were discussed in roadmap meetings and between sessions, and they directly motivate the creation of translation services and adaptors as first-class Arrowhead building blocks, in close collaboration with WP3 and WP4.

As a result, the Eclipse Arrowhead v5 translation concept is realized through the TranslationManager Support System, which enables interoperability when a consumer and

provider do not share a common communication interface and/or data model. At runtime, TranslationManager can assess feasibility and then compose a translation bridge by chaining available translator systems as modular building blocks, exposed via services for bridge creation, reporting, and management, and accessible over both HTTP and MQTT. In our reference implementation, an interface translator provides protocol-level bridging between Arrowhead's built-in generic_mqtt(s) and generic_http(s) profiles (and supports HTTP-to-HTTP paths when only the data model differs), while a Python-based data-model translator adapter allows plugging in arbitrary translation algorithms, including AI/ML semantic translators from WP4. A demonstration scenario - described in section 3 - validates the approach end-to-end by translating a producer's custom XML-like semantics into the IPC-2581 industrial standard while simultaneously bridging MQTT-to-HTTP. This shows how Arrowhead v5 can deliver scalable, composable, and extensible interoperability in heterogeneous CPSoS environments.

To support autonomous CPSoS operation, T2.2 also progressed the foundations of security and privacy safeguards for interoperability-enabled flows. The work targets robust access control, secure service invocation, and integrity protection across both local and global fPVN clouds. Concept elaboration started in this period, with emphasis on how security controls should remain enforceable even when translation bridges are composed dynamically at runtime.

1.5 The recent proceedings of Task2.3

Task T2.3 investigates various microservice-based methodologies for Industry5.0. The work targets concrete integration strategies for the project use cases. It also examines human-in-the-loop aspects in microservices/SOA-based SoS deployments. A core premise is that microservices are a primary enabler of structured data exchange and governed interoperability in Industry 5.0 CPSoS. The task delivers (i) reusable microservices for CPSoS, (ii) a practical methodology guide for industrial adoption, and (iii) hands-on technical support for the pilots.

During the previous period, T2.3 and T2.1 built on each others' previous results, and joint work with the WP3/WP4, as well as with the use cases progressed smoothly.

One of the main objectives of T2.3 is to define repeatable design patterns for CPSoS architectures in Industry 5.0 using the Eclipse Arrowhead v5.0 approach. The efforts in this period focused on three technical streams:

(i) Arrowhead v5.0 microservices for CPSoS building blocks.

Microservices supporting CPSoS composition are being developed along the Eclipse Arrowhead v5.0 roadmap. The focus is on reusable, interoperable components that fit the Arrowhead model of Local Clouds, Core Systems, and Support Systems. The goal is to enable consistent service registration/discovery, policy-based access control, and cross-system data exchange between heterogeneous industrial subsystems. This work was joint with T2.1, as the maturation of Eclipse Arrowhead and the introduction of new systems and services for Industry5.0 naturally comes together at this stage of the project. Partners are designing microservices that can be combined into higher-level CPSoS workflows, while keeping clear service contracts and separation of concerns. Results are shown in section 3 of this document.

(ii) Arrowhead Domain Specific Language (DSL) for engineering.

The subtask to develop the Arrowhead DSL as a SysML profile is driven by engineering and documentation requirements. The second integration round into Eclipse Papyrus has been completed successfully. The DSL was piloted in the LTU D7065E course with Master students and professionals. Participants used the DSL for black-box and partial white-box models of real use cases, including Local Cloud and system-of-systems views. Feedback was strongly positive, supporting continued maturation of the DSL as a professional engineering tool. Next steps include extending the Papyrus integration to better support Arrowhead v5.1 and v5.2 engineering phases and reducing manual steps when modeling Local Clouds and systems of Local Clouds.

(iii) fPVN cookbook and CPSoS engineering guidance.

Roadmap discussions continued on the Arrowhead fPVN cookbook, intended as a practical “how-to” guide for architects and developers. The cookbook targets microservice-based CPSoS design, including guidance on service decomposition, interface design, cross-cloud interoperability, and governance patterns. The ongoing definition of the GSoSD already provides a foundation for structuring these guidelines and aligning them with Arrowhead v5.0 concepts.

Finally, T2.3 continued to provide technical support for the industrial pilots. Support includes architecture and integration targeted training, and iterative troubleshooting. Dedicated communication channels are in place to ensure fast feedback loops and consistent alignment between pilot needs and the Arrowhead v5.0 roadmap. Results are presented in section 4 of this document.

1.6 The recent proceedings of Task2.4

1.6.1 General

During this reporting period, Task 2.4 has continued its focus on governance methods for both design-time and run-time stages. Building on the work from the previous period, further efforts have been directed towards governance aspects relevant for service-based and microservice-oriented architectures.

The work on machine-readable specifications has progressed with additional investigation of how service interface definitions and deployed services can be consistently identified and managed. This includes continued exploration of registries for service definitions and interface/API specifications, supporting their use across different solutions and tools such as service discovery and authentication/authorization. The relationship between interface specifications and service instances identities has been further examined as a basis for governance at run time.

The aim has been to reduce the need for manual configuration while maintaining controlled access to services. The current state of the work allows for demonstrations, but further refinement and validation are required before the results can be made available for wider use by the project partners.

The activities in Task 2.4 have also been informed by related work in WP2 on testing and validation of system-of-systems compositions in microservice environments. Experiences from

this work have provided input to architectural discussions, in particular concerning core microservice functionality and security aspects, and have influenced the governance approaches studied within this task.

Input from the ongoing work in use cases 1.9, 2.9, and 3.9 has provided valuable feedback for this task. Contributions to architectural discussions, interface definition, and implementation-related activities within these use cases have helped to assess the applicability of the proposed governance methods in concrete settings. While this interaction has supported the validation of the concepts, the extent of verification is still limited by the current level and scope of Arrowhead framework adoption within the use cases.

No major deviations from the planned work have been identified during this period.

1.6.2 The Eclipse Arrowhead Working Group Charter

The draft version of the *Eclipse Arrowhead WG Charter* have been created and circulation is started among already interested parties and possible members.

Two starting points of this art the mission and vision statements:

The mission of Eclipse Arrowhead is to provide a forum for individuals and organizations regarding development, maintenance, promotion and marketing of the Eclipse Arrowhead System of Systems architecture and implementation platform.

The main objective of the Eclipse Arrowhead Working group is to encourage, develop, and promote the open source Eclipse Arrowhead architecture and implementation platform targeting System of Systems application solution applicable to a wide range of application domains of commercial interest.

The discussion, update and finalization plan for the Eclipse Arrowhead WG Charter is the following.

The process is already ongoing with a structured review (of draft version 0.3) inside the Steering Committee, focusing on strategic alignment. Leader Members drive this phase, while Innovator Members provide cross-sector balance. Then focused consultations are organized with: Active technical contributors and project leads (scope, roadmap realism), Selected Innovator and Supporter Members (entry barriers, membership value), Invited Guest Members (academia, R&D partners) where relevant. This step ensures the charter reflects not only governance theory but operational reality.

Based on consolidated feedback, a v0.4 draft of the charter is prepared. Updates focus on aligning committee roles and responsibilities more explicitly, ensuring consistency with Eclipse Foundation Bylaws, WG Process, IP, and Antitrust policies. The updated draft is shared with all WG members via the official mailing list, allowing transparent review, time-bounded written comments, clarification requests rather than renegotiation of scope.

The Steering Committee conducts a voteing in accordance with the charter and Eclipse governance rules. Where required, a super-majority is applied (e.g., for membership fee or

structural changes). Upon approval, the updated charter becomes the authoritative governance document for the next operational phase of Eclipse Arrowhead.

The updated charter will be communicated in two layers during the project closing event in Oslo (June 9-11).

First, the already operational Eclipse Arrowhead Market-Oriented Action Group (MOAG) will receive a dedicated briefing that explicitly connects the charter update to market impact: ecosystem stability, clearer participation models, branding continuity, and predictable governance. This reassures the original industry group that Arrowhead's foundations are strengthened as the scope broadens.

Second, during the wider industry session, the charter update will be presented as a future-facing enabler: a stable, vendor-neutral framework that lowers adoption barriers, supports cross-domain SoS integration, and invites new industrial players into a governed, credible open-source ecosystem. The message will focus on *what this enables next* rather than on governance mechanics.

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 got elevated to new levels, now completing v5.2. WP2 keeps working together with the technical work packages WP3 and WP4 in order to create a proper interoperability platform, especially allowing translations and adaptations for various major digital data models and languages. Maturing of released Eclipse Arrowhead core services to TRL7-8 is ongoing as the Arrowhead reference implementation gets updated in GitHub. Defining an innovative microservices based architecture, as Eclipse Arrowhead v5.2. 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. The second 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 Specifications

In this period, meetings were continued to enhance the specifications. During this work, we have further detailed the specification of the Core and Support Systems. This resulted additional documents:

3.1.1 DynamicServiceOrchestration Core System

The system descriptor and four service descriptions for services provided by this system.

3.1.2 ConsumerAuthorization Core System

The system descriptor and four service descriptions for services provided by this system.

3.1.1 Authentication System

The system descriptor and two service descriptions for services provided by this system.

3.1.2 Blacklist Support System

The system descriptor and two service descriptions for services provided by this system.

3.1.3 TranslationManager Support System

The system descriptor and three service descriptions for services provided by this system. Also, system descriptor for interface translator application systems with one service descriptor, system descriptor for data model translator application systems with one service descriptor and a tutorial document for implementing such application systems to be used by the Use Cases.

3.1.4 DeviceQoSEvaluator Support System

A specification document for the development (coding).

The documentation website was updated with the associated HTTP and MQTT Interface Design Descriptions (IDD), configuration directions and many more supporting pages:

<https://aitia-iiot.github.io/ah5-docs-java-spring/home/welcome/>

AITIA actively participated in the joint Deep Tech workshop – involving Arrowhead fPVN, AIMS5.0 and SC4EU projects –, in Lisbon, October 2025. During the event, two presentations were hold: a feature highlight about the released Eclipse Arrowhead 5.1.0 and a detailed presentation about how the translation goals are supported by the framework, especially how to integrate data model translator systems that are using various translation technologies developed in WP4.

3.2 Reference Implementation

3.2.1 General

In *development phase 3* we produced the DynamicServiceOrchestration Core System, the ConsumerAuthorization Core System, the Authentication Core System and the Blacklist

Support System. At the end of this phase these systems (with the ServiceRegistry Core System developed in phase 2) have been released as version 5.0.0. The Eclipse Foundation also reviewed the release and approved it.

<https://projects.eclipse.org/projects/iot.arrowhead/releases/5.0.0>

<https://projects.eclipse.org/projects/iot.arrowhead/reviews/eclipse-arrowhead-5.0.0-release-review>

<https://github.com/eclipse-arrowhead/ah5-core-java-spring>

<https://github.com/eclipse-arrowhead/ah5-blacklist-java-spring>

In *development phase 4* we produced the TranslationManager Support System and integrated it with the DynamicServiceOrchestration Core System. This new support system has been released as version 5.1.0:

<https://projects.eclipse.org/projects/iot.arrowhead/releases/5.1.0>

<https://github.com/eclipse-arrowhead/ah5-translation-manager-java-spring>

We also created one reference implementation for an interface translator application system (from MQTT to HTTP) and one reference implementation for data model translator application system, that is an adapter for any kind of translation algorithm written in Python.

<https://github.com/Aitia-IIOT/ah5-app-aitia-interface-translator-to-generic-http-java-spring>

<https://github.com/Aitia-IIOT/ah5-app-aitia-datamodel-translator-python-wrapper-java-spring>

In *development phase 5* we implemented the SimpleStoreServiceOrchestration Core System and the DeviceQoS Evaluator Support System. The latter was integrated also with the DynamicServiceOrchestration Core System. The phase 5 results have been released as version 5.2.0.

<https://projects.eclipse.org/projects/iot.arrowhead/releases/5.2.0>

<https://github.com/eclipse-arrowhead/ah5-core-java-spring>

<https://github.com/eclipse-arrowhead/ah5-device-qos-evaluator-java-spring>

3.2.1 Consumer Athorization Core System

arrowhead-consumer-authorization

The ConsumerAuthorization Core system enables to manage and to authorize connections between various systems using authorization rules within an Eclipse Arrowhead Local Cloud. It also provides various authorization token generation functionalities for service consumptions that add an extra layer of security.

This core system provides four services: *authorization*, *authorizationToken*, *authorizationManagement* and *authorizationTokenManagement*. Consumers can choose between HTTP and MQTT interfaces.

The 5th generation is introducing policy based authorization rules instead of only peer-to-peer rules, allows the provider systems to define their own authorization rules (with the condition that management level rules always take priority).

https://aitia-iiot.github.io/ah5-docs-java-spring/core_systems/authorization/

It also offers multiple token security options with different way of workings.

<https://aitia-iiot.github.io/ah5-docs-java-spring/help/service-security/#token-based>

3.2.2 Dynamic Service Orchestration Core System

arrowhead-serviceorchestration-dynamic system

The service orchestration core systems enable to find matching providers for the consumer's specification within an Eclipse Arrowhead Local Cloud. The 5th generation divides the different orchestration strategies to individual core systems.

DynamicServiceOrchestration Core System implements the dynamic orchestration strategy, which is using the live data of the ServiceRegistry to collect matching provider candidates on the fly without any pre-defined orchestration rule. It also able to consult with other core and support system such as

- ConsumerAuthorization Core System:

To filter out provider candidates that the consumer is not allowed to use due to missing permissions (see Figure 3.1);

- DeviceQoS evaluator Support System:

To sort or filter the provider candidates based on quality of device requirements (for example: mean RTT, current CPU load, etc.)

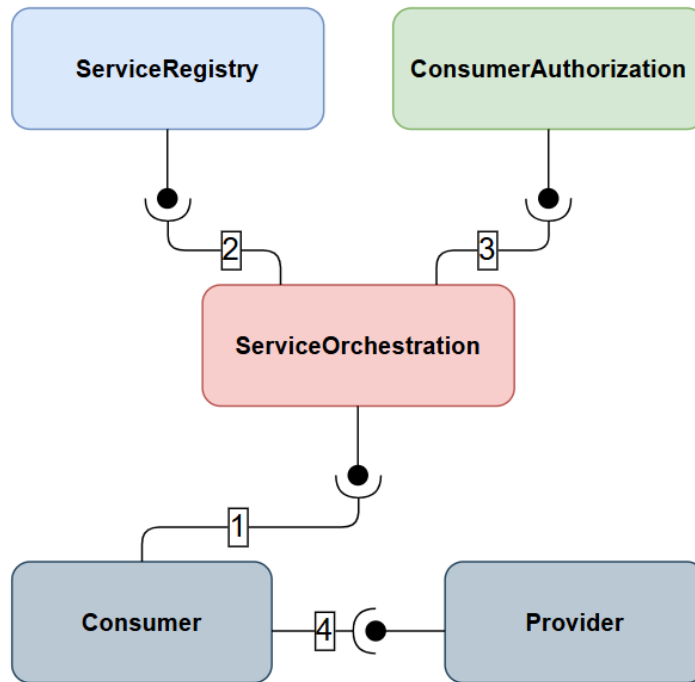
- TranslationManager Support System (see Figure 3.6):

To discover matching providers even if the native (same interface and data models) communication is not possible and to establish a translation bridge to achieve interoperability.

This core system provides four services: *serviceOrchestration*, *serviceOrchestrationPushManagement*, *serviceOrchestrationLockManagement* and *serviceOrchestrationHistoryManagement*. Consumers can choose between HTTP and MQTT interfaces.

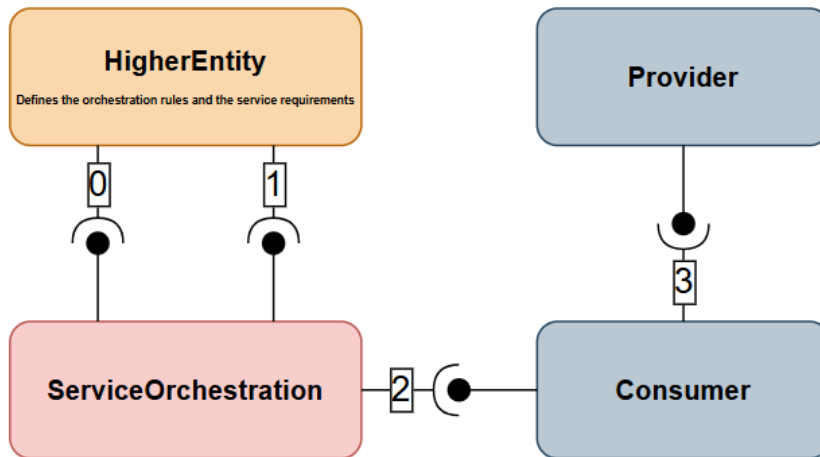
https://aitia-iiot.github.io/ah5-docs-java-spring/core_systems/service_orchestration_dynamic/

The 5th generation is also introducing the *push* type of orchestration (beside the traditional *pull* orchestration). It works the way that a consumer system is subscribed with a service requirement in order to receive new orchestration results every time when a higher entity triggers a related orchestration process. Hence, the actual matching service instances are being sent to the consumer without its direct request (See Figure 3.2).



- 1) Consumer's orchestration for everything what is needed [orchestration:pull]
- 2) ServiceOrchestration's discovery for the service details [serviceDiscovery:lookup]
- 3) ServiceOrchestration's request for authorization token(s) [authorizationToken:generate]
- 4) Actual service consumption

Figure 3.1 – ConsumerAuthorization Core system in action



- 0) A higher entity subscribes the consumer [orchestrationPushManagement:subscribe]
- 1) A higher entity triggers the orchestration process [orchestrationPushManagement:trigger]
- 2) Consumer receives the new orchestration results via its notify endpoint.
- 3) Consumer connects to the new provider

Figure 3.2 – How PushOrchestration works

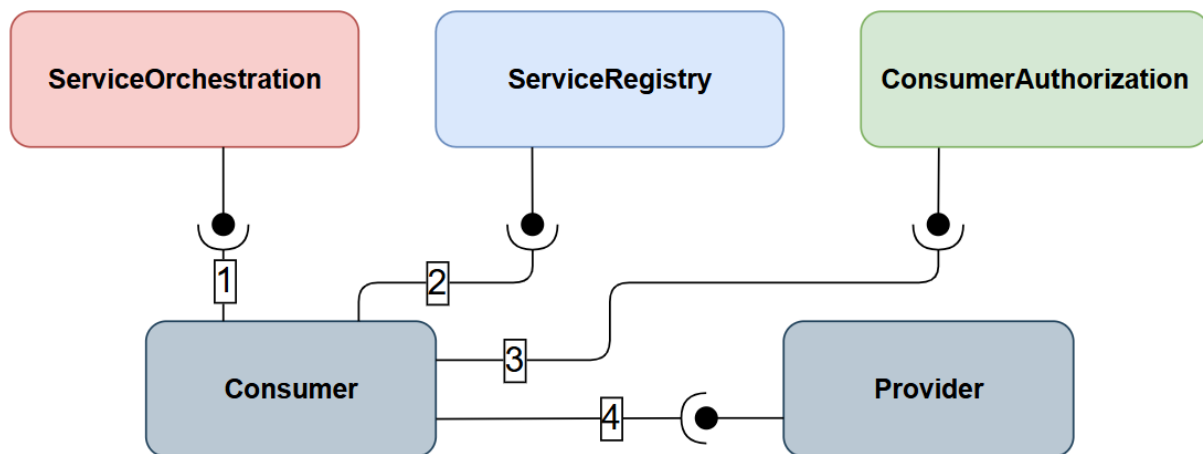
3.2.3 Simple Store Service Orchestration Core System

Arrowhead-serviceorchestration-simple

SimpleStoreServiceOrchestration Core System implements the simple-store orchestration strategy, which is a maintained list of peer-to-peer rules for consumer, provider, and service triplets. The simple store does not contain any service access information and not even consults with other core or support systems. Therefore, consumers have to manage every step themselves. Figure 3.3 shows such a scenario in a basic local cloud.

This core system provides four services: *serviceOrchestration*, *serviceOrchestrationPushManagement*, *serviceOrchestrationLockManagement* and *serviceOrchestrationHistoryManagement*. Consumers can choose between HTTP and MQTT interfaces.

https://aitia-iiot.github.io/ah5-docs-java-spring/core_systems/service_orchestration_simple_store/



- 1) Consumer's orchestration only for a service identifier [orchestration:pull]
- 2) Consumer's discovery for the service details [serviceDiscovery:lookup]
- 3) Consumer's request for authorization token(s) [authorizationToken:generate]
- 4) Actual service consumption

Figure 3.3 – The simple service Orchestration procedure

3.2.4 Authentication Core System

arrowhead-authentication

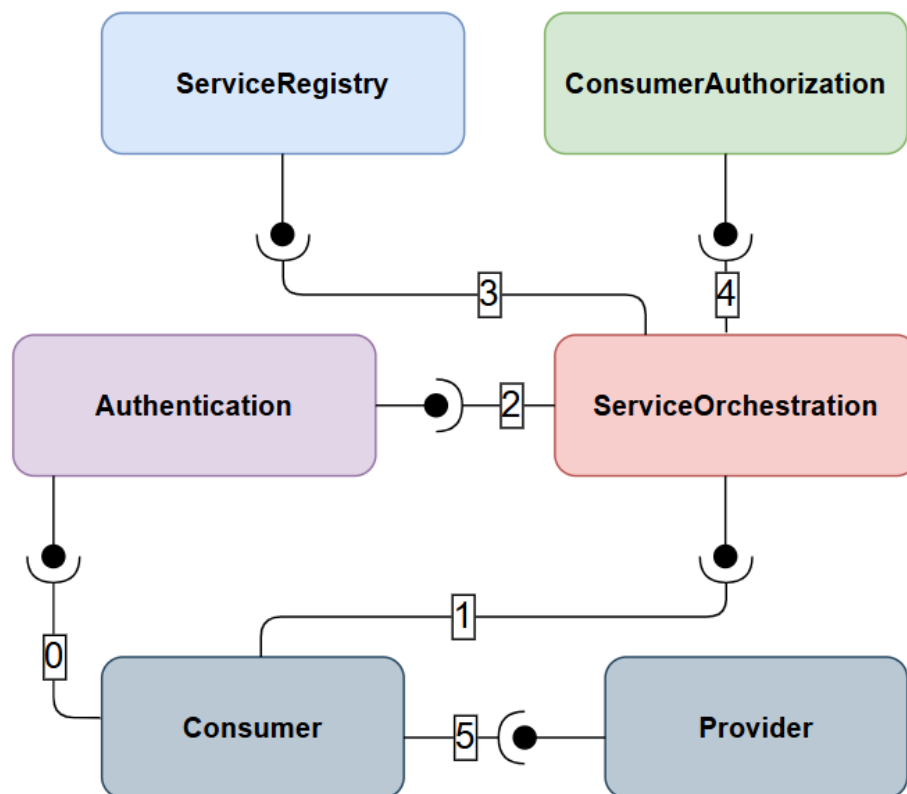
The 5th generation offers multiple authentication policies. One of them is the so called “outsourced” policy that requires a common identity service being integrated. This service allows providing various types of credentials to perform an authentication procedure, and always results a unique identity token. This way any kind of authentication solution is easily integratable into a Local Cloud.

https://aitia-iiot.github.io/ah5-docs-java-spring/api/authentication_policy/

The Authentication Core System provides this identity service with requiring system name and password combination as credentials. The implementation allows adding further type of credentials as well. Figure 3.4 shows a service orchestration scenario in a Local Cloud with Authentication Core System deployed.

This core system provides two services: *identity* and *identityManagement*. Consumers can choose between HTTP and MQTT interfaces.

https://aitia-iiot.github.io/ah5-docs-java-spring/core_systems/authentication/



- 0) Consumer's login for an identity token [identity:login]
- 1) Consumer's orchestration for everything what is needed [orchestration:pull]
- 2) ServiceOrchestration verifies the consumer's identity token [identity:verify]
- 3) ServiceOrchestration's discovery for the service details [serviceDiscovery:lookup]
- 4) ServiceOrchestration's request for authorization token(s) [authorizationToken:generate]
- 5) Actual service consumption

Figure 3.4 – Service Orchestration with the Authentication Core System deployed

The identity token is intended to share only with Core and Support systems and should never be sent to application systems. When a provider system needs authenticated service consumption, it can be achieved by requiring authorization token. Figure 3.5 shows a scenario using both, identity and authorization tokens.

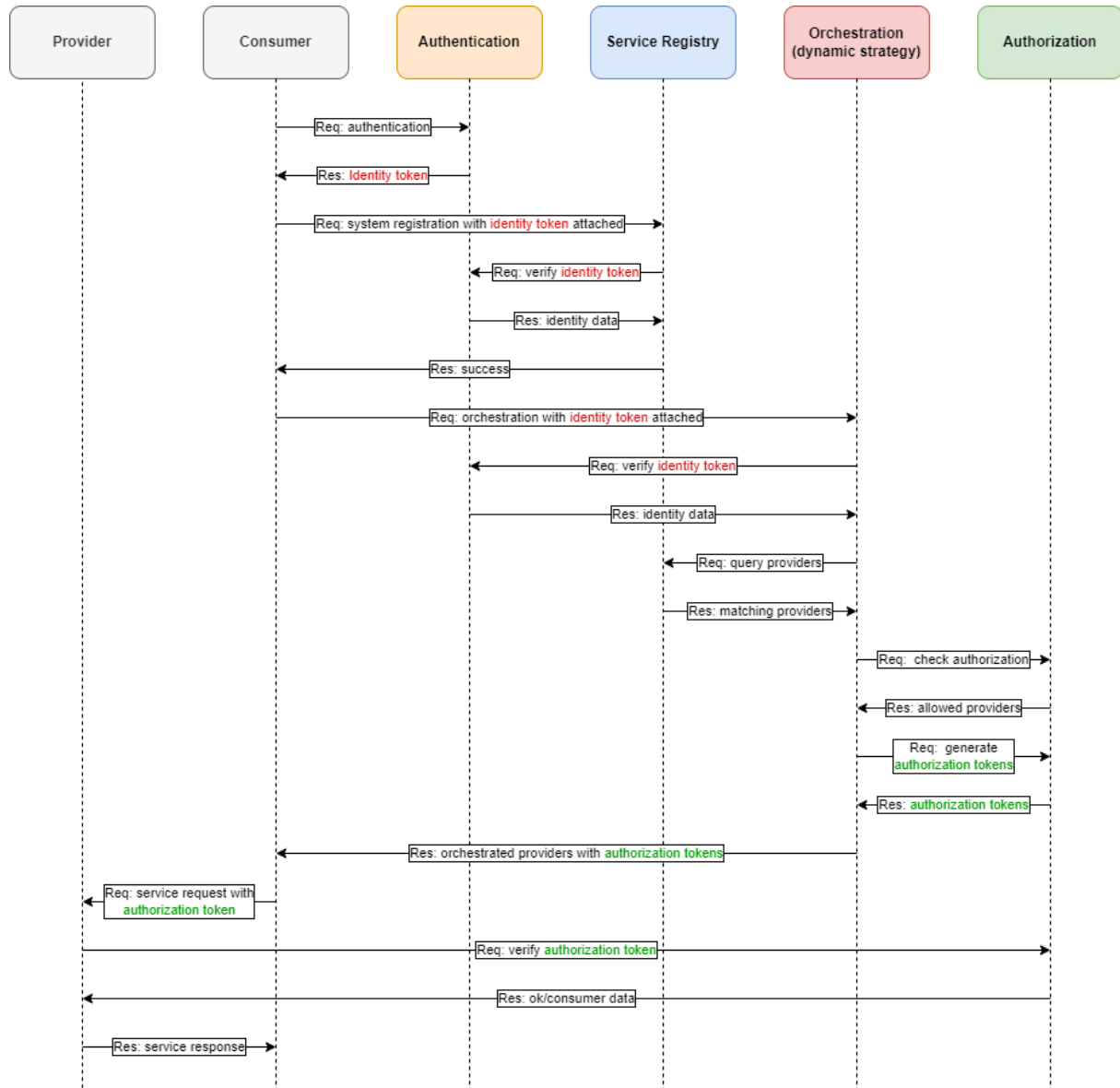


Figure 3.5 – Procedure using both identity and authorization tokens

3.2.5 Blacklist Support System

arrowhead-blacklist

The 5th generation introduces the Blacklist Support system, which makes it possible to ban systems from the Local Cloud, so that their requests will not be served. This system provides storage functionality for the blacklist entries that contain information about the ban for each system.

Every other Core & Support system (except the Authentication Core System) is able to check the requester application systems automatically with this Blacklist Support System if configured to do so.

This support system provides two services: *blacklistDiscovery* and *blacklistManagement*. Consumers can choose between HTTP and MQTT interfaces.

https://aitia-iiot.github.io/ah5-docs-java-spring/support_systems/blacklist/

3.2.6 Translation Manager Support system

arrowhead-translation-manager

The 5th generation introduces the TranslationManager Support system, which exists to help the communication between a consumer and a provider if there is no common interface between them and/or they are using different data models. The TranslationManager achieves this goal by building a translation bridge using various translator systems (application systems) as building blocks. An other example of such help is the ability to determine the possibility to build such bridge with the currently available translators.

This support system provides three services: *translationBridge*, *translationReport* and *translationBridgeManagement*. Consumers can choose between HTTP and MQTT interfaces.

https://aitia-iiot.github.io/ah5-docs-java-spring/support_systems/translation_manager/

Figure 3.6 shows how a translation bridge is established during a service orchestration procedure.

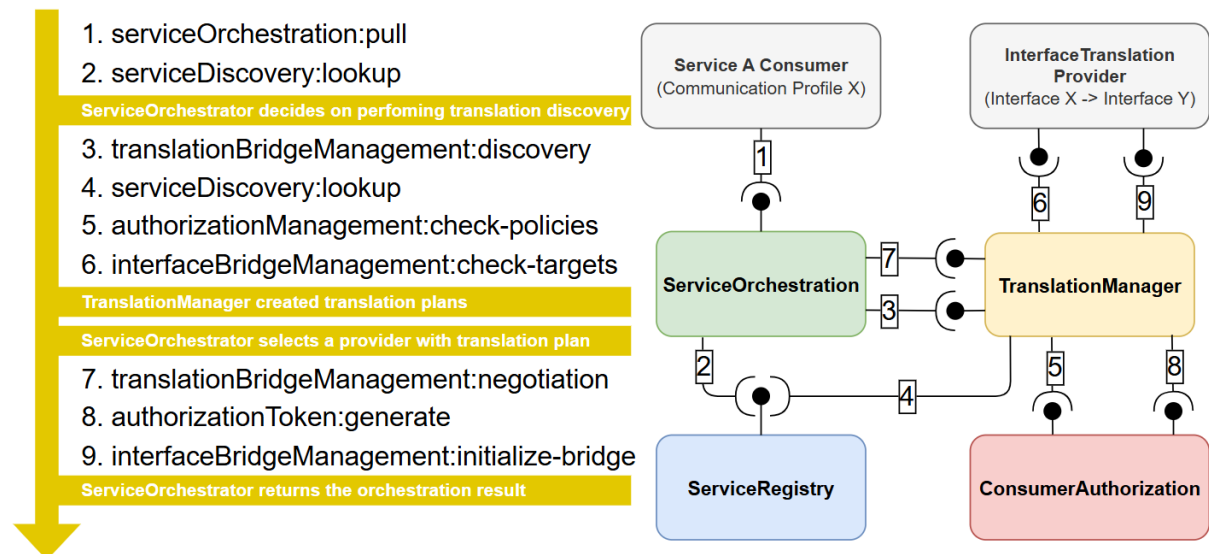


Figure 3.6 – The Translation Bridge concept in action

Figure 3.7 shows how a translation bridge works during a service consumption attempt with token level security.

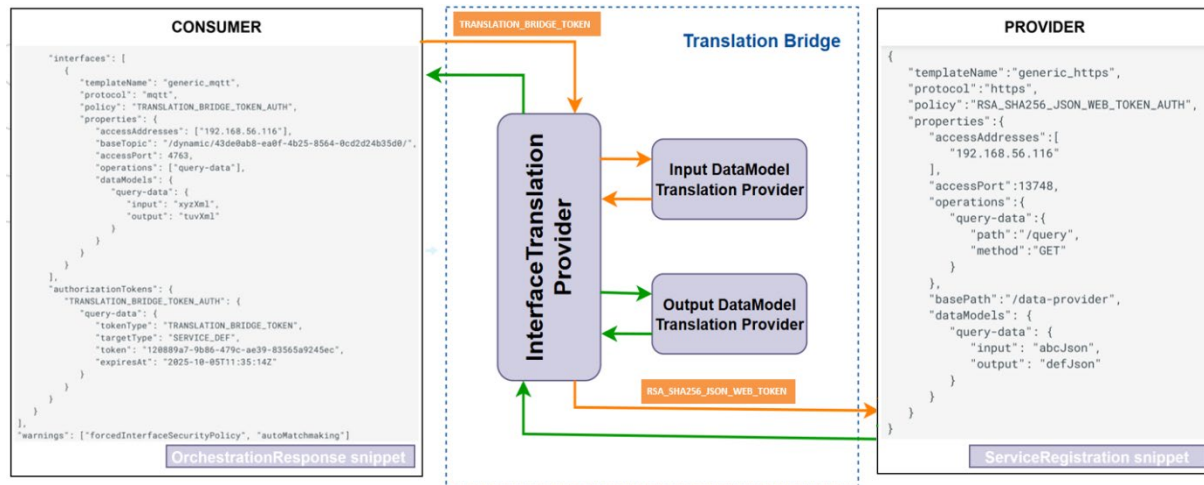


Figure 3.7 - Translation Bridge during a service consumption attempt with token level security

Also, it supports from generic_http(s) to generic_http(s) in order to achieve translation when only data models are different.

Furthermore, we have created a data model translator adapter that is capable to expose any kind of translation algorithm written in Python.

https://aitia-iiot.github.io/ah5-docs-java-spring/tools_assets/translation/translation_providers/

Translation Demo

Using the above mentioned building blocks a demo was created and showcased in the Lisbon DeepTech Workshop (28.10.2025). The use case scenario was inspired by work being done in WP4. Meaning that the produced data which is using a corporate custom data model – XML kind format and custom semantics – have to be stored in the IPC2581 industrial standard (printed circuit board and assembly manufacturing description data and transfer methodology). Moreover, the data producer is using generic_mqtt communication interface while the data saver provider offers only generic_http (See Figure 3.8).

The data model translator is a semantic AI based Python solution (developed in WP4) which is integrated into the Local Cloud using the above mentioned adapter system.

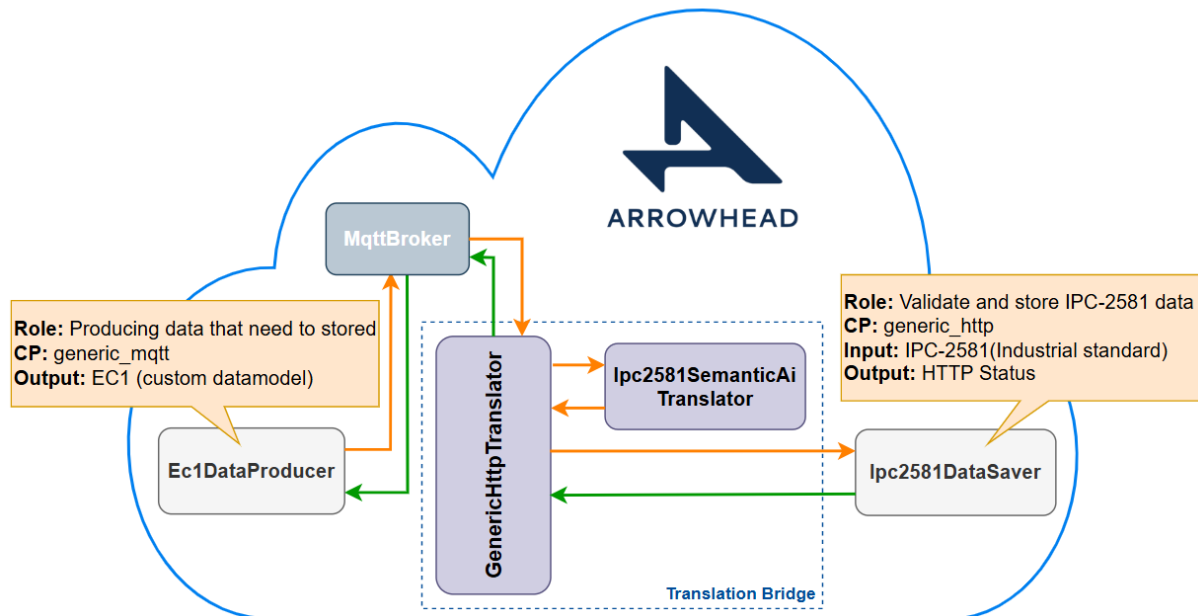


Figure 3.8 – Translation Demo setup

3.2.7 Device QoS Evaluator Support System

arrowhead-device-qos-evaluator

5th generation allows deploying multiple Quality of Service Support Systems into a Local Cloud and these QoS systems can collect different metrics and offer sorting and filtering using those metrics. This is exposed via a common *qualityEvaluation* service which the DynamicServiceOrchestration Core System is able to utilize during the orchestration process.

The DeviceQoSEvaluator Support System enables to analyze and evaluate the provider candidates based on the state of the device, the candidate runs on. The current version collects the aggregated values (minimum, maximum, mean, median and latest - within a specified measurement window) of the following metrics:

- RTT (Round-trip Time)
- Total CPU Load
- Used Memory
- Network Ingress Load
- Network Egress Load

This support system provides two services: *qualityEvaluation* and *deviceQualityDataManagement*. Consumers can choose between HTTP and MQTT interfaces.

https://aitia-iiot.github.io/ah5-docs-java-spring/support_systems/device_qos_evaluator/

The RTT measurement can happen without any assistance from the application system side, since it is using High UDP port test (closed port connection attempt).

The collection of the other metrics requires running a light-weight Python program on the devices. This tool is released as part of the DeviceQoS Evaluator Support System package.

Every calculated value is normalized between 0 and 100 (the lower the better). During the evaluation the requester specifies that which calculated values should be included with what weight to determine the order of the provider candidates or to filter out those below a specified threshold. The sum of the weights is required to be ~ 1 . These conditions guaranties the QoS values are always between 0 and 100 as well.

4. Integration results in use-cases

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

This use case presents the application of Model-Based Enterprise (MBE) principles to enable interoperable and intelligent management of Product Value Networks across the complete lifecycle of aerospace electromechanical components. Its primary objective is to implement digital and automated strategies that enhance manufacturing productivity while ensuring end-to-end traceability of assets throughout the design, production, and operational phases. By integrating MBE methodologies with standardized modeling and translation tools, the use case enhances efficiency, reliability, and cross-platform interoperability within aerospace systems engineering.

One component of this use case focuses on the use of translators for data model transformation. Accordingly, during this final reporting period, work on translation services has continued, and dedicated translator components have been integrated to ensure seamless interoperability across the workflow. These include UNINOVA's DiTAG tool and AI-based translation services developed by AITIA. The following figure presents the overall architecture of the translation solutions implemented within the scope of this use case.

Use Case Description

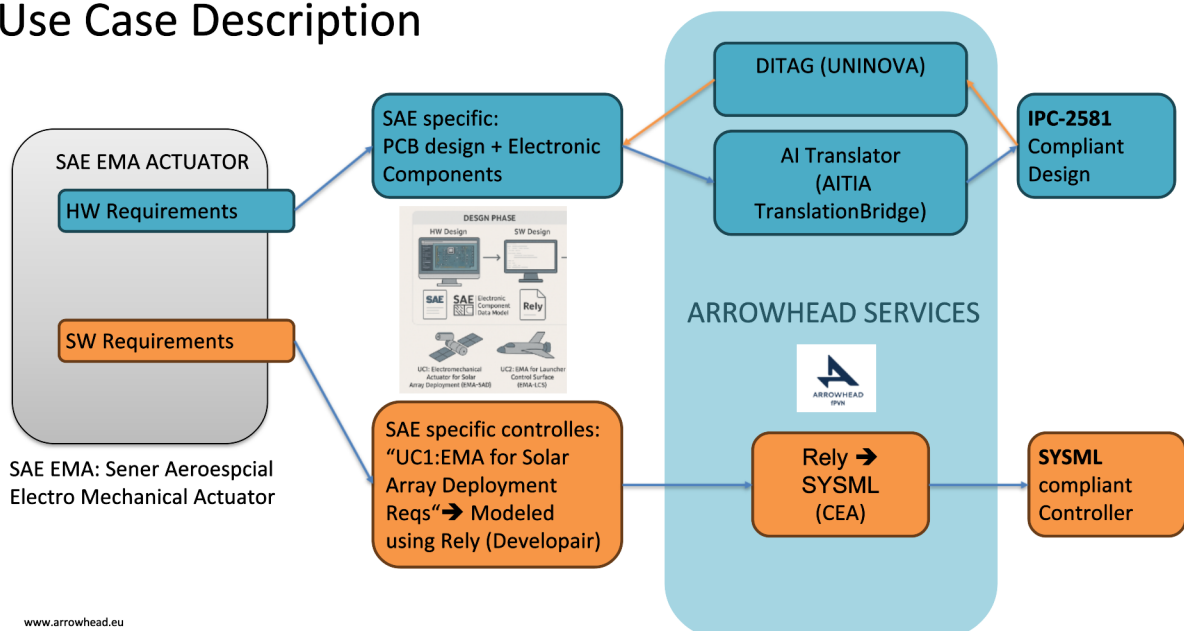


Figure 4.1 - SAE Use Case architecture for Data Model translation

Ontology-Based and AI-Based Translation Services

Within this context, translation services play a key role in enabling interoperability between heterogeneous data models. UNINOVA's DiTAG tool provides ontology-based translation from IPC-2581 to the SAE Electronic Component (EC) Data Model. By leveraging semantic

mappings, DiTAG ensures consistent alignment between standardized electronic design formats and domain-specific aerospace models, enabling reliable data exchange and integration across systems.

Complementing this approach, AITIA has investigated AI-based translation techniques to support the reverse translation - from the SAE EC Data Model back to IPC-2581 - thus enabling bidirectional interoperability. In the recent period, AITIA has focused on standardizing SAE's custom EC XML datasets by automatically mapping them to IPC-2581-compliant representations. The proposed solution follows a two-step AI-driven approach. First, a mapping definition table is generated using Retrieval-Augmented Generation (RAG), combining predefined prompts with iterative retrieval of relevant IPC-2581 elements from a vector store. This process uses semantic descriptions of custom XML tags to identify suitable standardized elements. Second, the generated mapping definitions are applied - using predefined functions or Large Language Models (LLMs) - to perform the automatic translation and generate standardized XML content.

The AITIA translation system, referred to as the *Translation Manager*, has progressed beyond the initial planning phase and is now part of the ongoing implementation activities. Although it was not included in the earliest prototype, it is currently being integrated alongside the IPC-2581 and SAE data models, reinforcing collaboration with AITIA and significantly expanding the overall translation capabilities of the ecosystem. In the resulting setup, DiTAG performs ontology-based translation from IPC-2581 to the SAE EC Data Model, while AITIA's AI-based translator supports the reverse direction, enabling a robust, bidirectional translation workflow. The Arrowhead translation concept applied to this use case is presented by Figure 4.2.

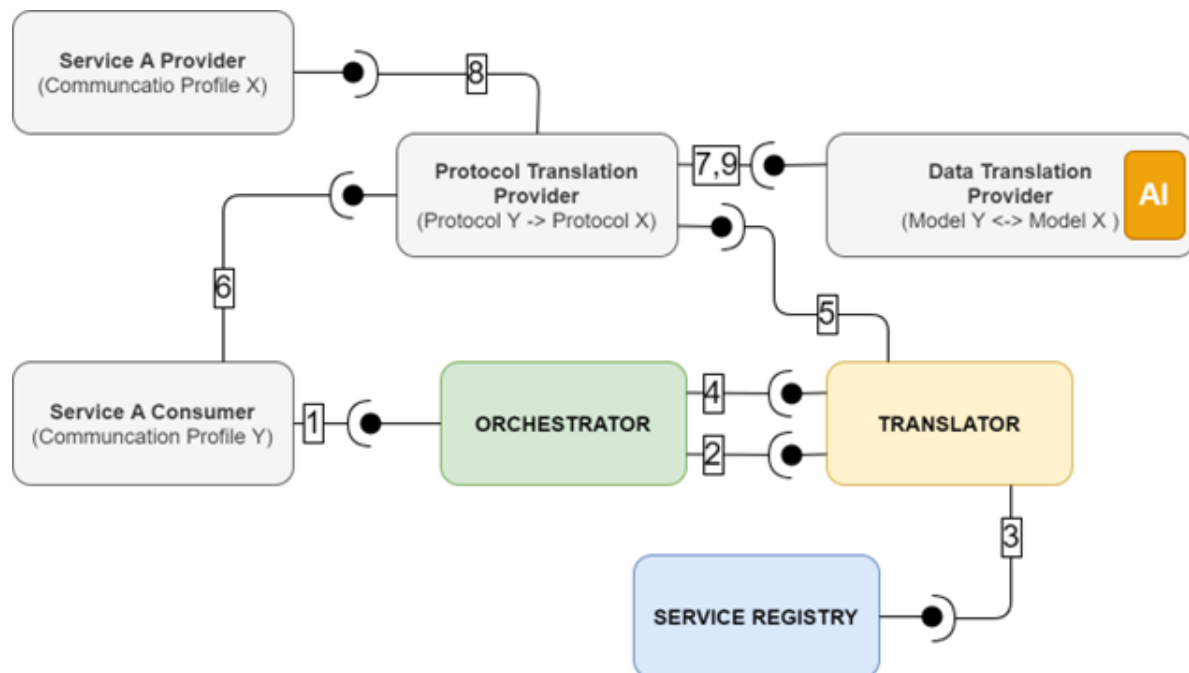


Figure 4.2 – Applying The Translation Services to the SAE Use Case

In addition, part of the use case is based on implementing a quantum inspired algorithm to optimize logistics problems. During the third and final reporting period, the integration of the Multiverse logistics optimisation use case into the Arrowhead ecosystem has been successfully completed. Building on a microservice-based approach, the final architecture of the use case is shown in the next figure:

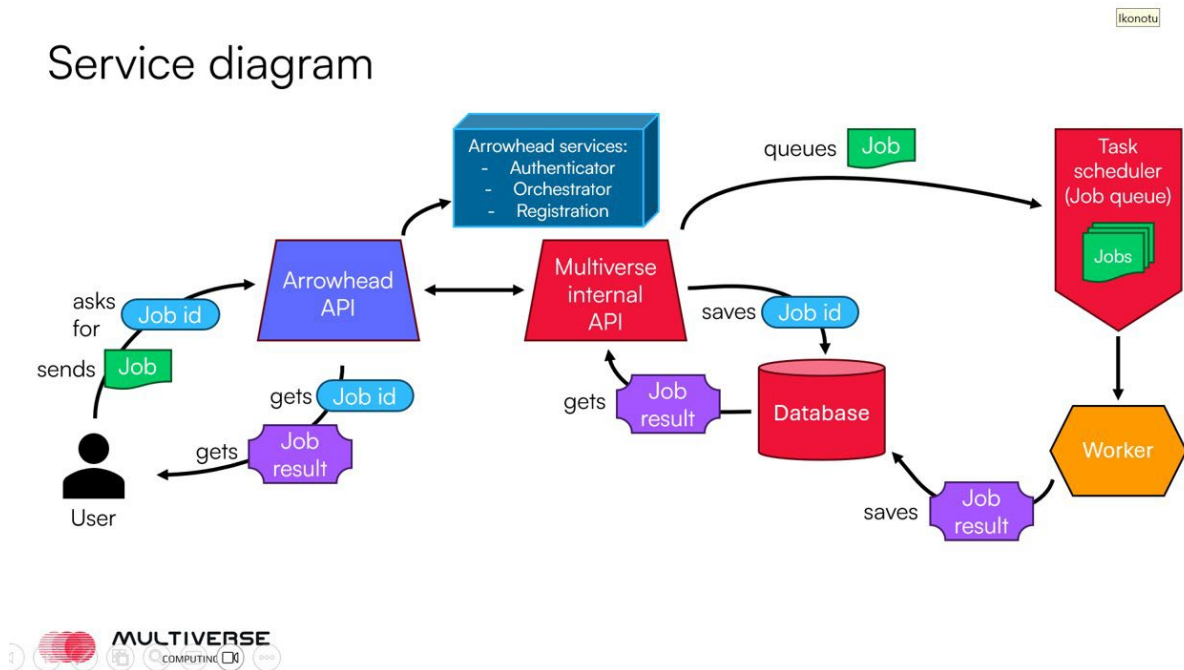


Figure 4.3 - Logistics Optimisation Arrowhead Service architecture

This consolidated architecture specifies the service interactions required for the demonstration, including the deployment of Multiverse’s quantum-inspired optimisation algorithms as Arrowhead-compliant services. The optimisation solution has been fully integrated into the Arrowhead ecosystem as a service, enabling seamless interaction with the rest of the system architecture.

The final architecture of the demonstrators developed within this use case illustrates the integration of quantum-inspired optimisation services, ontology-based and AI-based translation components, standardized data models, and Arrowhead core services. This integrated setup provides a complete design-to-validation workflow that demonstrates interoperable, intelligent, and regulation-aware management of aerospace Product Value Networks.

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

4.2.1 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.

This demonstrator uses Arrowhead v4.6, at this stage, and plans to validate v5.1 for the next phase.

Figure 4.4 presents the proposed model from AVL.



Figure 4.4 – Measurement data processing pipeline

Based on the aims of the Arrowhead framework's microservice architecture, the subsystems (presented by Figure 4.4) 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 4.5.).

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

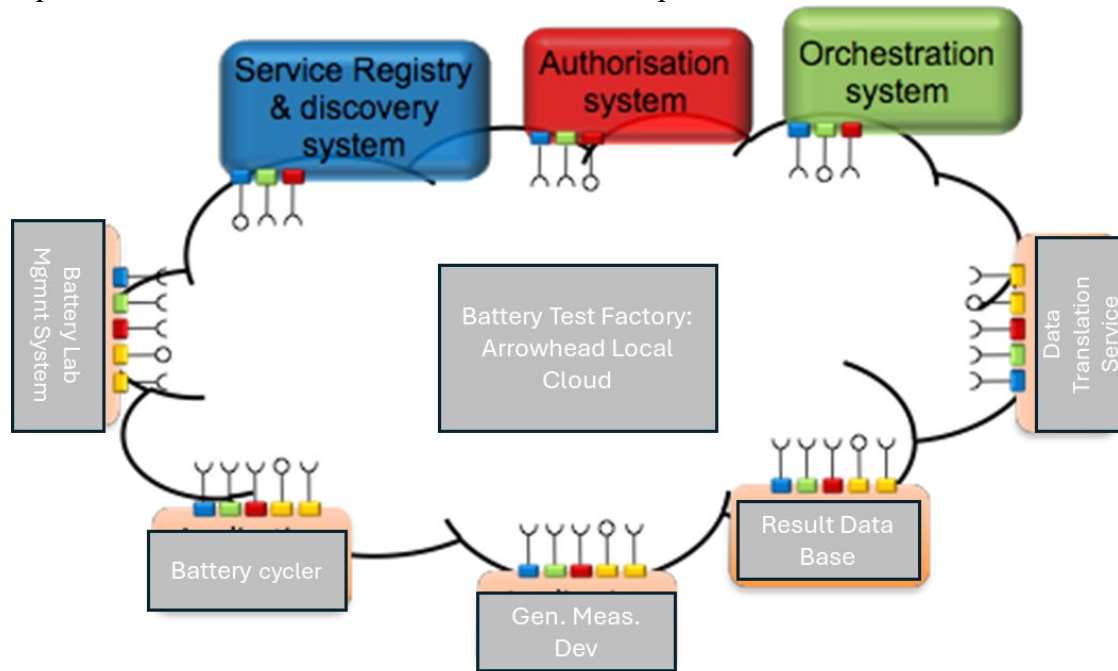


Figure 4.5 – Arrowhead microservice concept regarding UC 1.6-Testlab

The overall integrated system architecture is shown by Figure 4.6. In there, the Arrowhead core systems reside on the right side of the Figure, providing Local Arrowhead Cloud services to the various levels of the data management, test factory management, digital twin lifecycles and the DITEST elements.

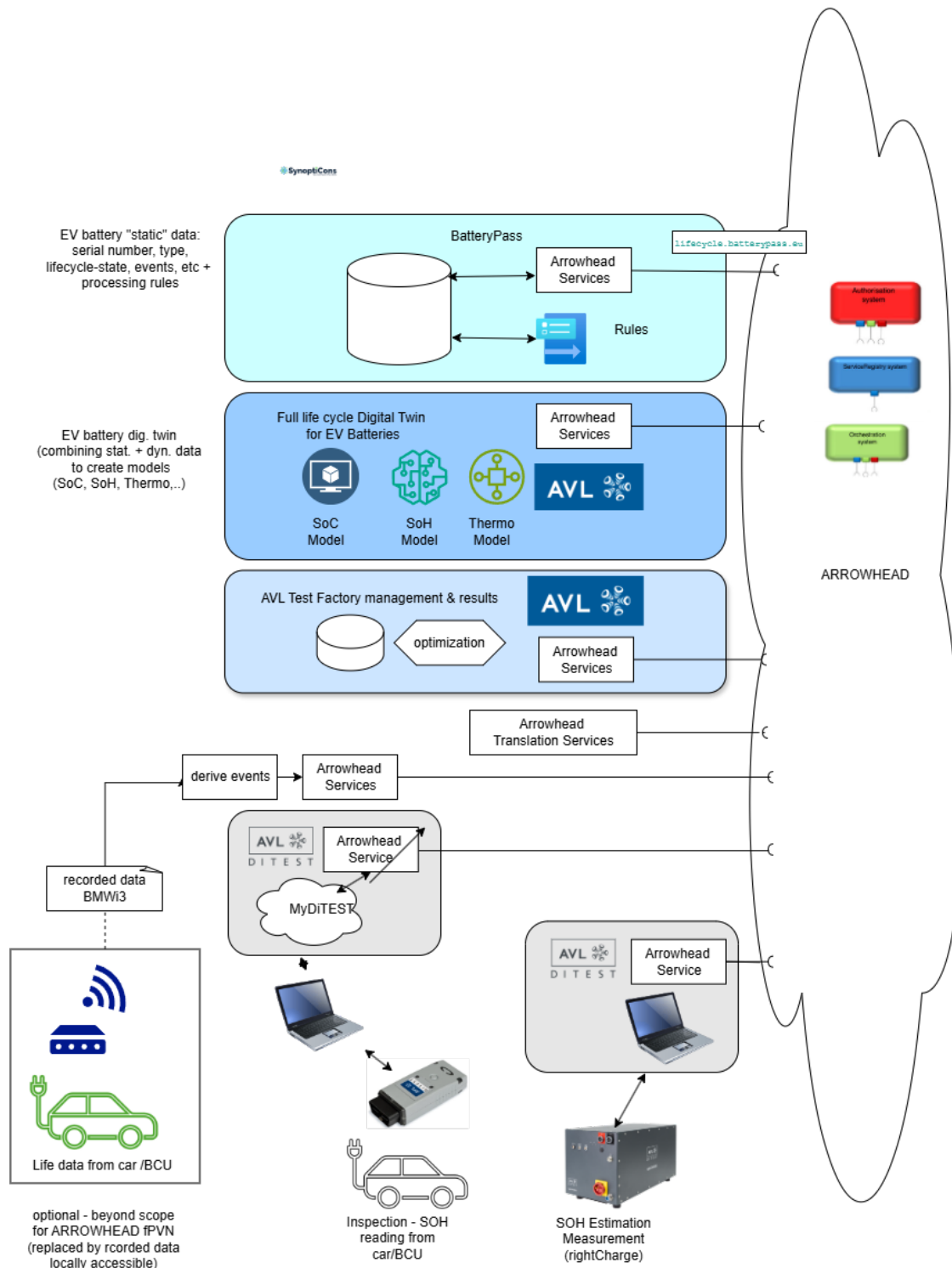


Figure 4.6 - Overall UC 1.6 Architecture with Arrowhead microservices

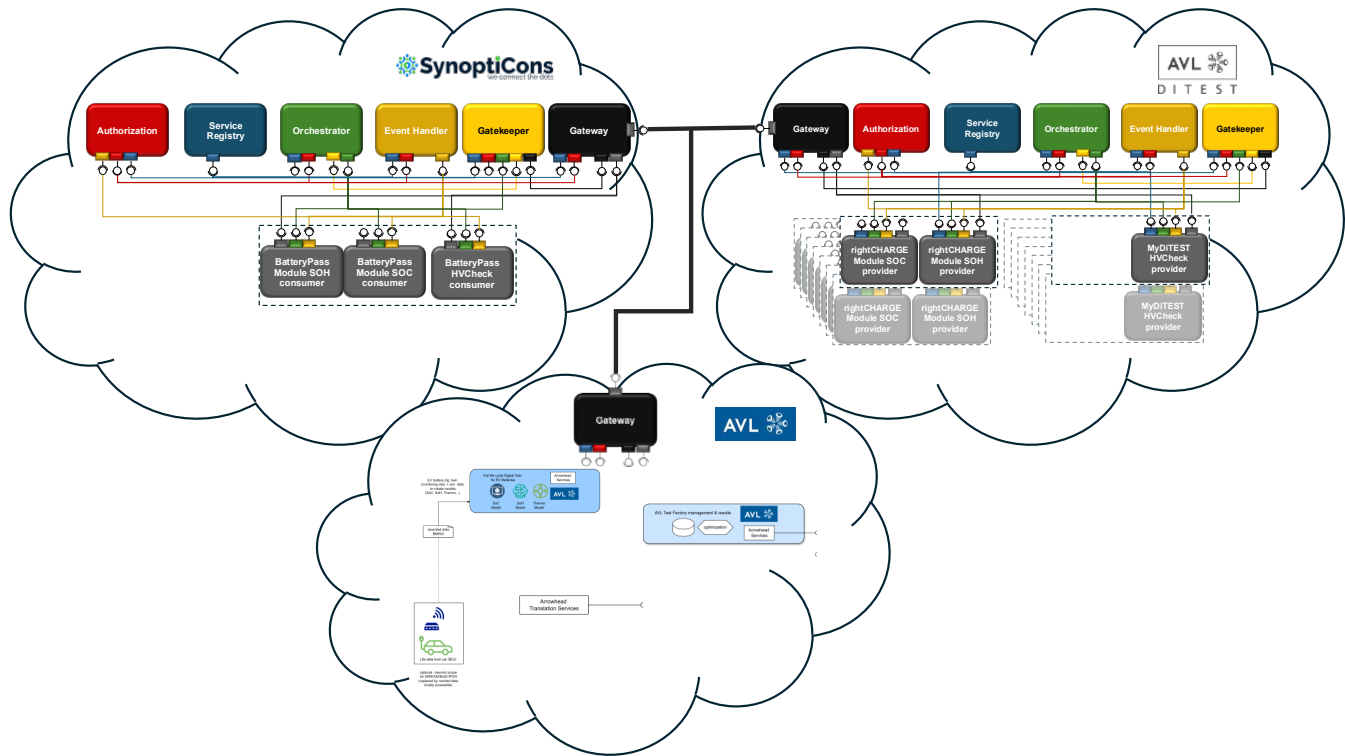


Figure 4.7 – Connecting data silos with Arrowhead microservices for UC1.6

The conceptual plan of basing the communication between the different systems / data silos on microservices via the Arrowhead framework is shown by Figure 4.7. Implementation is ongoing for Y3.

4.2.2 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, ... State 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 4.8.)

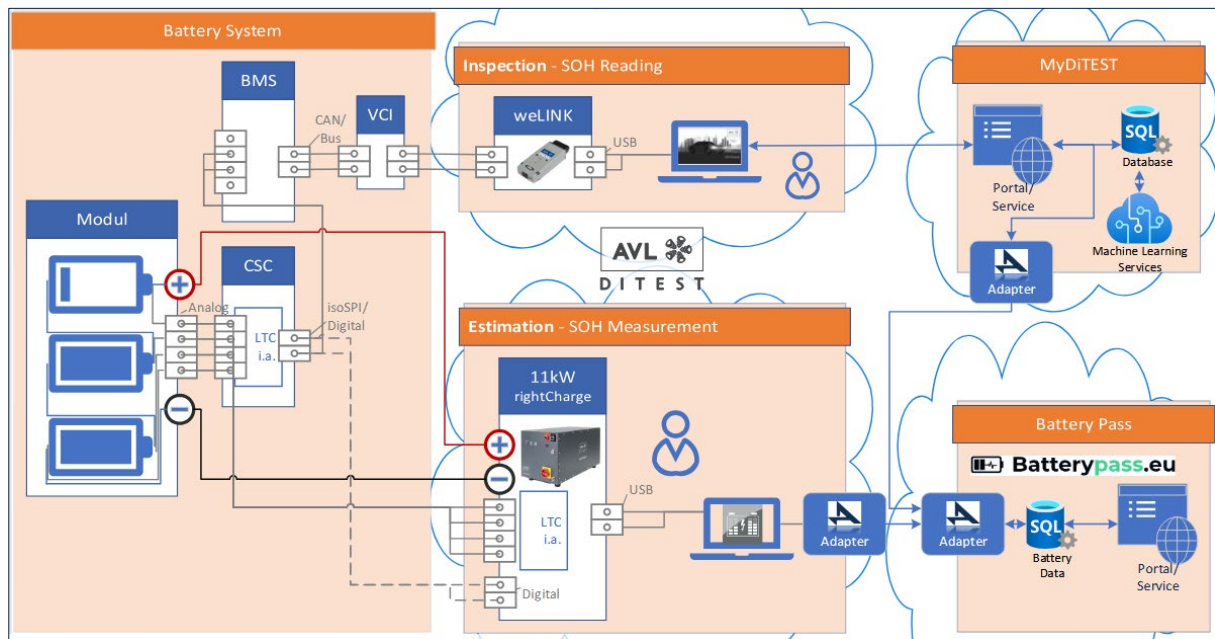


Figure 4.8 – Arrowhead microservice concept regarding UC 1.6-EoL Treatment

4.3 UC 2.6 – Architecture Concept: Command Robustness Screening API

4.3.1 Architectural Overview

Use Case 2.6 implements a microservice-based screening API for evaluating the noise robustness of spoken command vocabularies prior to deployment in industrial environments. The architecture follows the Eclipse Arrowhead Local Cloud model and adopts a service-oriented composition in which independent application systems provide command storage, noise storage, computational analysis, and user interaction. These application systems are integrated through Arrowhead Core and Support Systems, enabling service registration, discovery, authorization, orchestration, and QoS-aware provider selection.

The overall objective is to support scalable, human executed pre-deployment analysis of command(s), while allowing independent development of data repositories, analysis algorithms, and user interfaces.

Within a single Arrowhead Local Cloud, the UC2.6 architecture consists of:

- CommandLibrary Application System
- NoiseLibrary Application System
- DataProcessor Application System (multiple instances)
- UserInterface Application System (multiple instances)
- Arrowhead Core Systems: ServiceRegistry, DynamicServiceOrchestration, ConsumerAuthorization
- Arrowhead Support System: DeviceQoSEvaluator

All systems communicate exclusively via Arrowhead service interfaces using HTTP or MQTT.

4.3.2 Application Systems and Responsibilities

4.3.2.1 CommandLibrary System

The CommandLibrary system stores candidate spoken commands created or curated by SDU. Additionally, user may upload commands. These commands may be textual representations, phonemic representations, or associated audio files.

The system exposes a **commandDiscovery** service with the following operations:

- **query**: retrieve available command set or individual commands
- **upload**: store new commands or command sets
- **delete**: remove stored commands

A single instance of the CommandLibrary is within the Local Cloud, acting as a continual repository.

The CommandLibrary does not perform any signal processing or analysis; it serves purely as a data provider.

4.3.2.2 NoiseLibrary System

The NoiseLibrary system stores industrial noise corpora collected or generated by SDU, including metadata such as environment type, spectral information, and recording conditions.

The system exposes a **noiseDiscovery** service with operations:

- **query**: retrieve available noise corpora
- **upload**: store new noise recordings. In a later iteration, these recordings could be uploaded into the NoiseLibrary.
- **delete**: remove stored noise data

As with the CommandLibrary, a single instance is assumed. The NoiseLibrary acts as a persistent provider of noise data.

4.3.2.3 DataProcessor System

The DataProcessor system performs the core computational task of evaluating command robustness under noise.

Multiple instances of DataProcessor may be deployed within the Local Cloud to enable horizontal scaling.

The system exposes an **analyzeCommandInNoise** service with three operations:

- **start**: initiate an analysis job
- **result**: retrieve the results of a completed job
- **abort**: cancel an ongoing job

The **start** operation receives references to:

- A command set identifier(s) or upload reference(s)
- A noise corpus identifier(s) or upload reference(s)
- Optional analysis profile parameters

The **result** operation returns a structured result object including:

- Overall Robustness Index (RI); a score of robustness based on our developed algorithm and a visual colour coding (traffic light; red-yellow-green) classification based on recommendation for use
- Optional per-segment or per-command scores
- Optional diagnostic metadata

The **DataProcessor** consumes the **query** operations of both **commandDiscovery** and **noiseDiscovery** in order to load the selected commands and noise data.

The **DataProcessor** is conceptually stateless with respect to long-term storage; persistent data remain in the **CommandLibrary** and **NoiseLibrary**. Temporary job state may be cached locally.

4.3.2.4 **UserInterface System**

The **UserInterface** system provides a graphical interface for human users.

Multiple instances may be deployed concurrently.

The **UserInterface** consumes:

- **commandDiscovery.query** and **commandDiscovery.upload**
- **noiseDiscovery.query** and **noiseDiscovery.upload**
- **analyzeCommandInNoise.start**
- **analyzeCommandInNoise.result**

The **UserInterface** does not perform orchestration logic itself. Instead, it requests orchestration from Arrowhead Core Systems prior to invoking provider services.

4.3.3 **Service-Oriented Interaction Flow**

A typical interaction proceeds as follows:

- 1 The **UserInterface** queries the **CommandLibrary** for available command sets.
- 2 The **UserInterface** queries the **NoiseLibrary** for available noise corpora.
- 3 Optionally, the user uploads new commands and/or noise through the respective upload services.
- 4 The **UserInterface** requests orchestration for the **analyzeCommandInNoise** service from the **DynamicServiceOrchestration Core System**.
- 5 The **DynamicServiceOrchestration Core System** discovers candidate **DataProcessor** providers from the **ServiceRegistry**.
- 6 The **ConsumerAuthorization Core System** filters out providers for which the **UserInterface** lacks permission.
- 7 The **DeviceQoS Evaluator Support System** ranks remaining **DataProcessor** candidates based on device-level metrics.
- 8 The **DynamicServiceOrchestration Core System** returns the selected provider endpoint.
- 9 The **UserInterface** invokes **analyzeCommandInNoise.start**.
- 10 The **UserInterface** periodically invokes **analyzeCommandInNoise.result** until the result is available.

This flow ensures that each analysis request is routed to an authorized and currently suitable DataProcessor instance.

4.3.4 Orchestration and QoS-Aware Provider Selection

Because robustness analysis may be computationally expensive, UC2.6 relies on Arrowhead's DynamicServiceOrchestration in combination with the DeviceQoSEvaluator Support System.

DataProcessor instances are continuously monitored by the DeviceQoSEvaluator, which collects metrics such as CPU load, memory usage, and round-trip time. During orchestration, these metrics are used to rank candidate providers.

This enables:

- Automatic load balancing
- Smooth scaling by deploying additional DataProcessor instances
- No hard-coded provider addresses in the UserInterface

4.3.5 Security and Authorization

All service invocations follow Arrowhead intra-cloud authorization.

The UserInterface obtains authorization tokens via the ConsumerAuthorization Core System. Tokens are attached to service invocations toward CommandLibrary, NoiseLibrary, and DataProcessor.

Upload operations may be restricted to specific roles or identities, while query and analysis operations may be open to a broader set of authorized consumers.

4.3.6 Mapping from User Workflow to Services

The user-facing workflow is represented as the following service interactions:

Workflow Step	Consuming System	Providing System	Service
Select library commands	UserInterface	CommandLibrary	commandDiscovery.query
Upload own commands	UserInterface	CommandLibrary	commandDiscovery.upload
Select library noise	UserInterface	NoiseLibrary	noiseDiscovery.query
Upload own noise	UserInterface	NoiseLibrary	noiseDiscovery.upload
Start analysis	UserInterface	DataProcessor	analyzeCommandInNoise.start
Retrieve result	UserInterface	DataProcessor	analyzeCommandInNoise.result

This mapping ensures traceability from user interaction to microservice-level behavior.

4.3.7 Scalability and Evolution

UC2.6 scales primarily along the DataProcessor dimension. As the number of concurrent users grows, additional DataProcessor instances can be deployed and registered without modifying any other system.

The analysis algorithm itself is encapsulated entirely within the DataProcessor system. Improvements to the robustness model, scoring methods, or feature extraction pipeline therefore do not impact the CommandLibrary, NoiseLibrary, or UserInterface systems.

4.3.8 Relation to Arrowhead Translation Capabilities

UC2.6 currently assumes a common internal data model across application systems and therefore does not require TranslationManager integration in the initial prototype.

However, the architecture permits future insertion of TranslationManager if alternative phonetic representations, feature formats, or external corpora with incompatible schemas are introduced.

4.3.9 Summary

UC2.6 demonstrates an Arrowhead-compliant microservice architecture for a speech-based pre-deployment screening tool, combining human-in-the-loop interaction, QoS-aware orchestration, and scalable analysis services within a Local Cloud. The architecture illustrates how domain-specific signal-processing functionality can be cleanly embedded into the Arrowhead service ecosystem while preserving modularity, scalability, and independent evolution of components.

4.4 Process industry (WP9, UC1.9, 2.9 and 3.9) - Integrating information systems in process industry fPVNs

In the process industry fPVNs, Eclipse Arrowhead service framework features are used as a standard, agreed way of discovering and binding select services. The benefits of the microservice framework are especially in the dynamic late binding of various services used in complex integrations as well as in the unified model of authorizing systems in value networks spanning multiple stakeholders. In all use cases 1.9, 2.9 and 3.9 technical information systems are registered within Eclipse Arrowhead for discovery and binding in the value networks. In addition, the new features of embedding translation capabilities are seen beneficial and supporting the concept of dynamically injecting translations and thus directly increasing interoperability, as also envisioned and designed in the use case 3.9 technical information exchange process (see figure 4.9).

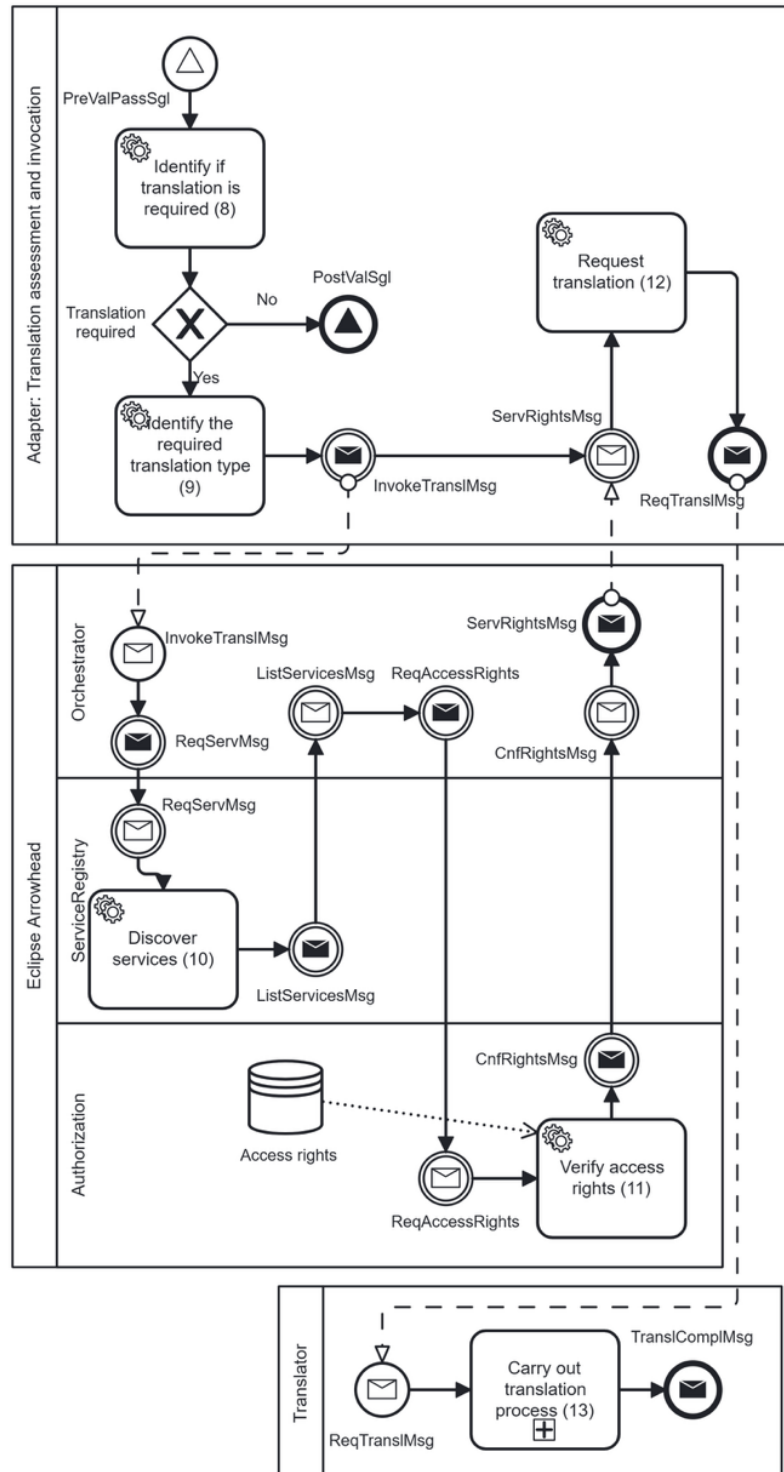


Figure 4.9 – Arrowhead microservices in the use case 3.9 technical information exchange and concept of dynamic injection of translations.

5. 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] and [6].

This target has been achieved by realizing 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.



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?"*

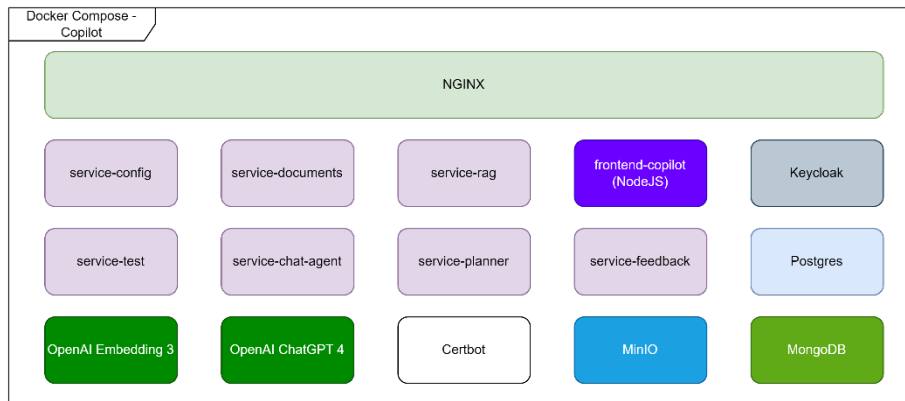


Figure 6.2 – The main elements of the Arrowhead Expert Copilot [5]

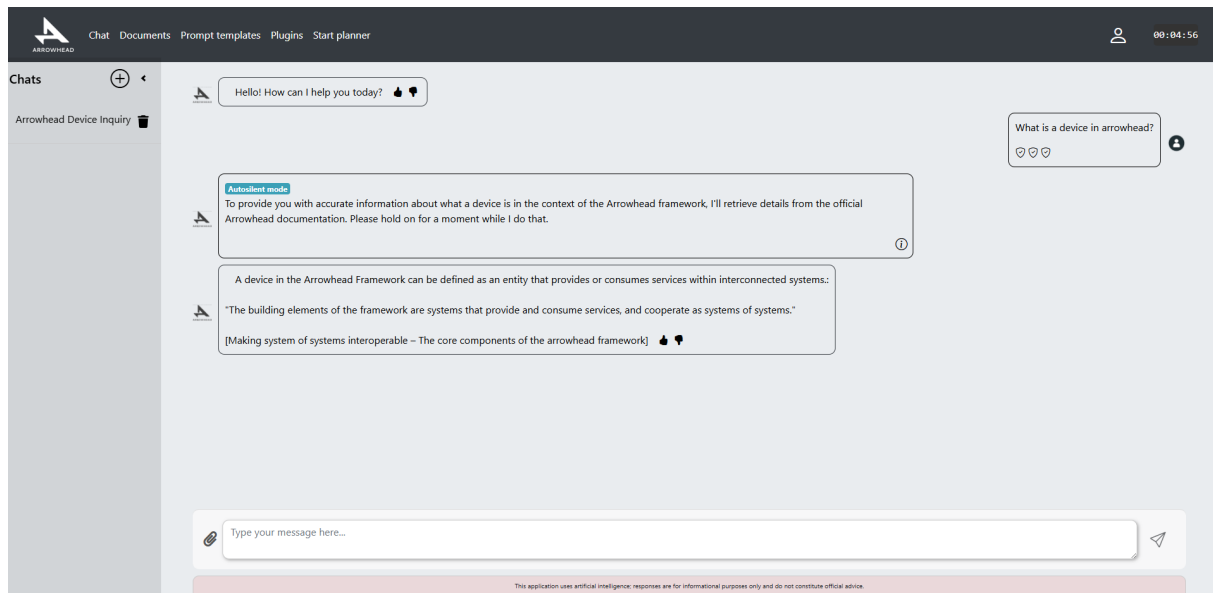


Figure 6.3 – The screenshot of the Arrowhead Expert Copilot [5]

The Arrowhead Management Copilot is able to plan and execute management and operation tasks real-time, in the working Arrowhead Local Clouds (for which it has access to). Its screenshot evidence is shown by Figure 6.4. Further details on the system can be found in [6]. A demonstration video was published on Eclipse Arrowhead Youtube under [7]

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"*.

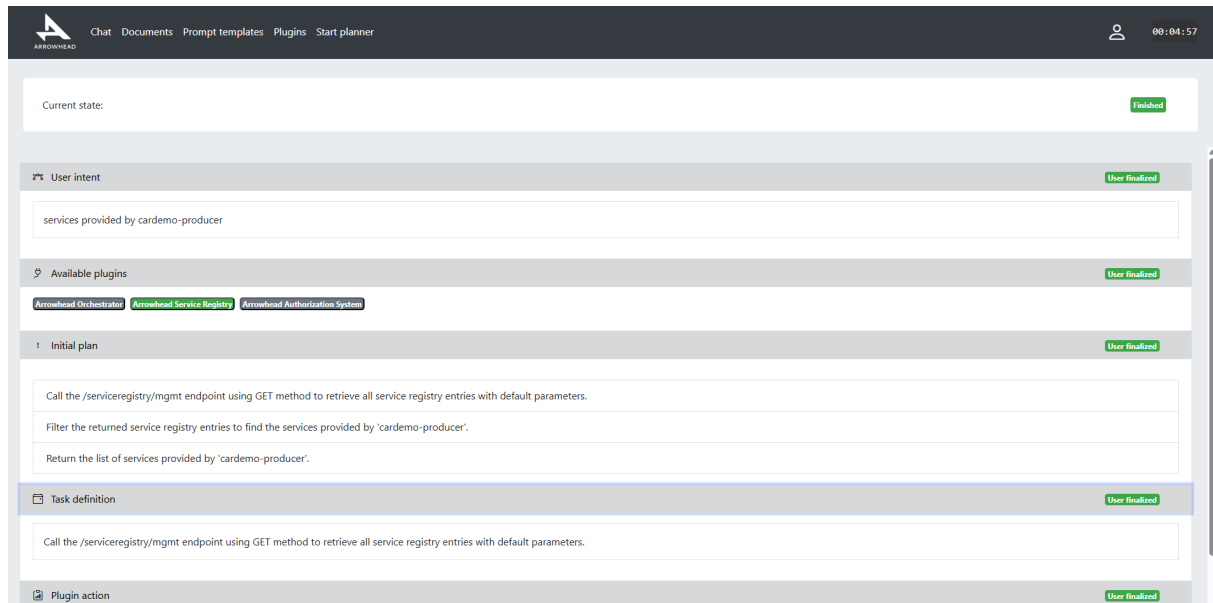


Figure 6.4 – The main elements of the Arrowhead Management Copilot [6]

This Management Copilot has been deployed as a private-preview application in the BME testbed. The application is also made available as an open-source Docker compose for self-hosting on Github [8].

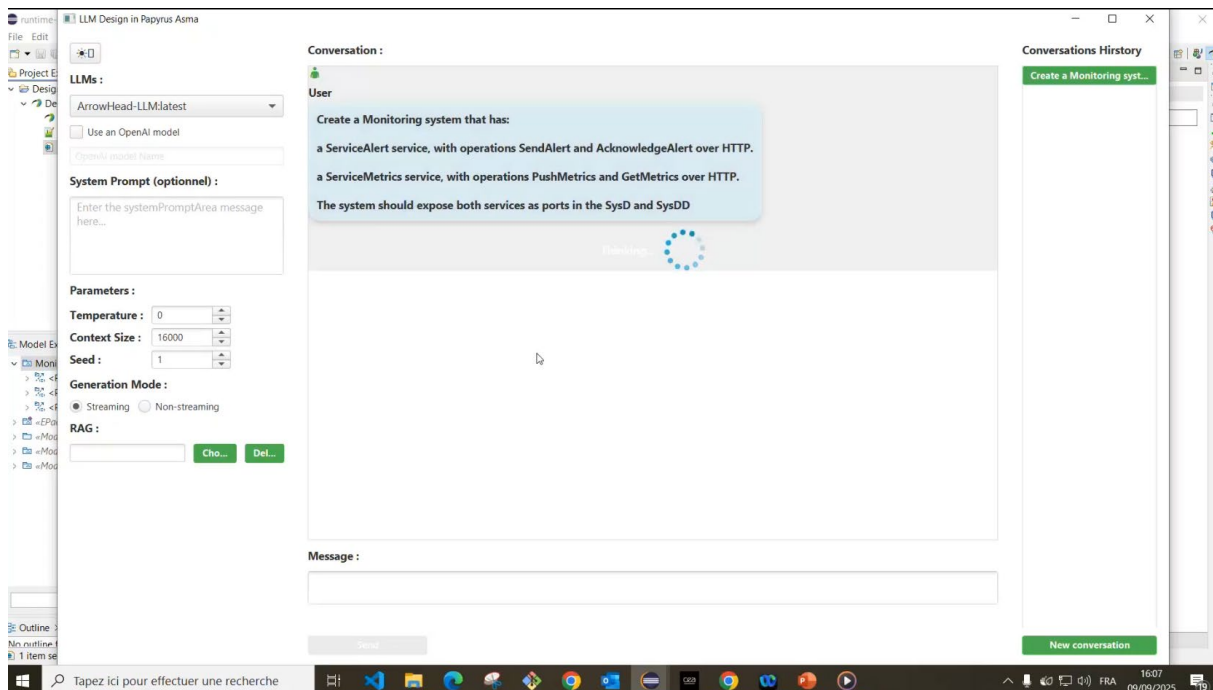


Figure 6.5 – The main elements of the Arrowhead Design Copilot [9]

Regarding the Design Copilot, a demonstration has been recorded on Eclipse Arrowhead Youtube [9].

6. Collaboration with other technical WPs

WP2 maintained its regular cross-work package coordination meetings with WP3 and WP4 to continue covering the integration-related requirements for T3.1-T3.3 and T4.1-T4.3. These joint efforts feed directly into the microservice-based Eclipse Arrowhead reference architecture. Covering the consolidated requirements focused on:

- ensuring syntactic, semantic, and pragmatic interoperability across systems;
- supporting interoperability across major industrial standards;
- creating and integrating the Arrowhead DSL;
- enabling seamless integration of AI-based translator modules; and supporting model-based translator modules as well.

Based on the outcomes of the initial T3.1 survey, participants reported more than 27 major industrial data models and over 10 data representation formats. In response, the Arrowhead fPVN work packages have continued developing and aligning translator capabilities for these representations. The identified formats can be grouped into three categories:

- (1) XML and JSON formats, including predefined or standardized schemas;
- (2) EXPRESS language-based models; and
- (3) model-based representations (e.g., UML).

Furthermore, the TranslationBridge concept came into life, as a reference implementation.

As a result of the WP2-WP3-WP4 collaboration, the Eclipse Arrowhead core systems are being designed to support the following capabilities:

- schema-driven translation subsystems where schemas can be uploaded by consumers and providers and managed as meta-information by AH components;
- schema-aware orchestration; automatic detection of interoperability gaps with corresponding translator instantiation and composition; and
- the extension of orchestration parameters with context-specific metadata to better address pragmatic interoperability.

7. Conclusion

This is the third deliverable – D2.3 – of the WP2 “Microservices” work package of the Arrowhead fPVN project.

During the reporting period, the Eclipse Arrowhead framework was further matured through continued specification work and refinement of the Core and Support Systems. The partners produced detailed system descriptors and service descriptions for several key components, including the DynamicServiceOrchestration and ConsumerAuthorization Core Systems, the Authentication System, and the Blacklist Support System. A major step was the specification of the TranslationManager Support System, covering its services as well as reference descriptors and guidance for integrating interface translators and data-model translator application systems for use by the pilots. In parallel, the DeviceQoS evaluator Support System was specified at a level suitable to directly drive implementation, and the documentation set was expanded with updated interface design descriptions for HTTP/MQTT and practical configuration guidance. The period also included dissemination and validation activities at the Deep Tech workshop in Lisbon (October 2025), where the Arrowhead v5.1.0 feature set and the framework’s translation integration approach were presented and discussed with a broader audience.

On the implementation side, multiple development phases delivered concrete reference systems and release milestones. Phase 3 completed the DynamicServiceOrchestration, ConsumerAuthorization, Authentication, and Blacklist systems and, together with the earlier ServiceRegistry, formed the Eclipse Arrowhead v5.0.0 release approved through the Eclipse Foundation review process. Phase 4 introduced the TranslationManager Support System and integrated it with DynamicServiceOrchestration, resulting in the v5.1.0 release, alongside reference translator implementations: an interface translator bridging MQTT-to-HTTP and a data-model translator adapter capable of exposing Python-based translation algorithms. Phase 5 added the SimpleStoreServiceOrchestration Core System and the DeviceQoS evaluator Support System, with the latter also integrated into DynamicServiceOrchestration, culminating in the v5.2.0 release. Together, these steps significantly strengthen Arrowhead’s capability for dynamic orchestration, governance, QoS-aware operation, and runtime interoperability through translator-based bridging.

As an originally unplanned result, WP2 conceptualized and further developed 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 not only the basic concepts – but the actual working modules for the identification of non-interoperable properties, Translators, as well as the QoS monitoring concept elements.

8. Appendixes

Besides this “Deliverable D2.3 Main document)” the Roadmap discussions and results can be found on github:

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

9. References

- [1] <https://github.com/Aitia-IIOT/ah5-common-java-spring>
- [2] <https://github.com/Aitia-IIOT/ah5-core-java-spring>
- [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," In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, Seoul, Korea, Republic of, 2024, pp. 1-6, doi: 10.1109/NOMS59830.2024.10575845.
- [6] Cs. Hegedűs, Á. Tóth, M. Bancsics, and P. Varga, "Multi-Agentive Plan-and-Solve Engine for Managing System of Systems via Natural Language,". In *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, Honolulu, Hawaii, USA, 2025 pp. 1-6, doi: 10.1109/NOMS57970.2025.11073647.
- [7] Arrowhead Copilot: AI-Driven Innovation for System Engineering video on project Youtube: <https://www.youtube.com/watch?v=JuTLenwYkG8&t=907s>
- [8] Arrowhead Management Copilot code repository <https://github.com/arrowheadcopilot/planner-onprem>
- [9] Demonstration of Arrowhead Design Copilot. Eclipse Arrowhead Youtube. https://www.youtube.com/watch?v=F8MHTjm_btg

10. Revision history

10.1 Contributing and reviewing partners

Contributions	Reviews	Participants	Representing partner
Architecture Definition			AITIA
Architecture Definition			BME
Architecture Definition			SINETIQ
Architecture Definition			LTU
Use Case Integration			TAU
Architecture discussion and review			EUROTECH
Architecture discussion and review			BEIA
Architecture discussion and review			LEONARDO

10.2 Amendments

No.	Date	Version	Subject of Amendments	Author
1	27/11/2025	0.1	Initial version	Tamas Bordi, Pal Varga
2	12/12/2025	0.2	Task descriptions and references added	Pal Varga
3	23/12/2025	0.3	Use case results added	David Hastbacka
4	9/01/2026	0.4	Use Case results integrated	Pal Varga, Miren Illarramendi
5	18/01/2026	0.5	Further use cases added, and editing task results	Pal Varga, David Rutqvist
6	19/01/2026	0.6	Arrowhead Copilot update integration	Csaba Hegedűs
7	20/01/2026	0.7	Final editing and release for review	Pal Varga
8	26/02/2026	1.0	Added WG charter update plan and UC2.6 finalization	Pal Varga

10.3 Quality assurance

No	Date	Version	Approved by
1	18/01/2026	0.5	Jerker Delsing
2	19/01/2026	0.7	Jerker Delsing
3	26/02/2026	1.0	Jerker Delsing