

Deliverable D4.3

WP4: Autonomous Translation Technology

Work package leaders:

Carl Borngund
carl.borngund@ltu.se

Filipe Moutinho
fcm@fct.unl.pt

Abstract

This document constitutes Deliverable D4.3 of the Arrowhead fPVN project. This document outlines the Autonomous Translation Technology, built to meet the specific needs detailed in the provided use cases.

Grant Agreement No.	101111977
Project Acronym	Arrowhead fPVN
Project Full Title	Arrowhead flexible Production Value Network
Dissemination Level	PU
Date of Delivery	November 2025
Deliverable Number	4.3
Deliverable Name	Autonomous Translation Technology
AL / Task Related	Tasks 4.1, 4.2, 4.3, and 4.4
Author(s)	Carl Borngrund and Filipe Moutinho
Contributors	The Arrowhead fPVN WP4 Partners
Reviewer	Jerker Delsing
Keywords	Translation, Interoperability, Ontology, Artificial Intelligence, Modelling
Abstract	This document constitutes Deliverable D4.3 of the Arrowhead fPVN project. This document outlines the Autonomous Translation Technology, built to meet the specific needs detailed in the provided use cases.

Technological Summary

The goal of Deliverable 4.3 is to highlight the many technical solutions that have been developed throughout the Arrowhead fPVN project. To help the reader to navigate Deliverable 4.3, **this section aims to give a short introduction to each of the proposed solutions, highlight the strengths of the proposed solutions, highlight the entity that has developed the solution, and estimate the current TRL of the proposed solutions.**

It is important to note that this section only highlights proposals and prototypes (defined in Section 4) and omits support activities, as it does not adhere to the goal of highlighting the actual technological solutions. Please refer to the specific Section 4.1, Section 4.2, and Section 4.3 for information regarding support activities.

Ontology-Based Translation

The ontology-based translation proposals and prototypes are discussed below:

Data Interoperability Translators Automatic Generator: DITAG is an ontology-based translation solution that automatically generates executable data translators from semantically annotated XML or JSON schemas. DITAG enables interoperability between service providers and consumers that use different data models by verifying semantic and structural compatibility. DITAG's main strength is zero-configuration integration, where schema evolution can be handled by updating semantic descriptions without modifying consumer code. Developed by: UNINOVA. Estimated TRL: 5. Full Description: Section 4.1.1.

DITAG Manager: The DITAG Manager provides orchestration and lifecycle management for ontology-based translators generated by DITAG. The DITAG Manager supports the discovery, configuration, and deployment of translators within the Arrowhead framework. The DITAG Managers' primary strength is enabling runtime management and operational use of ontology-based translation across system of systems paradigms. Developed by: UNINOVA. Estimated TRL: 4. Full Description: Section 4.1.2.

Arrowhead Framework Ontology: The Arrowhead Framework Ontology defines a formal semantic model for describing services, data, and interoperability concepts within the Arrowhead ecosystem. AFO supports consistent semantic annotations and enables reasoning, traceability, and verification across translation solutions. AFO's key strength is serving as a shared semantic backbone for heterogeneous translators. Developed by: LTU. Estimated TRL: 4. Full Description: Section 4.1.3.

PolyGIIoT: PolyGIIoT is an ontology-based interoperability approach for heterogeneous industrial IoT environments. PolyGIIoT focuses on harmonising data across platforms using shared semantic models and ontology-driven mappings. PolyGIIoT's main strength is enabling cross-domain, vendor-neutral data interoperability. Developed by: ISEP. Estimated TRL: 4. Full Description: Section 4.1.4.

HAScO/Repo: HAScO/Repo is a repository used to describe, align, and trace scientific and industrial data through shared semantics. Within translation workflows, HAScO/Repo enables automatic derivation of translation logic and preserves the semantic provenance of harmonised data. HAScO/Repo key strength is supporting auditable, provenance-aware interoperability. Developed by: Piaget. Estimated TRL: 5. Full Description: Section 4.1.5.

Ontology-Based Translation in Business Process: This proposal investigates the integration of ontology-based translation into executable business processes. It identifies translation needs using super-ontology concepts and embeds translation steps into process choreographies. Its strength lies in connecting semantic translation with process automation in long-running industrial workflows. Developed by: TAU. Estimated TRL: 3. Full Description: Section 4.1.7.

Arrowhead fPVN Ontology-Driven Interpretation: This solution explores ontology-driven interpretation of industrial data within the Arrowhead fPVN context. It enables consistent semantic reasoning across systems and supports lifecycle traceability and verification. Its strength lies in grounding data interpretation and translation in formal ontologies. Developed by: BEIA. Estimated TRL: 4. Full Description: Section 4.1.8.

AI-Based Translation

The AI-based translation proposals and prototypes are discussed below:

AI-Based Translation System: The AI-Based Translation System applies machine learning techniques to automate data model mapping based on semantic similarity. The system learns relationships between data elements using vector embeddings and similarity search. This AI-based translation system's main strength is reducing manual engineering effort when translating loosely aligned or weakly documented data models. Developed by: AITIA & BME. Estimated TRL: 4. Full Description: Section 4.2.4.

AI-Based Translation Integrated in the Arrowhead Framework: This solution integrates AI-based translation capabilities into the Arrowhead framework. It enables learning-based translators to be deployed, discovered, and orchestrated as Arrowhead services. Its strength lies in combining AI-driven translation with a service-oriented architecture. Developed by: AITIA & BME. Estimated TRL: 5. Full Description: Section 4.2.5.

ML/AI Translation for Environmental Sensor Data Streams: This proposal applies ML-based translation techniques to heterogeneous environmental sensor data streams. It focuses on learning mappings between evolving sensor schemas in IoT environments. Its strength lies in scalability and adaptability to dynamic data sources. Developed by: BEIA. Estimated TRL: 4. Full Description: Section 4.2.11.

Percepta: Percepta explores AI-driven interpretation and translation of complex industrial data. Percepta combines semantic similarity analysis with learning-based reasoning to support translation workflows. Percepta's main strength is addressing ambiguity and variability in industrial data representations. Developed by: ISEP. Estimated TRL: 4. Full Description: Section 4.2.12.

Model-Based Translation

The model-based translation proposals and prototypes are discussed below:

Arrowhead v5.1 Translation Support System: This solution provides model-based translation support integrated with Arrowhead v5.1. It enables structured mappings between system models and runtime data exchanges. Its strength lies in maintaining consistency between system design models and operational data. Developed by: LTU. Estimated TRL: 4. Full Description: Section 4.3.1.

Object Detection in Engineering Diagrams with Scarce Training Data: This proposal investigates AI-assisted interpretation of engineering diagrams to support model-based translation. It focuses on extracting structured information from legacy graphical artefacts with limited training data. Its strength lies in enabling translation inputs where formal data models are unavailable. Developed by: LTU & CEA. Estimated TRL: 4. Full Description: Section 4.3.3.

Arrowhead DSL (SysMLv1) to Arrowhead SysML v2 Library: This solution translates legacy SysMLv1-based Arrowhead DSL models into SysML v2 representations. It supports migration to modern MBSE workflows while preserving system semantics. Its main strength is ensuring continuity across modelling generations. Developed by: CEA. Estimated TRL: 4. Full Description: Section 4.3.5.

Using LLM to Generate Arrowhead DSL Models: This solution uses large language models to generate Arrowhead DSL models from natural language descriptions. It lowers the barrier to system modelling and accelerates early design activities. Its strength lies in improving accessibility and productivity in model creation. Developed by: CEA & AITIA. Estimated TRL: 4. Full Description: Section 4.3.6.

Rely to SysML: This solution translates requirements models from the Rely tool into SysML. It enables integration between requirements engineering and system modelling environments. Its strength lies in supporting traceable and model-driven validation workflows. Developed by: CEA & SAE. Estimated TRL: 5. Full Description: Section 4.3.7.

Translating Industrial Data into System Models with SysMLv2: This solution focuses on translating industrial data into SysML v2 system models using formal semantics. It supports interoperability, verification, and lifecycle traceability within MBSE ecosystems. Its strength lies in connecting operational data with system-level models. Developed by: IncQuery Labs. Estimated TRL: 3. Full Description: Section 4.3.9.

UML/SysML-driven Translation for IoT Environmental Data: This proposal applies UML and SysML abstractions to model environmental sensing systems and translate data streams into standardised representations. It maintains consistency between physical sensing, data exchange, and system models. Its strength lies in model-level coherence across domains. Developed by: BEIA. Estimated TRL: 3. Full Description: Section 4.3.11.

Aerobridge Translator: Aerobridge is a model-based translation solution for converting aerospace data to and from S5000F-compliant XML. Aerobridge supports standardised data exchange between suppliers and customers and includes schema and business-rule validation. Aerobridge's strength lies in enforcing standard compliance in industrial data exchange. Developed by: LDO. Estimated TRL: 5. Full Description: Section 4.3.12.

Contents

1	Introduction	10
2	Summary of WP4 Aims and Tasks	10
3	WP4 Objectives	11
3.1	Major Project Objectives	11
3.2	WP Objectives	11
4	Autonomous Translation Technology	13
4.1	Ontology–Based Translation	14
4.1.1	Data Interoperability Translators Automatic Generator (DITAG) (Prototype)	14
4.1.2	DITAG Manager (Prototype)	17
4.1.3	Arrowhead Framework Ontology (Prototype)	26
4.1.4	PolyGilloT (Prototype)	30
4.1.5	HAScO/Repo Support for Arrowhead’s Semantic Services (Prototype)	31
4.1.6	HAScO/Repo: Battery Testing Datasets as Foundation for Demonstrating Eclipse Arrowhead Semantic Integration (Prototype)	41
4.1.7	Ontology-based Translation in Business Processes (Proposal)	49
4.1.8	Arrowhead fPVN Ontology–Driven Interpretation (Prototype)	51
4.2	AI–Based Translation	52
4.2.1	AI Agents (Support Activity)	52
4.2.2	MCP – Model Context Protocol (Support Activity)	53
4.2.3	Prompting Techniques (Support Activity)	54
4.2.4	AI-based Translation System (Prototype)	55
4.2.5	Integration with the Arrowhead 5.0 Microservice Paradigm (Prototype)	57
4.2.6	Measurable Progress of the AI-based Translation System (Support Activity)	57
4.2.7	AI-translation Evaluation Pipeline (Support Activity)	60
4.2.8	Quantitative Evaluation of a Semantic RAG-Driven XML Translation (Support Activity)	62
4.2.9	Next Steps - AI-based Translation	68
4.2.10	Dissemination	68
4.2.11	ML/AI Translation for Environmental Sensor Data Streams (Prototype)	68
4.2.12	Percepta (Prototype)	69
4.3	Model–Based Translation	71
4.3.1	Arrowhead v5.1 Translation Support System (Prototype)	71
4.3.2	SysML-based Translation - Towards a Model Governance Approach (Support Activity)	71
4.3.3	Object Detection in Engineering Diagrams with Scarce Training Data (Prototype)	72
4.3.4	Payload Recognition and Mapping (Support Activity)	73
4.3.5	Arrowhead DSL (SysMLv1) to Arrowhead SysML v2 Library (Prototype)	74
4.3.6	Using LLM to Generate Arrowhead DSL (SysMLv1) Models (Prototype)	75
4.3.7	Rely to SysML (Prototype)	76
4.3.8	Exploring Deep-learning-based Meta-modelling for Data Translation (Support Activity)	79
4.3.9	Translating Industrial Data Into System Models with SysML v2 (Proposal)	88
4.3.10	Data Representation and Translation with Emerging Paradigms (Support Activity)	89
4.3.11	UML/SysML-driven Translation for IoT Environmental Data (Proposal)	90
4.3.12	Aerobridge Translator (Prototype)	90
4.4	Validation and Verification Processes	91
4.4.1	V&V Process for the Ontology-based Translators	92
4.4.2	V&V Process for the AI-based Translators	92
4.4.3	V&V Process for the Model-based Translators	93

5	Use Cases Collaboration and Support Activity	93
5.1	UC 1_6	93
5.1.1	Main Challenge	94
5.1.2	Approach	94
5.1.3	Validation in the Use Case	95
5.1.4	Conclusion	95
5.2	UC 1_7	95
5.2.1	DiTAG IPC-2581 to Electronic Component Data Model	95
5.2.2	Electronic Component Data Model to IPC 2581 translation	96
5.2.3	ISO 10303 translation	98
5.2.4	Rely to SysML	98
5.3	UC 2_9	100
5.4	V&V Process Adoption in the UCs	101
5.4.1	UC 1_6	101
5.4.2	UC 1_7	102
5.4.3	UC 2_7	104
6	Applying Local LLMs	113
7	Cross WP Collaboration	113
7.1	Integration of BEIA's Environmental Monitoring Datasets	114
7.1.1	Dataset Flow (as described in D4.2)	114
8	Appendices	114
9	Conclusions	115
10	References	115
11	Revision History	117
11.1	Contributing and Reviewing Partners	117
11.2	Amendments	118
11.3	Quality Assurance	118

Abbreviations

Abbreviation	
A3ST	Annotating Schemas to Support Semantic Translations
AfPVN	Arrowhead Future Production Value Network
AI	Artificial Intelligence
AML	Alignment Mapping Language
BPMN	Business Process Model and Notation
CAE	Computer-aided Engineering
CoT	Chain-of-Thought
CSV	Comma-Separated Values
D4.3	Deliverable 4.3
DEXPI	Data Exchange in the Process Industry
DITAG	Data Interoperability Translators Automatic Generator
DSL	Domain-specific Language
EA	Enterprise Architect
GraphRAG	Graph Retrieval-Augmented Generation
HAScO	Human-Aware Science Ontology
HTML	HyperText Markup Language
IDO	Industrial Data Ontology
INR	Implicit Neural Representations
INS	Instrument Semantic Template
ITS	Intelligent Transportation Systems
KGs	Knowledge Graphs
LLM	Large Language Model
mAP	Mean Average Precision
mAR	Mean Average Recall
MBSE	Model-Based Systems Engineering
MCP	Model Context Protocol
ML	Machine Learning
NLP	Natural Language Processing
OL	Ontology Learning
OWL	Web Ontology Language
P&IDs	Piping and Instrumentation Diagrams
PROV-O	Provenance Ontology
QoS	Quality of Service
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema

Abbreviations (Cont)

Abbreviation	
RSU	Roadside Units
RX	Reception
SAWSDL	Semantic Annotations for WSDL and XML Schema
SDD	Study Data Dictionary
SIO	Semanticscience Integrated Ontology
SoA	Service-oriented Architecture
SoC	State of Charge
SPARQL	SPARQL Protocol and RDF Query Language
SysML	Systems Modelling Language
TX	Transmission
UML	Unified Modelling Language
URI	Uniform Resource Identifier
V&V	Verification and Validation
VSTOI	Instrument Hierarchy Extensions
WP	Work package
XSD	XML Schema
XML	Extensible Markup Language

1 Introduction

Work Package (WP) 4 purpose is to research, design, develop, and deploy autonomous translation solutions that support translation between designated data models. The collaboration between relevant partners showcases their work in terms of ontology-based, AI-based, and model-based approaches. This is the final deliverable (D4.3) of WP4 and presents the work that has been done internally within the WP and the collaboration with the use-cases over the past 30 months.

The rest of the document is organised as follows: Section 2 gives a summary of the high-level aims of WP4, together with an outline of the individual purpose of each task within the WP. Section 3 outlines the contribution of WP4 in terms of the major project objectives and WP objectives as described in the Arrowhead fPVN project. Section 4 is the main section of this work that describes all the technological outcomes of WP4 throughout the project. Section 5 highlights the collaboration between the use-cases within the AfPVN project and WP4. Section 6 describes the considerations when applying local large-language models. Section 7 presents the cross-collaboration of WP4 in terms of internal collaborations to achieve the project's common objectives. Section 8 includes a list of relevant appendices. Section 9 concludes this document.

2 Summary of WP4 Aims and Tasks

WP4 aims to research, design, develop, and deploy translation services, facilitating data model translation across prominent data models and standards. This is to address the grand challenge of: *"Transforming labour-intensive, application-specific, and inflexible solutions into autonomous machine-to-machine information interoperability for production value networks information exchange and sharing"*. In order to address this grand challenge, WP4 consists of four tasks:

- Task 4.1 Super-ontology Based Data Model Translation
- Task 4.2 ML/AI Automated Data Model Translation
- Task 4.3 Model-Based Translation
- Task 4.4 Datasets and Translation Quality Assessment

The high-level, long-term project goal of each task is the following:

Task 4.1: Task 4.1 will investigate the usage of ontologies/super-ontologies as an intermediate between data sources of different data modelling languages to facilitate autonomous translation. The focus of Task 4.1 is to leverage ontology-based translation tools to facilitate translation between microservices relevant to the current workflow management. Key actions include identifying relevant ontologies, applying expertise in workload languages for seamless translation, and exploring machine learning techniques to address mapping challenges.

Task 4.2: Task 4.2 will investigate how Artificial Intelligence/Machine learning (AI/ML) can be used for translation between major data modelling languages. These will leverage real-world data from a diverse number of use cases in order to examine the capability of AI/ML methods for the translation of related data modelling languages through the creation of datasets. Task 4.2 will also investigate the possibility of introducing a translation microservice based on the architecture and libraries from WP2, employing Natural Language Processing (NLP), Graph Neural Networks, and dictionary learning.

Task 4.3: Task 4.3 will investigate model-based approaches, such as Systems Modelling Language (SysML) and Unified Modelling Language (UML), to examine their capability and feasibility of providing autonomous translation between major data modelling languages. Proposed translation solutions will be tested with real-world data from a variety of use cases to identify challenges and verify their effectiveness. Activities of task 4.3 include the investigation of UML meta models, identifying similarities between modelling languages using formal semantics or learning methods. Furthermore, Task 4.3 will examine the feasibility of provisioning translation microservices based on the architecture and libraries provided by WP2.

Task 4.4: Task 4.4 will provide essential datasets suitable for training, development, and early validation of translation quality in order to ensure the quality of the project's lifecycle. Task 4.4 will collaborate closely with the use cases, other WPs, and tasks within WP4 to ensure a comprehensive dataset procurement that supports rigorous evaluation of translation outcomes and is aligned with the project's objectives and quality benchmarks.

3 WP4 Objectives

The primary objective of WP4 is to provide translation service technology enabling data model translation between major data modelling languages, such as ISO 10303, ISO 15926, IEC 81346, and S5000F. Furthermore, WP4 has been divided into the following set of sub-objectives:

- Investigating the capabilities and feasibility of a super ontology approach
- Investigating the capabilities and feasibility of an ML/AI-based approach
- Investigating the capabilities and feasibility of a model-based approach
- Provision of dataset for translation development and early assessment of translation quality
- In cooperation with WP3, to provide translation microservices based on the above approaches

3.1 Major Project Objectives

Table 1 shows the relevant project objectives which are addressed by this deliverable and how they are supported.

Table 1: Project Objectives Addressed in the Document.

Objective	Contribution
#1. Facilitate more than 50% of needed translations in realistic production value networks by autonomous machine-based translation micro-services, thus significantly reducing the need for human support.	This document outlines the autonomous translation technology that has been developed in WP4 within the AfPVN project during the first 30 months in order to achieve the targeted translation percentage. This is based on use case collaboration in order to integrate translation seamlessly into production value networks.

3.2 WP Objectives

Table 2 shows the relevant WP objectives which are addressed by this deliverable and how they are supported.

Table 2: WP Objectives Addressed in the Document.

Objective	Contribution
Investigating the capabilities and feasibility of a super ontology approach.	This document outlines the work conducted on the usage of ontology-based approaches for translation, as detailed in Section 4.1. These technical solutions include, but are not limited to, Data Interoperability Translators, Automatic Generator, Arrowhead Framework Ontology, and HAScO/Repo.
Investigating the capabilities and feasibility of an ML/AI-based approach.	This document outlines the work conducted on the usage of AI/ML-based approaches for translation, as detailed in Section 4.2. These technical solutions include, but are not limited to, AI-based translation system architecture and integration into the Arrowhead Framework v5.1.
Investigating the capabilities and feasibility of a model-based approach.	This document outlines the work conducted on the usage of model-based approaches for translation, as detailed in Section 4.3. These technical solutions include, but are not limited to, the Arrowhead v5.1 Translation support system, LLMs to generate Arrowhead DLS models, and SysMLv1 to SysMLv2 translation.
Provision of data sets for translation development and early assessment of translation quality.	This document outlines the work conducted on the verification and validation process of datasets for translation, as detailed in Section 4.4.
In cooperation with WP3, to provide translation microservices based on the above approaches	This document includes the collaboration with WP3 and WP2 to provide translation microservices, as outlined in Section 7.

4 Autonomous Translation Technology

Effective communication between computing systems depends on compatible interfaces and a shared understanding of data. This includes technical aspects such as communication protocols, encodings, encryption, and compression, as well as higher-level considerations, including message and payload structure, and semantics. Once data is exchanged, accurate interpretation is critical. However, mismatches in interfaces or data formats can limit the communication of data.

Translation techniques may serve as either a temporary or permanent solution to bridge incompatibilities in the interfaces or data understanding. These are particularly valuable in scenarios such as system failover, where an alternative system lacks compatibility, or in the integration of heterogeneous systems from different vendors or versions, where autonomous translation systems can reduce engineering overhead. Similarly, updates to legacy systems may be complicated due to the underlying hardware or software solutions, which makes translation an attractive alternative.

However, autonomous translation faces significant challenges, particularly in achieving deep semantic understanding across diverse data models and in handling multimodal data. Historically, interoperability has often relied on adherence to standardised exchange formats and the use of well-defined APIs, which enable more reliable communication across systems.

To address these challenges, three translation approaches are explored, following the same structure as the WP task partitioning:

- **Ontology-based Translation:** This approach uses ontologies, which are formal representations of knowledge, mapping concepts and relationships between different data models. By leveraging ontologies, semantic similarities and differences can be identified and autonomously translated, facilitating better interoperability between heterogeneous data sources.
- **AI-based Translation:** AI-based translation leverages machine learning techniques to automatically learn the underlying structure of different data models and how to translate them. These approaches typically involve training a machine learning model on a large, representative dataset to develop high-performing and accurate translation solutions that can handle the complex structures and semantics that often exist in data.
- **Model-based Translation:** Model-based translation involves translating data between different models by mapping elements and attributes from one model to another. This approach focuses on understanding the structure and semantics of each model and developing mappings that preserve the meaning and integrity of the data during translation.

In addition to validate and verify (V&V) the approaches defined above, to support the development of autonomous microservice-based translators, a set of concrete steps is proposed:

- **Validation and Verification Process:** These processes, which differ depending on approach, aim to outline the actions, methodologies, and tools needed for a rigorous V&V process to ensure efficient adoption of translators within the project use cases.

In this deliverable, like previous deliverables, the autonomous translation technologies are categorised as *Proposal*, *Prototype*, and *Support Activity* to capture their conceptual, developmental, and analytical phases.

The categories are defined as follows:

- **Prototype (TRL 3+):** Tangible prototypes that can be tested and used (code and some documentation available).
- **Proposal (TRL 2):** Initial conceptualisation and proposition of translation solutions, outlining the theoretical framework and methodology.
- **Support Activity:** Pre-studies, partial investigations, and analyses aimed at enhancing the domain's state-of-the-art knowledge.

This framework highlights the project's tangible progress in technological readiness. It also provides a clear structure for assessing the maturity of each developed translator and its contribution toward operational deployment.

4.1 Ontology-Based Translation

The following section outlines the efforts and progress regarding the technical solutions at the 30-month mark of the project concerning ontology-based translations. These efforts include the analysis of requirements, design, development, validation, and integrated translation microservices of the proposed solutions.

4.1.1 Data Interoperability Translators Automatic Generator (DITAG) (Prototype)

4.1.1.1 DITAG Tool

During this project, the Data Interoperability Translators Automatic Generator (DITAG) tool was developed. This tool, which uses an ontology-based approach, receives metadata with semantic annotations, verifies semantic and data compatibility, and automatically generates translators when compatible. Specifically, DITAG automatically generates translators based on ontologies and Schemas (XML or JSON schemas) with semantic annotations and also performs message/document translation, as illustrated in Figure 1. These messages/-documents can have multiple origins, such as web service results or sensory telemetry information. The need for translation arises when a service provider uses a data model different from that of the consumer, making it unreadable for the consumer.

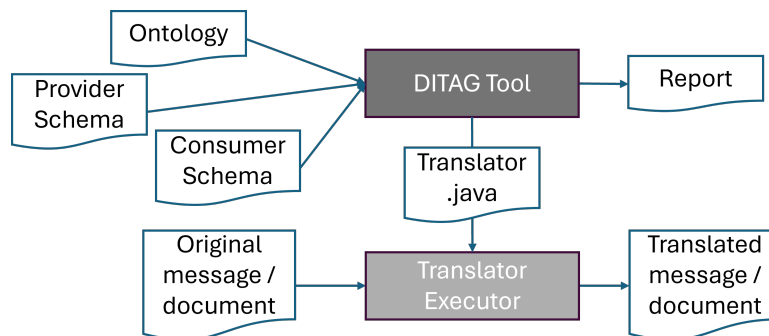


Figure 1: DITAG approach.

The elements presented in the Figure 1 are defined as follows: role:

- **Original message/document:** represents the set of messages or documents that the provider sends/-generates. The messages/documents' content format can be XML or JSON. Figure 2 presents an XML message of a provider.
- **Provider/Consumer Schema:** describes the structure of the provider/consumer messages or documents. These schemas must have semantic annotations for a shared/reference ontology. Figure 3 presents the XSD of the provider, which defines the XML sample from Figure 2, whereas Figure 4 presents the XSD of a consumer.
- **Ontology:** represents the ontology that is used in both provider and consumer Schemas.
- **Translated message/document:** represents the set of messages or documents (XML or JSON) that the consumer can read/understand (in compliance with Consumer Schema). These messages/documents are generated by using the translator automatically generated by DITAG. Figure 5 presents one of those examples.
- **Translator.java:** the translator generated by DITAG that supports message/document translation, as described in Section 4.1.2. Figure 6 shows a code excerpt from the generated translator.
- **Report:** report about providers/consumers compatibility.

During this period, DITAG was extended to support new annotation mechanisms and include bug fixes. As described in previous deliverables, DITAG is a Java application that automatically generates translators to support the interaction between application systems (*Providers* and *Consumers*) that send and receive data in XML and/or JSON. To do this, DITAG requires *Providers* and *Consumers* metadata, specifically XML Schemas

```

1 <tempReadings>
2   <tempIndoorValue>24</tempIndoorValue>
3   <tempIndoorUnits>celsius</tempIndoorUnits>
4   <tempOutdoorValue>16</tempOutdoorValue>
5   <tempOutdoorUnits>celsius</tempOutdoorUnits>
6 </tempReadings>
7

```

Figure 2: XML example from a provider.

```

1 <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:a3st="
2   http://gres.uninova.pt/a3st" xmlns:sawSDL="http://www.w3.org/ns/sawSDL">
3   <xs:element name="tempReadings">
4     <xs:complexType>
5       <xs:sequence>
6         <xs:element name="tempIndoorValue" type="xs:float" sawSDL:modelReference="/IndoorTemperatureSensor/hasDecValue" a3st:group="1" />
7         <xs:element name="tempIndoorUnits" type="xs:string" sawSDL:modelReference="/IndoorTemperatureSensor/hasTempUnits/temperatureUnits"
8           a3st:group="1" a3st:mdi-ref="1" />
9         <xs:element name="tempOutdoorValue" type="xs:float" sawSDL:modelReference="/OutdoorTemperatureSensor/hasDecValue" a3st:group="1" /
10        >
11        <xs:element name="tempOutdoorUnits" type="xs:string" sawSDL:modelReference="/OutdoorTemperatureSensor/hasTempUnits/
12          temperatureUnits" a3st:group="1" a3st:mdi-ref="1" />
13      </xs:sequence>
14    </xs:complexType>
15  </xs:element>
16  <xs:annotation>
17    <xs:appinfo>
18      <a3st:metadata a3st:ontology="http://gres.uninova.pt/tag/validation/json/TempSensorV2.owl">
19        <a3st:map-data-inds>
20          <a3st:map-data-ind a3st:mdi-id="1">
21            <a3st:map-value-ind a3st:individual="Fahrenheit" a3st:value="fahrenheit" />
22            <a3st:map-value-ind a3st:individual="Celsius" a3st:value="celsius" />
23          </a3st:map-data-ind>
24        </a3st:map-data-inds>
25        <a3st:data-property-values>
26          <a3st:data-property-value name="tempIndoorLat" type="xs:float" a3st:group="1" a3st:property="/IndoorTemperatureSensor/hasLocation
27            /Latitude[hasLocUnits/LocationUnits[{DD}]]/hasDecValue" a3st:value="38.66016"/>
28          <a3st:data-property-value name="tempIndoorLong" type="xs:float" a3st:group="1" a3st:property="/IndoorTemperatureSensor/
29            hasLocation/Longitude[hasLocUnits/LocationUnits[{DD}]]/hasDecValue" a3st:value="-9.20405"/>
30          <a3st:data-property-value name="tempOutdoorLat" type="xs:float" a3st:group="2" a3st:property="/OutdoorTemperatureSensor/
31            hasLocation/Latitude[hasLocUnits/LocationUnits[{DD}]]/hasDecValue" a3st:value="38.65984"/>
32          <a3st:data-property-value name="tempOutdoorLong" type="xs:float" a3st:group="2" a3st:property="/OutdoorTemperatureSensor/
33            hasLocation/Longitude[hasLocUnits/LocationUnits[{DD}]]/hasDecValue" a3st:value="-9.20326"/>
34        </a3st:data-property-values>
35      </a3st:metadata>
36    </xs:appinfo>
37  </xs:annotation>
38 </xs:schema>
39

```

Figure 3: XML Schema of the provider, corresponding to the XML example in Figure 2.

(XSDs) or JSON Schemas with semantic annotations. Semantic annotations provide additional information about the meaning of the elements and attributes defined in XSD documents, using Ontologies specified in Web Ontology Language (OWL) [1]. These semantic annotations are added to the XSDs using the SAWSDL (Semantic Annotations for WSDL and XML Schema) [2] extension attribute "modelReference" with annotation paths [3]. Group identifiers, which were previously included in semantic annotations, as proposed in [4], are now directly associated with nodes (XSD elements and attributes). Moreover, the Annotating Schemas to Support Semantic Translations (A3ST) extension [5] is also employed to supplement SAWSDL annotations. A3ST enables additional information to be provided about an element without altering the original structure of the XSD. The A3ST extension was updated to a more mature version (an example of A3ST annotations is presented in Figure 3, from line 14 to 27). DITAG also uses JSON Schemas with semantic annotations to support JSON data, as proposed in [6].

DITAG also generates a report in JSON format for external tools. The generated JSON report details the possible combinations/pairs between the elements collected from provider and consumer schemas, the best combination/match, and the number of obligatory matches and optional matches. This report can be analysed by external tools, facilitating DITAG integration, for instance, within the Arrowhead framework. The most important information of the JSON report is the data contained in "response" and in "match" (Figure 7). When the provider and consumer are not compatible, "match" is null. If they are compatible, the report shows the number of "obligatory-matches" and "optional-matches" that the provider supplies for the given consumer.

```

1 <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:a3st="
  http://gres.uninova.pt/a3st" xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
2 <xs:element name="data">
3 <xs:complexType>
4 <xs:sequence>
5 <xs:element name="temperature">
6 <xs:complexType>
7 <xs:attribute name="value" type="xs:float" use="required" sawsdl:modelReference="/TemperatureSensor/hasDecValue"/>
8 <xs:attribute name="units" type="xs:string" use="required" sawsdl:modelReference="/TemperatureSensor/hasTempUnits/
  TemperatureUnits" a3st:mdi-ref="1" />
9 </xs:complexType>
10 </xs:element>
11 <xs:element name="location">
12 <xs:complexType>
13 <xs:attribute name="latitude" type="xs:float" use="required" sawsdl:modelReference="/TemperatureSensor/hasLocation/Latitude[
  hasLocUnits/LocationUnits[{DD}]]/hasDecValue" />
14 <xs:attribute name="longitude" type="xs:float" use="required" sawsdl:modelReference="/TemperatureSensor/hasLocation/Longitude[
  hasLocUnits/LocationUnits[{DD}]]/hasDecValue" />
15 </xs:complexType>
16 </xs:element>
17 </xs:sequence>
18 </xs:complexType>
19 </xs:element>
20 <xs:annotation>
21 <xs:appinfo>
22 <a3st:metadata a3st:ontology="http://gres.uninova.pt/tag/validation/json/TempSensorV2.owl">
23 <a3st:map-data-inds>
24 <a3st:map-data-ind a3st:mdi-id="1">
25 <a3st:map-value-ind a3st:individual="Fahrenheit" a3st:value="F" />
26 <a3st:map-value-ind a3st:individual="Celsius" a3st:value="C" />
27 </a3st:map-data-ind>
28 </a3st:map-data-inds>
29 </a3st:metadata>
30 </xs:appinfo>
31 </xs:annotation>
32 </xs:schema>
33

```

Figure 4: XML Schema example from a consumer.

```

1 <data>
2 <temperature value="24" units="C"/>
3 <location latitude="38.66016" longitude="-9.20405" />
4 </data>
5

```

Figure 5: Consumer XML generated by DITAG, in compliance with the schema of Figure 4.

```

1 . . .
2 consumer = new Consumer();
3 consumer.temperature = new Consumer.Temperature();
4 consumer.temperature.setValue((float) (provider.getTempIndoorValue()));
5 consumer.temperature.setUnits(this.individualMap_data_temperature_units.get((provider.getTempIndoorUnits())));
6 consumer.location = new Consumer.Location();
7 consumer.location = new Consumer.Location();
8 consumer.location.setLatitude((float) (38.66016));
9 consumer.location.setLongitude((float) (-9.20405));
10 . . .
11

```

Figure 6: Excerpt of the Translator.java code generated by DITAG for the previous examples.

4.1.1.2 DITAG Tool Validation

To validate the DITAG tool, a set of test scenarios was defined, which are presented in Table 3. Each scenario aimed to evaluate the capability of checking compatibility and generating translators for different providers and consumers. To this end, an XSD was created and annotated for each provider and consumer, and an example provider XML message was used. Each scenario relates to a specific type of discrepancy between a provider and a consumer. With regard to structural differences in messages, for example, DITAG was tested to verify if it could handle variations such as data stored in attributes versus elements, or parent-child nesting structures. Each case was also tested in reverse; for instance, a scenario was tested in which the provider stores data in an element while the consumer expects it in an attribute.

```

1 {
2   "combinations": [ ... ],
3   "response": {
4     "code": 0,
5     "message": "Operation Completed Successfully"
6   },
7   "match": {
8     "obligatory-matches": [
9       "value - tempIndoorValue",
10      "units - tempIndoorUnits",
11      "latitude - tempIndoorLat",
12      "longitude - tempIndoorLong"
13    ],
14    "optional-matches": [],
15    "pairs": [ ... ]
16  },
17  "tryLog": [ ... ]
18 }
19

```

Figure 7: Excerpt of the Report.json generated by the DITAG Tool for the presented examples.

The "Generated" column indicates whether DITAG successfully generated the translator, with a tick confirming that the generation (or lack thereof) was as intended. For instance, if the provider contains fewer occurrences of an element than the consumer requires ($m < n$), generation is not possible, which is expected. Upon successful generation, each translator was executed to produce the consumer XML file, correctly mapping provider data to the expected structure, types, units, and semantic constraints on the consumer side. Overall, these tests provided a comprehensive assessment of DITAG's performance.

Table 3: Tests for the automatic generation of translators

Problem	Description		Generated
Structural differences	Provider stores data in an attribute	Consumer expects it in an element	yes ✓
	Provider has a parent with n children	Consumer expects the parent, within an ancestor	yes ✓
Cardinality mismatch	Provider element allows n occurrences	Consumer allows m occurrences ($m < n$)	yes ✓
	Provider element allows m occurrences	Consumer requires n occurrences ($m < n$)	no ✓
Diff. field names	Provider uses, e.g., <code>val</code>	Consumer expects <code>value</code>	yes ✓
Semantic differences	Provider sends data from a subclass	Consumer expects data from the class	yes ✓
	Provider sends data from a class	Consumer expects data from a subclass	no ✓
Grouping elements	Provider has 2 groups	Consumer expects 1 group	yes ✓
	Provider has 1 group	Consumer expects 2 groups	no ✓
Diff. data values	Provider uses, e.g., <code>m</code> and <code>%</code>	Consumer uses <code>metre</code> and <code>percentage</code>	yes ✓
Missing data	Provider doesn't send it	Consumer expects that data	no ✓
	Provider doesn't send it, but it's in the XSD	Consumer expects that data	yes ✓

4.1.2 DITAG Manager (Prototype)

Deliverable 4.2 described the process of integrating the DITAG translation tool into the Arrowhead Framework. This deliverable presents the prototype developed in the project, the DITAG-Manager (DM).

Integrating DITAG into Arrowhead enables service consumers to translate messages resulting from service provision into messages/documents whose structure and semantics they can understand. The strategy consists of using the DITAG tool as a library, integrated into a new service-based tool.

The DITAG-Manager follows the microservice paradigm and the ability to operate autonomously, as described in this chapter. This prototype allows DITAG to be integrated into an Arrowhead local Cloud. DM provides the functionality to automatically generate translators in real time as needed, resulting from the dynamics of service provision, with the respective exchange of messages within an Arrowhead Cloud. That is, when a consumer requests a service from a provider, if the consumer does not recognise the message format, the DM automatically translates it into a suitable, recognisable format.

In the next sections, we will describe DM functionality in a more abstract, macroscopic way, describing the most relevant aspects of the developed tool, namely the translation services layer and the processes and sequences of operations that the DM implements to enable message exchange between providers and consumers in a Cloud Arrowhead. In addition, we will present some examples of translation using DM. It shows

how to perform a translation using the service layer through the API. Lastly, we will demonstrate the concrete use of DM in an ecosystem supported by an Arrowhead Cloud.

4.1.2.1 How DITAG-Manager Works

The DM's operation is divided into two consecutive stages. The first stage consists of generating a translator for a given service. The second stage involves using the translator to translate the messages exchanged between a service provider and its respective consumer.

Figure 8 describes the translation process, which generally implements the DITAG usage method described above. The difference here is that interactions are performed via web services, as described below. In this preliminary phase, when a document needs to be translated, the DITAG Manager receives as input the service provider's XSD schema, the service consumer's XSD schema, and a reference ontology. As described in section 4.1.1.1, these documents are necessary to generate a translator.

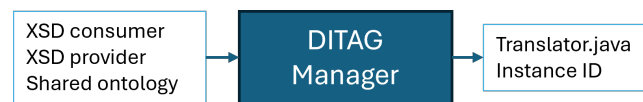


Figure 8: DITAG-Manager translation generation.

With these inputs, the DM uses the DITAG tool to generate the translator. The result of this process is a Java class, `Translator.java`, and an ID instance that allows this translator to be identified inside the DM and used later. The "instance ID" identifier then represents the translator generated from the schemas and ontology for a given service from an Arrowhead provider. After its creation, the translator must be stored and made available for future use. This phase must be performed only once per translator.

In the next phase, DM is effectively used to translate messages. These messages contain the information exchanged when a consumer requests a service from an Arrowhead provider. Figure 9 illustrates this process.

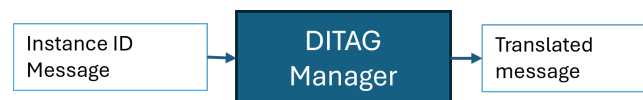


Figure 9: Using the generated translator.

The identifier mentioned above allows the DM to choose which generated translator to use for a translation. As shown below, if the content of any XSD document or ontology changes, the generated ID no longer corresponds to the existing translator, as each translator is associated with a unique ID generated using the Secure Hash Algorithm (SHA-2-256). When this happens, the DM reverts to the previous phase, where it regenerates a translator based on the new schemas and the ontology.

4.1.2.2 Development of DITAG-Manager

For the development of the DM, several requirements were considered, which are shown in Figure 10. The functional and non-functional requirements were established to enable the DM to provide translator generation and message translation using a microservices-based, autonomous approach.

Satisfying these requirements involves developing a platform with several components, as illustrated in Figure 11, which represents the conceptual architecture of the DM as proposed in deliverable 4.2.

As illustrated in the architecture, DITAG is the core of the translation manager. It was implemented based on Spring Boot. The "Ontology Manager" block consists of a small repository that houses ontologies and XSD schemas. The processes of generating translators and translating messages are accessible through web services provided by the "Rest Controller" block. Finally, a block for documenting services was also considered, namely via an Open API, represented here by the Swagger API block.

The internal structure of the most relevant blocks of DITAG-Manager is described in Figure 12, which presents the core components: the REST Controller and DITAG.

DITAG-Manager requirements	
Functional requirements	
FR 1	Handle requests for Translation creation
FR 2	Handle requests for translating documents
FR 3	Create Instances of each translator on the fly
FR 4	Provide for Storing and managing XSD definitions and Ontologies
FR 5	Decides whether translation is required or not on the fly
FR 6	Automatic management of Translation instances
FR 7	Provide web service access to the DM functions
Non-functional requirements	
NFR 1	Provide translation in real-time
NFR 2	Small footprint in memory and processing for each translation
NFR 3	Avoid the use of the filesystem during translations
NFR 4	Manager fully automatic, no human intervention for its operation

Figure 10: DITAG-Manager requirements.

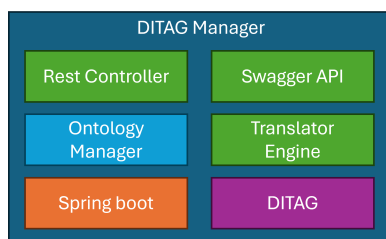


Figure 11: DITAG-Manager conceptual architecture.

DITAG Manager		Description
Rest Controller	createTranslate()	Receives the XSD schemas and an ontology. Invokes DITAG to create a translator. The resulting translator is identified by a unique ID. A codebase for the translator is created and accessed through the ID.
	translateMessage()	Receives the message and the translator's ID.
	findTranslator()	Query for the existence of a translator using metadata information, like XSD schemas and ontologies (work in progress).
	... other methods	
...	..	
DITAG	createTranslator()	Creates the translator.
	translateMessage()	Translates a message or document.
	...	
Translator instances	v1:04108f264f7883b049e036c...	Translator between provider 1 and consumer 1
	v1:60a1b62baf86c8f4903d049...	Translator between provider 1 and consumer 2
	...	...

Figure 12: Internal structure of DITAG-Manager

The goal was to develop DM so it could be integrated into an Arrowhead local cloud, facilitating service provisioning through automatic message translation. As such, the DM was integrated as an Arrowhead provider under the name DITAG-Provider. This provider runs on Arrowhead 5.0, the latest version.

Within the Cloud, the dynamics of interactions are illustrated in the sequence diagram in Figure 13, which shows the two main processes: the provision of a service with or without translation.

As illustrated in the diagram, a consumer and a provider register with the AF-SR (Service Registry) module. The provider supplies, for example, the "getTemperature" service. The consumer, in turn, needs to request the "getTemperature" service. To do this, the consumer prepares a request for this service through a module called "Semantic Gateway" (SG).

Based on the request's characteristics, the SG makes a direct request to a provider of the "getTemperature"

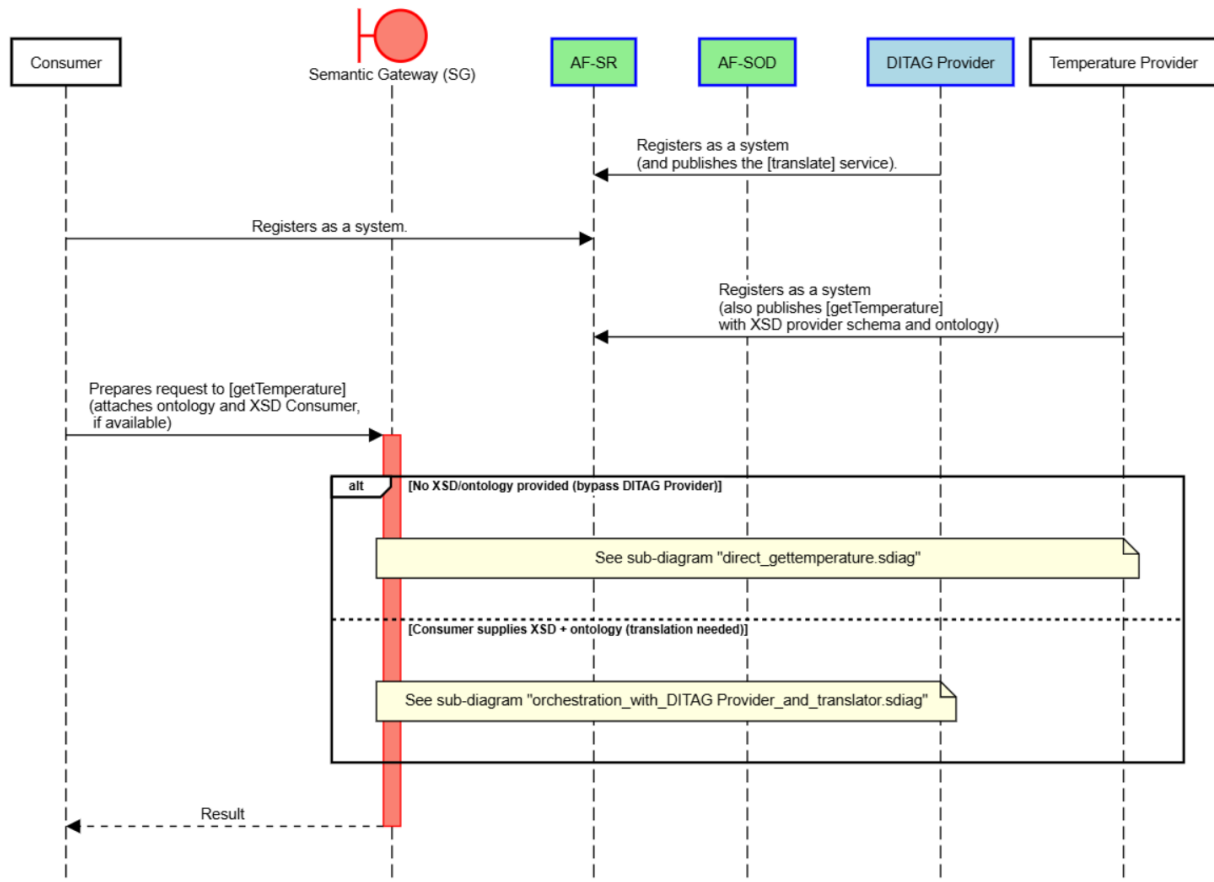


Figure 13: DITAG Provider integration in an Arrowhead Local Cloud.

service, without the need to translate the message or, if necessary, to search for a DITAG-Provider to translate it, as described below.

4.1.2.3 Provision of Service Without Translation

The sequence diagram shown in Figure 14 illustrates the process of requesting a service, in this case, "getTemperature". The SG examines the request's content, searching for metadata that contains semantic information for a possible translation of the message content resulting from the provision of a service. If this information does not exist, then SG prepares an orchestration for the service "getTemperature". Eventually, the orchestration results in the "Temperature Provider" yielding the mentioned service.

After the provider is selected and orchestrated, the SG module makes a direct request to the endpoint for the provider's "getTemperature" service. The result of this request is finally made available to the consumer, without translation.

4.1.2.4 Provision of Service With Translation

The sequence diagram shown in Figure 15 illustrates the "getTemperature" service request process, which, in this case, involves translation. As before, this process begins with the request being sent to the SG module. This time, the request includes semantic information indicating the need for message translation, resulting in a more complex interaction scheme in which the DITAG provider acts as an intermediary. That is, if the consumer's request to the SG module is represented by the tuple ("getTemperature", []), no translation is necessary; if the tuple is of the form ("getTemperature", [xsdConsumer, Ontology]), the SG determines that the service's content needs translation.

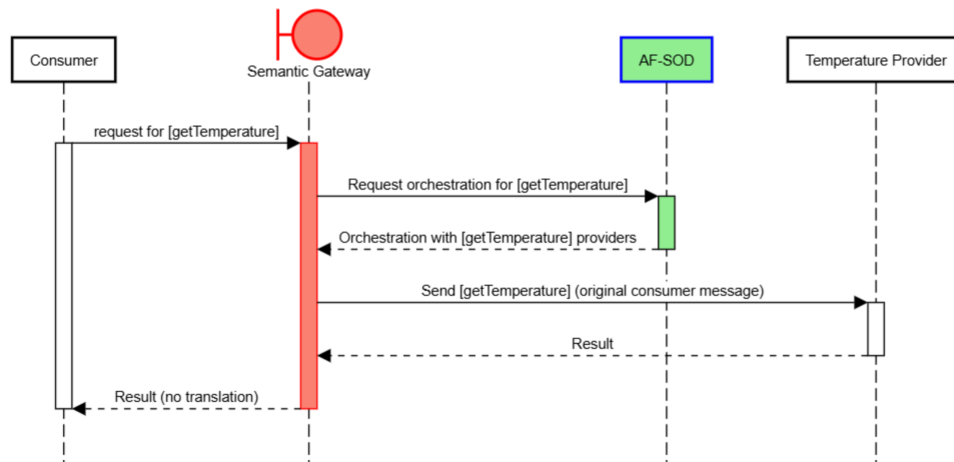


Figure 14: Direct "getTemperature" (no translation).

The [xsdConsumer, Ontology] list of the tuple mentioned above consists of the semantic information included in the request metadata. The XSD schema describes the structure of the message that the consumer accepts, and includes semantic annotations to the ontology, as explained in section 4.1.1.1. Since, in this case, the request metadata includes an XSD and an ontology, it is necessary to translate the message content resulting from the service provision.

As shown in the diagram above, the SG module sends an orchestration request to AF-SOD for a "translate" service provider. After obtaining a provider, the SG forwards the "getTemperature" request to the DITAG provider, sending the metadata mentioned above.

In turn, the DITAG provider receives the request, extracts the consumer schema and ontology. This information sends an orchestration request to the "getTemperature" service. Among the potential providers for this service, the DITAG provider, with the help of the DITAG tool, selects a compatible one. At that point, the DITAG provider makes the "getTemperature" service request to the chosen provider. After receiving the response message, two situations may occur.

The first time, no translator has been generated yet, so the DITAG-Provider uses the DITAG-manager library (which in turn uses the DITAG tool) to generate a translator and create a runtime instance of the translator. In this case, the DITAG-Manager translates the message and returns it as a response to the SG, which in turn returns it to the consumer.

The next time the "getTemperature" service is requested, an instantiated translator already exists, so the DITAG provider uses that instance to translate the message. On subsequent times, the translation process is very fast, less than 1ms, since the instance remains active in memory for a certain time-to-live (TTL) period. This approach provides the real-time performance defined in the requirements table presented above.

As mentioned above, translator instances are uniquely identified by a cryptographic identifier derived from the content of the XSD schemas and the ontology, serving as a fingerprint for the translator using the SHA-256 hash function. This means that when the content of the ontology or one of the schemas is modified, the generated ID will be different, and the existing instance will no longer correspond to the new identifier. At that point, the DITAG Manager creates a new translator associated with the new ID. The previous instance is eventually destroyed when its TTL expires without being used.

This last aspect provides the necessary autonomy for the DITAG-Manager to function independently without human intervention.

4.1.2.5 Testing the DITAG-Manager API

As mentioned earlier, this section will present some translation examples, in this case using the DITAG-Manager Open API, shown in Figure 16. The translation services provided by the DITAG library will be tested here. In the next section, the use of DITAG-Manager within an Arrowhead cloud will be tested.

The API's "createDinamicTranslator" service is used to generate a translator. Figure 17 indicates which fields must be filled in, namely, which in this case are the XSD schemas and the ontology.

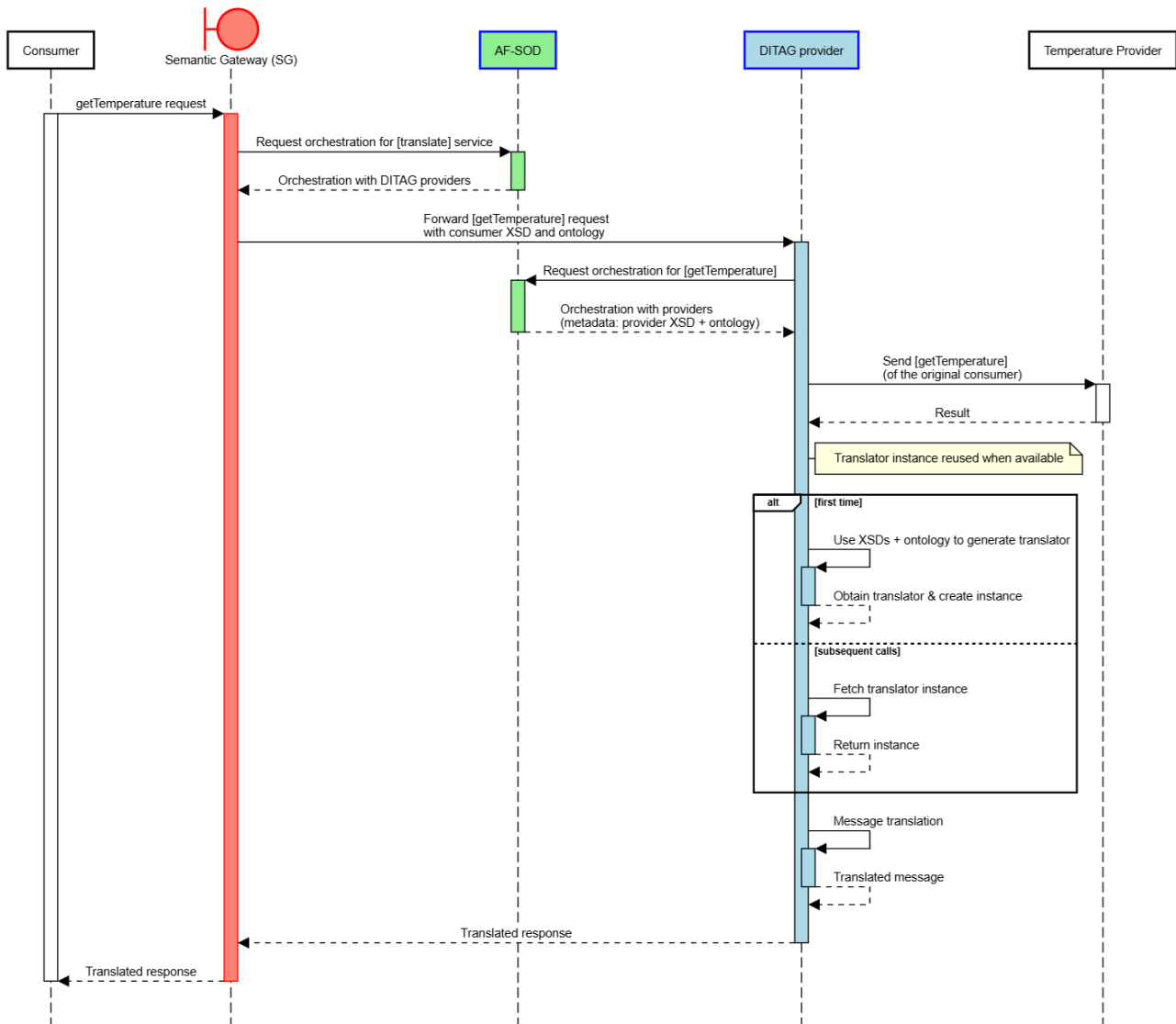


Figure 15: Orchestration via DITAG (incl. translator lifecycle).

The response to the translation service can be found in Figure 18, which shows the ID of the created translator instance and the source code of the translator. The source code is shown here for illustrative purposes only. In a context where DITAG-Manager is running in production mode, only the translator ID is returned.

Figure 19 shows an example of translating a message using the translator created earlier. This time, the inputs are the translator instance identifier and the message to be translated.

The result of this translation is illustrated in Figure 20. As shown, the message underwent structural and semantic modifications to ensure it could be understood by the service consumer.

Figure 21 shows the list of translators created using the API. The second translator is identical to the first. What happened was that the second time around, one of the schemas was slightly altered - in this case, the consumer schema.

The change consisted solely of inserting an extra space in the schema. However, because DITAG-Manager detects a different schema, it is forced to generate a new translator. As mentioned earlier, if only one of them is used, the other will eventually be destroyed at the end of a TTL period, freeing up memory and resources for new translators. Again, this mechanism enables DITAG-Manager to operate autonomously without requiring human intervention.

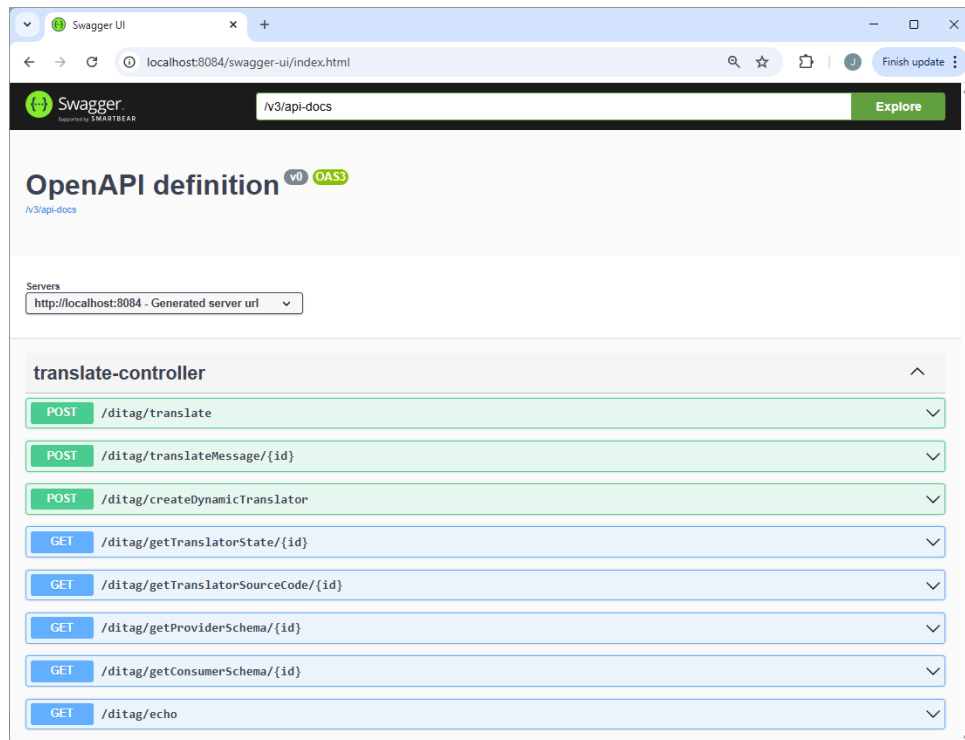


Figure 16: Open API Swagger with the DITAG-Manager services.

4.1.2.6 Testing the DITAG-Manager in an Arrowhead Cloud

This subsection presents the tests of the integration of DITAG-Manager running within a local Arrowhead cloud. As mentioned earlier, the strategy adopted was to integrate DM as an Arrowhead provider, which in this case provides translation services to other Arrowhead providers and consumers. The way the translation service provider works is illustrated in the sequence diagrams shown in figures 13 to 15, provided earlier.

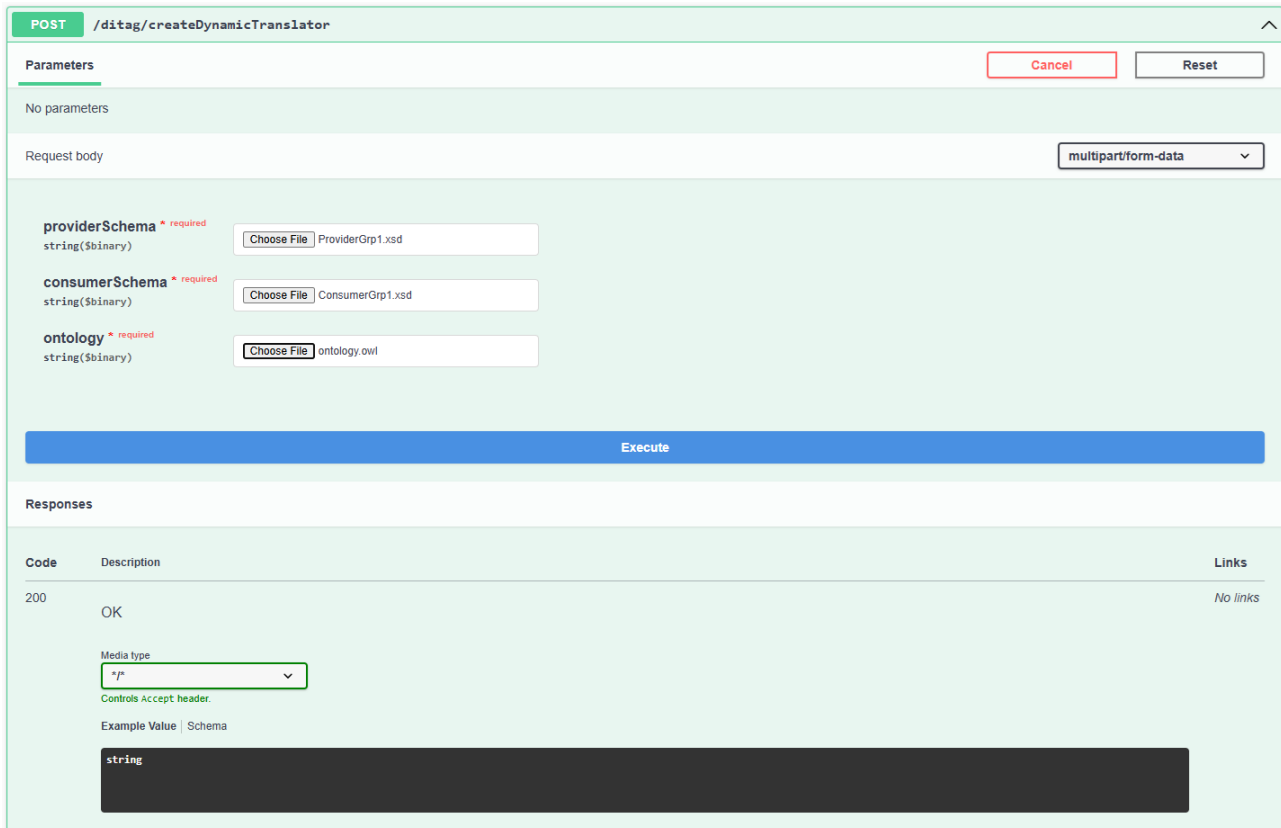
The following examples illustrate the two use cases described in the figures. The first example illustrates a service provision between a provider and a consumer that does not require translation. The second example illustrates the same service, this time with translation.

Figure 22 shows the output of the provider console as its services are requested; in this case, the "getTemperature" service. The structure of the provider's messages never changes.

Figure 23 shows the console output associated with an Arrowhead consumer. As can be seen, the example has a small menu of options, so that the consumer can choose between requesting the "getTemperature" service without translation (character 'n') and requesting it with translation (character 'm'). In a production mode context, this option would be fixed for each consumer, depending on the use case. In this example, however, both options are used to demonstrate the approach. As shown in the figure, when choosing a service without translation, the message received is identical to the message sent by the translator. The dynamics of this example are illustrated in the sequence diagram shown earlier in Figure 14.

In the following example, the consumer will request the "getTemperature" service, but with translation (character 'm'). The type of request made is the same as the previous one. The difference is that in this case, the consumer attaches metadata with information from the xsdConsumer schema and the ontology. The provider, in turn, had already registered the "getTemperature" service with metadata that held the xsdProvider and the ontology in the Arrowhead Cloud Service Registry.

When the consumer makes the request, attaching the metadata above, the DITAG-Manager becomes involved in the service provision interaction, as illustrated in the sequence diagram shown earlier in Figure 15. DITAG-Manager, specifically the DITAG-Provider, translates the original message from the provider using the respective translator, which may have been generated during the first request to the "getTemperature" service or reused on subsequent occasions. Reusing the instance enables the translator to respond faster,



POST /ditag/createDynamicTranslator

Parameters Cancel Reset

No parameters

Request body multipart/form-data

providerSchema * required
string(\$binary) Choose File ProviderGrp1.xsd

consumerSchema * required
string(\$binary) Choose File ConsumerGrp1.xsd

ontology * required
string(\$binary) Choose File ontology.owl

Execute

Responses

Code	Description	Links
200	OK	No links

Media type: */

Controls: Accept header.

Example Value | Schema

string

Figure 17: Creating a translator with DITA-Manager.

thereby improving the DITAG-Manager's real-time performance. Figure 24 shows the original message and the same message translated in the DITAG-provider console. As can be seen, the messages are structurally and semantically distinct.

Figure 25, specifically the one on the left, shows the message received by the consumer, which corresponds to the message shown in Figure 24 and translated by the DM. The second figure on the right shows the response to the same "getTemperature" service requested by the consumer, without translation.

As shown in the figure above, the total request time with and without translation is 160ms and 91ms, respectively. These times are reasonable, considering that DM is in the prototype phase and has not yet been optimised.

To demonstrate that DM can operate in real time, a simulation scenario was implemented for a parts assembly cell consisting of a robot and a conveyor, as shown in Figure 26.

The scenario was adapted from the example repository in the CoppeliaSim simulation software (<https://www.coppeliarobotics.com/>). In the adapted example, a small change was made, placing temperature sensors (simulated) and a proximity sensor, to be able to count the number of finished products. The CoppeliaSIM scenario required the development of a new Arrowhead provider, Arrowhead.AssemblyCell.provider, implemented in Python. The settings for its registration as a provider in the Arrowhead cloud are shown in Figure 27. The figure also shows that this provider can work with DITAG-Manager, as it registers its service with metadata that includes an xsdProvider and an ontology.

In this example, the same schemes and ontologies as in the previous examples have been retained. Thus, although the provider provides information on the number of parts assembled, the consumer continues to receive only information on the cell's temperature. For illustrative purposes, the same example is shown in Figure 28, without translation, in which the consumer receives information on the number of parts assembled.

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8084/ditag/createDynamicTranslator' \
  -H 'accept: */*' \
  -H 'Content-Type: multipart/form-data' \
  -F 'providerSchema=@ProviderGrp1.xsd;type=application/xml' \
  -F 'consumerSchema=@ConsumerGrp1.xsd;type=application/xml' \
  -F 'ontology=@ontology.owl'
```

Request URL

```
http://localhost:8084/ditag/createDynamicTranslator
```

Server response

Code
Details

200

Response body

```
({"id":"v1:60a1b62ba86c8f4903d049edbc233b6020badafb68bb0c2509b295155e6cf9:d980aaf2d54e66627395acaba2dc6db148d4be92c77036aa0aaf42679c2f3bb:2f508a24169f1737d3230bfc3def6df0b2361b5d2168ca4dbcb948c78f9a1d","translator":{"package pt.uninova.ditag.translator;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import java.lang.reflect.Field;
import java.util.*;

import org.json.JSONObject;
import org.json.JSONTokener;
import org.json.XML;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.StringWriter;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlAccessorType;

connection: keep-alive
content-length: 28817
content-type: text/plain;charset=UTF-8
date: Fri, 14 Nov 2025 23:33:49 GMT
keep-alive: timeout=60
```

Download

Response headers

```
connection: keep-alive
content-length: 28817
content-type: text/plain;charset=UTF-8
date: Fri, 14 Nov 2025 23:33:49 GMT
keep-alive: timeout=60
```

Responses

Code
Description
Links

200
OK
No links

Figure 18: Resulting translator.

POST /ditag/translateMessage/{id}

Parameters

Cancel
Reset

Name	Description
id * required string (path)	v1:60a1b62ba86c8f4903d049edbc233b6020badafb68bb0c2509b295155e6cf9:d980aaf2d54e66627395acaba2dc6db148d4be92c77036aa0aaf42679c2f3bb:2f508a24169f1737d3230bfc3def6df0b2361b5d2168ca4dbcb948c78f9a1d

Request body required

text/plain

```
<datavalues>
  <sensor1temp>25.5</sensor1temp>
  <sensor2temp>26.8</sensor2temp>
  <sensor3temp>27.3</sensor3temp>
  <sensor4temp>28.1</sensor4temp>
  <relay1state>1</relay1state>
  <units>Celsius</units>
  <Relay2state>0</Relay2state>
</datavalues>
```

Execute
Clear

Figure 19: Request for the translation of a message.


```
provider-1
<datavalues>
  <sensor1temp>22.93</sensor1temp>
  <sensor2temp>23.93</sensor2temp>
  <sensor3temp>22.43</sensor3temp>
  <sensor4temp>22.93</sensor4temp>
  <relay1state>1</relay1state>
  <units>Cel</units>
  <relay2state>0</relay2state>
</datavalues>

<datavalues>
  <sensor1temp>22.06</sensor1temp>
  <sensor2temp>23.06</sensor2temp>
  <sensor3temp>21.56</sensor3temp>
  <sensor4temp>22.06</sensor4temp>
  <relay1state>1</relay1state>
  <units>Cel</units>
  <relay2state>0</relay2state>
</datavalues>

<datavalues>
  <sensor1temp>17.45</sensor1temp>
  <sensor2temp>18.45</sensor2temp>
  <sensor3temp>16.95</sensor3temp>
  <sensor4temp>17.45</sensor4temp>
  <relay1state>1</relay1state>
  <units>Cel</units>
  <relay2state>0</relay2state>
</datavalues>
```

Figure 22: Messages of the provider's requested services.

```
consumer-1
<a3st:map-value-ind a3st:individual="Celsius" a3st:value="C" />
</a3st:map-data-ind-->
<!--continuar
a suportar o map-data-ind-->

</a3st:metadata>
</xs:appinfo>
</xs:annotation>

</xs:schema>, interfaces=[{templateName=generic_http, protocol=http, policy=NONE, properties={
accessAddresses=[localhost, 127.0.0.1], accessPort=8087, operations={getoperation={path=/getTem
perature, method=GET}}, basePath=/telemetry}}, authorizationTokens={}]], warnings=[notExclusiv
e]}
THE URI: http://localhost:8087/telemetry/getTemperature
[Consumer-1] provider response:
<datavalues>
  <sensor1temp>15.14</sensor1temp>
  <sensor2temp>16.14</sensor2temp>
  <sensor3temp>14.64</sensor3temp>
  <sensor4temp>15.14</sensor4temp>
  <relay1state>1</relay1state>
  <units>Cel</units>
  <relay2state>0</relay2state>
</datavalues>

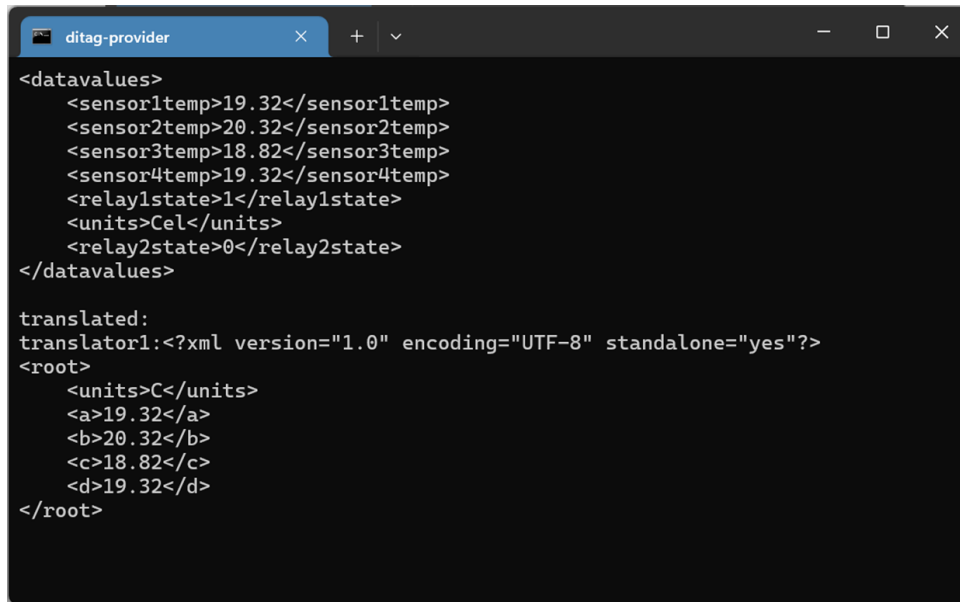
ontologyName=null, xsdSchemaName=null
requestTime(ms): 2166
Request getTemperature service
n: without metadata; m: with metadata;
```

Figure 23: Message without translation received by the consumer.

dependable deployment, orchestration, and evolution of Arrowhead-based systems.

AFO focuses on the core structural elements of an Arrowhead local cloud and the services they expose:

- **Local Cloud:** A bounded context that contains systems. The ontology provides both directions of this relation (contains / isContainedIn) to support intuitive querying.
- **System:** The unit of deployment and operation in Arrowhead. Each system is explicitly decomposed into:
 - **Unit Asset:** The hardware/software component that provides and/or consumes services.



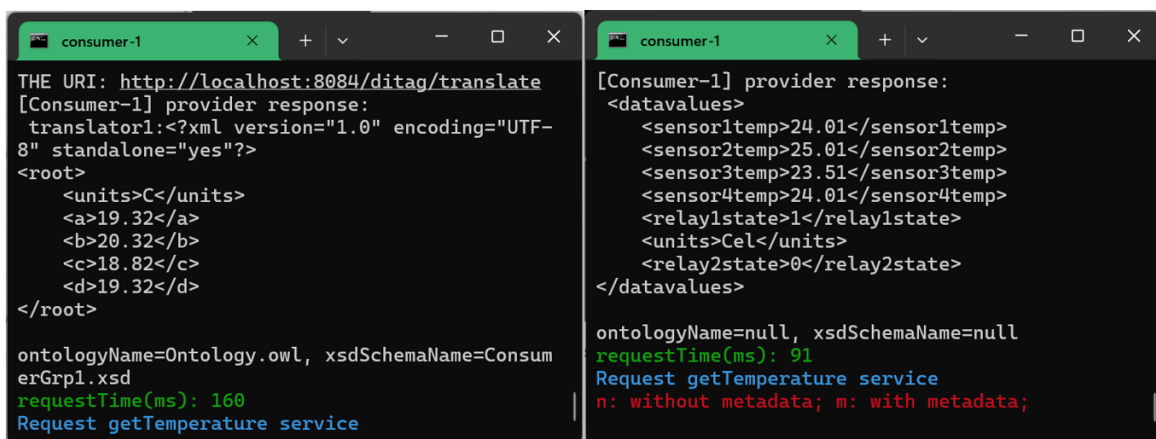
```

<datavalues>
  <sensor1temp>19.32</sensor1temp>
  <sensor2temp>20.32</sensor2temp>
  <sensor3temp>18.82</sensor3temp>
  <sensor4temp>19.32</sensor4temp>
  <relay1state>1</relay1state>
  <units>Cel</units>
  <relay2state>0</relay2state>
</datavalues>

translated:
translator1:<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
  <units>C</units>
  <a>19.32</a>
  <b>20.32</b>
  <c>18.82</c>
  <d>19.32</d>
</root>

```

Figure 24: Original and translated message on the DITAG provider console.



```

THE URI: http://localhost:8084/ditag/translate
[Consumer-1] provider response:
translator1:<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
  <units>C</units>
  <a>19.32</a>
  <b>20.32</b>
  <c>18.82</c>
  <d>19.32</d>
</root>

ontologyName=Ontology.owl, xsdSchemaName=ConsumerGrp1.xsd
requestTime(ms): 160
Request getTemperature service

```

```

[Consumer-1] provider response:
<datavalues>
  <sensor1temp>24.01</sensor1temp>
  <sensor2temp>25.01</sensor2temp>
  <sensor3temp>23.51</sensor3temp>
  <sensor4temp>24.01</sensor4temp>
  <relay1state>1</relay1state>
  <units>Cel</units>
  <relay2state>0</relay2state>
</datavalues>

ontologyName=null, xsdSchemaName=null
requestTime(ms): 91
Request getTemperature service
n: without metadata; m: with metadata;

```

Figure 25: Messages received by the consumer, with translation and without translation.

- * Service
- * Consumed Service
- **Husk**: The communication boundary to the local cloud.
 - * Host
 - * Server

Within a local cloud, services are described with required metadata that supports discovery and orchestration: endpoint URL, service definition, and whether the service is subscribable. The AFO models these attributes so that registries and orchestrators can operate on a common, validated information base. In line with Arrowhead practice, **UnitAssets are modelled to provide at least one service**; consuming behaviour can be added where applicable, but service provision is the baseline for independent operability.

Mandatory and optional core systems (e.g., service registry, orchestrator, certificate authority) are modelled as subclasses of **System**. AFO supports generated metadata (e.g., hasName "serviceregistrar") to help tools and reasoners classify instances and apply profile checks. This lightweight annotation complements the formal axioms and improves traceability from configuration artefacts to ontology individuals.

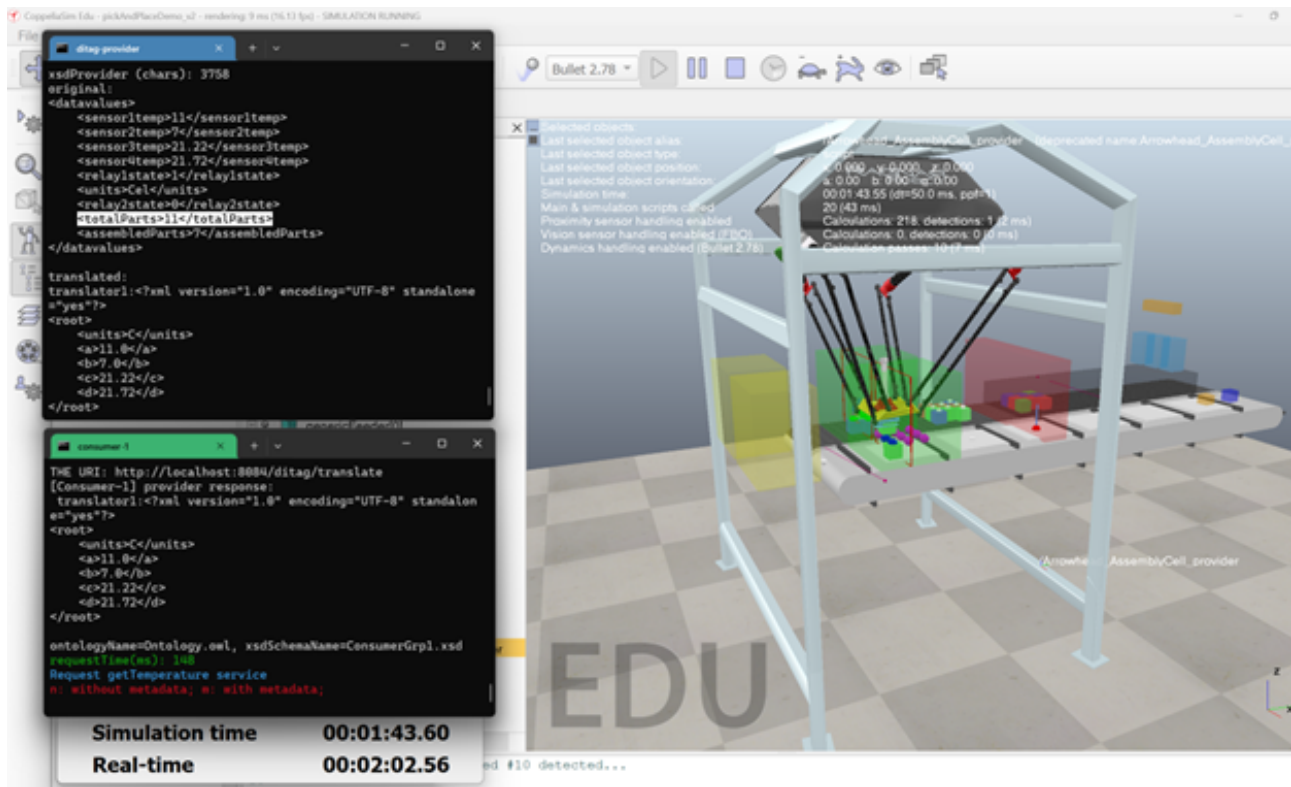


Figure 26: The Assembly cell provider with a consumer and the DITAG-provider.

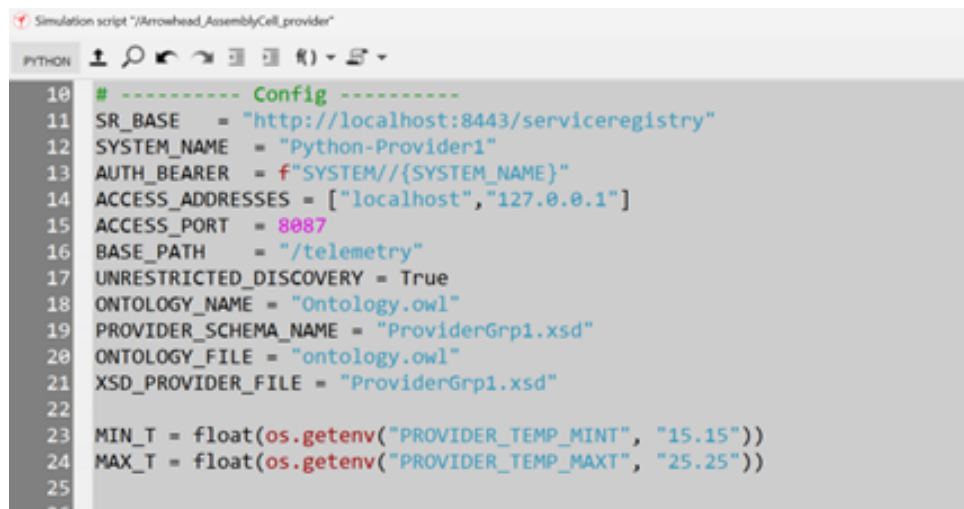


Figure 27: Access configuration for the Arrowhead.AssemblyCell_provider.

In summary, AFO codifies Arrowhead local cloud structure and constraints into a reusable, verifiable knowledge base. It improves the reliability of deployments, shortens integration cycles, and provides a solid foundation for automated orchestration and continuous compliance.

The AFO is part of a larger project bridging standards into a semantic interoperable environment using the Industrial Data Ontology (IDO) as an upper ontology. Some of which include DEXPI, additional STEP standards. The aim is to further test the AFO in UC 2.9.

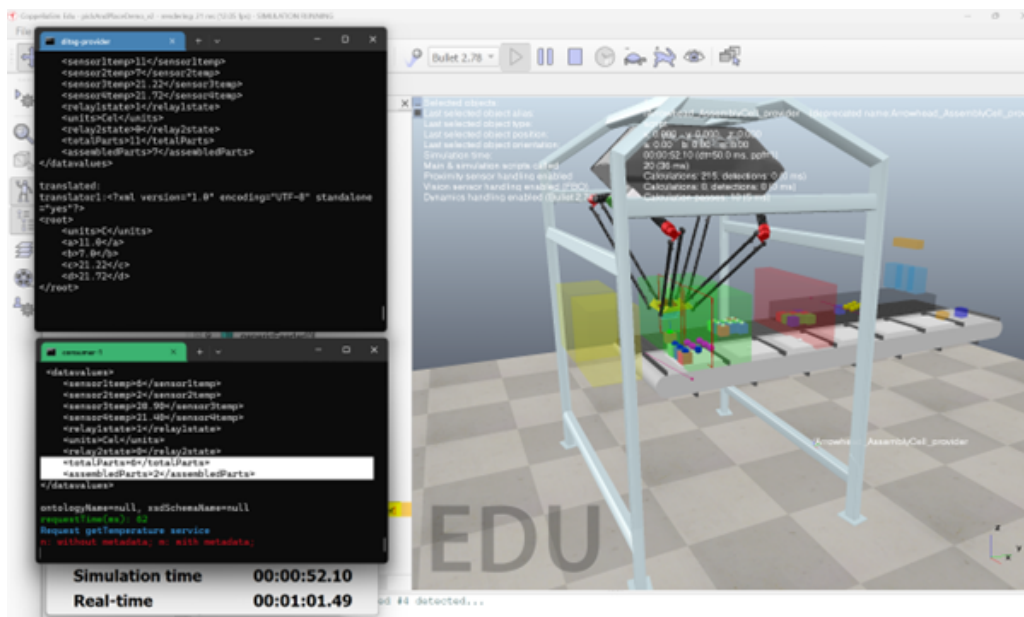


Figure 28: Message with assembly parts received by the consumer.

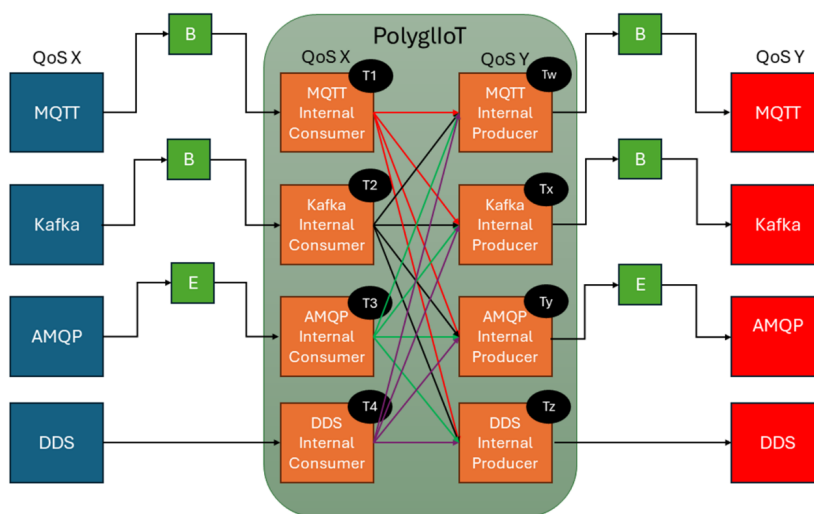


Figure 29: PolyGloT architecture

4.1.4 PolyGloT (Prototype)

PolyGloT, Figure 29, was developed to address interoperability challenges among heterogeneous IoT systems that use different communication middleware technologies. This tool is particularly suitable for industrial IoT environments, smart manufacturing, and distributed automation systems where multiple communication standards coexist. It enables the interconnection of different communication middleware while ensuring Quality of Service (QoS) requirements in message delivery. The supported protocols are MQTT, Kafka, DDS, and AMQP. Thanks to its abstract design, adding support for new middlewares becomes straightforward.

Producers and consumers that rely on PolyGloT for message translation are considered external. They communicate with others of the same type, which are considered internal. The tool provides three levels of QoS, inspired by Kafka and MQTT: At Most Once, At Least Once, and Exactly Once.

The system is deployed within the Arrowhead framework as a service to enhance flexibility, scalability, and

availability. It can also be deployed in a Kubernetes environment, allowing load balancing by dynamically instantiating new tool instances when necessary. The architecture follows a modular approach, with translation modules acting as middleware adapters that handle protocol-specific message serialisation, deserialisation, and QoS management.

Performance tests were conducted on the system to evaluate throughput and latency under various QoS levels and middleware combinations. Results showed that, in general, higher QoS reliability leads to lower performance. DDS experienced a performance penalty due to its JNI-based setup, since it is not natively supported in Java (the language in which the system was developed).

Future work includes integrating PolyglIoT into the new Arrowhead version and increasing security mechanisms, such as generating a private key for both ends while ensuring encryption on the communication.

4.1.5 HAScO/Repo Support for Arrowhead's Semantic Services (Prototype)

Automated interoperability across heterogeneous industrial systems requires more than message-level communication: it requires a shared understanding of the meaning, structure, and context of exchanged data. The Arrowhead Framework provides a powerful service-oriented architecture (SOA) for distributed industrial automation, but does not prescribe mechanisms for formally describing the semantics of the data that services produce or consume. As a result, Arrowhead services typically exchange syntactic payloads (e.g., JSON fields, numeric values) whose semantics must be interpreted manually or inferred from context.

This system addresses this challenge by presenting an ontology-based approach for automated translation of data models within the Arrowhead fPVN project. It introduces a semantic infrastructure designed to enrich Arrowhead services with machine-actionable metadata describing the meaning, provenance, and constraints of exchanged data. The methodology is based on two metadata templates:

- the **Instrument Semantic Template (INS)**, which describes instruments, detectors, actuators, and their functional roles; and
- the **Study Data Dictionary (SDD)**, which provides structured semantic descriptions of dataset variables, including attributes, units, provenance, and contextual associations.

Together, these templates generate semantic annotations that are ingested into a knowledge graph, enabling Arrowhead services to produce semantically enriched outputs. Each value delivered by an enhanced service is assigned a unique URI, allowing consumers and translation agents to query and retrieve explanations through SPARQL.

This ontology-based approach is demonstrated in the Arrowhead Battery Testing Use Case, integrating heterogeneous datasets from AVL and Sandia National Laboratories. The results show that structured semantic metadata supports automated harmonisation of data models across partners.

4.1.5.1 Objectives

The primary objective of Task 4.1 is to define and validate Approach 1: Ontology-Based Translation of Data Models, addressing the challenges of heterogeneity in Arrowhead fPVN data sources. Specific goals include:

1. Provide a structured semantic methodology

Introduce metadata templates (INS and SDD) that enable consistent, ontology-grounded descriptions of instruments, variables, and study components.

2. Enable machine-actionable data interpretation

Demonstrate how semantic annotations allow Arrowhead services and consumers to interpret the meaning of values through semantic URIs and knowledge graph queries.

3. Support automated translation between heterogeneous datasets

Show how semantic metadata enables DITAG translators and other interoperability tools to align data from different partners.

4. Strengthen the Arrowhead Framework without modifying its core

Provide an external semantic layer that integrates smoothly with Arrowhead's existing service registry, orchestration, and communication mechanisms.

5. Establish a foundation for subsequent WP4 tasks

Prepare the semantic infrastructure required by:

- T4.2 (AI-based translation), and
- T4.3 (Model-based translation)

4.1.5.2 Relation to Project Objectives

This solution contributes directly to the main objectives of the Arrowhead fPVN project, especially those addressing **interoperability, predictive integration, and cross-domain data harmonisation**.

Contribution to WP4 Objectives

WP4 aims to enable automated data translation and harmonisation. This solution provides:

- the semantic foundations for aligning heterogeneous data;
 - metadata models describing datasets in a machine-interpretable form;
- a knowledge infrastructure enabling automated semantic queries;
- the initial framework used later by rule-based and ML-based translation systems.

Contribution to WP1 and WP2

The ontology-based metadata produced in this solution enhances:

- **WP1:** High-level semantic modelling of industrial processes, and
- **WP2:** FAIR-compliant data handling strategies.

Contribution to Industrial Use Cases

The semantic approach is validated using:

- Sandia National Laboratories (battery stress and characterisation tests),
- AVL (end-of-life battery assessment and multi-channel diagnostics).

These partners' datasets were chosen because they are heterogeneous in format, naming conventions, and measurement practices—making them ideal for demonstrating ontology-based harmonisation.

Contribution to the Arrowhead Ecosystem

By enabling enriched services that expose semantic explanations, this solution advances Arrowhead from an SOA with minimal data semantics to a **semantically aware interoperable framework**. This supports:

- automated service discovery,
- compatibility assessment,
- reasoning-based selection of services, and
- explainable data-driven decision support.

4.1.5.3 Background and Motivation

The modern industrial automation landscape is characterised by heterogeneous data sources, diverse measurement devices, evolving domain vocabularies, and multi-organisational collaborations. While Arrowhead provides efficient service orchestration and communication, it currently lacks mechanisms for ensuring that data exchanged between services is semantically coherent.

Several challenges motivate this deliverable:

Heterogeneous Data Practices Across Partners

Even when partners measure similar variables, they often differ in:

- variable names (e.g., "U_Volt" vs "Voltage_V"),
- units (e.g., "°C" vs "Kelvin"),
- sampling intervals and cycles,
- measurement device capabilities,
- internal test protocols or cycle definitions.

These differences hinder automated integration.

Arrowhead's Implicit Notion of "Data kind"

Arrowhead services describe the types of data they produce using loosely defined metadata. This approach does not specify:

- what an attribute represents,
- the entity it describes,
- the units or categorical codes,
- the provenance or generating instrument,
- the temporal context of measurements.

This vagueness prevents automated reasoning.

Manual or Missing Semantic Annotations

Real-world deployments rarely include semantic annotations. Manual annotation is error-prone and not scalable.

Need for Machine-actionable Metadata for Digital Twins and Automation

As Arrowhead evolves toward predictive, federated digital ecosystems, automated interpretation of data becomes essential for:

- digital twins,
- predictive diagnostics,
- simulation-driven optimisation,
- cross-factory harmonisation,
- battery passport integration.

These needs require a structured semantic framework.

4.1.5.4 Semantic Foundations

Ontology-based interoperability relies on a set of semantic technologies standardised by the W3C. These technologies enable meaning, context, and provenance to be encoded in machine-interpretable forms. The semantic infrastructure in this solution uses:

- **RDF** for graph-based data representation,
- **RDFS** for class/property hierarchies,
- **OWL** for expressive ontological modeling,
- **PROV-O** for provenance chains,
- **QUDT** for units and quantities,
- **HAScO/SIO/VSTOI** for domain-specific semantics.

A brief overview is provided below.

RDF – Resource Description Framework

RDF represents data as atomic triples: `<subject> <predicate> <object>`

This enables a flexible graph-based model supporting:

- globally unique URIs,
- multi-ontology integration,
- semantic queries (SPARQL),
- incremental knowledge enrichment.

All metadata generated by INS and SDD templates is eventually transformed into RDF.

RDFS – RDF Schema

RDFS provides mechanisms for:

- class hierarchies (`rdfs:Class`, `rdfs:subClassOf`),
- property hierarchies,
- domain and range constraints.

INS templates rely heavily on RDFS to model instrument and detector hierarchies.

OWL – Web Ontology Language

OWL introduces logical constructs for:

- class equivalence,
- property restrictions,
- cardinality constraints,
- description logic reasoning.

OWL enables automated inference about instruments, variables, roles, and experimental relationships.

PROV-O – Provenance Ontology

Used for:

- describing how values were generated,
- identifying agents, activities, and entities,

- constructing lineage and reproducibility chains.

Enhanced services rely on PROV-O to describe the origin of each value.

SIO, HAScO, VSTOI

These ontologies provide:

1. SIO – Semanticscience Integrated Ontology

- Domain-general scientific constructs (e.g., measurements, attributes, entities).

2. HAScO – Human-Aware Science Ontology

- Study designs, variables, data collection processes, and roles.

3. VSTOI – Instrument Hierarchy Extensions

- Detectors, actuators, subcontainers, and sensor semantics.
- Together, they enable a rich semantic layer for Arrowhead data streams.

4.1.5.5 Instrument Semantic Template (INS)

The **Instrument Semantic Template (INS)** provides a structured representation of the sensing and actuation elements involved in a study or operational environment. Its purpose is to formalise the composition, functional components, and semantic roles of instruments so that downstream processes—such as data annotation and translation—can accurately reflect the context in which data is generated.

The INS template is based on the VSTOI ontology (Instrument Ontology Extensions) and aligns with principles of modularity, provenance, and semantic clarity. It comprises several sheets, each mapping directly to a semantic structure in the knowledge graph.

Purpose of the INS Template

The template addresses several needs in Arrowhead data exchange:

- **Explicit modeling of instrument structure**

Clarifies how instruments, detectors, and actuators relate.

- **Representation of sensing capabilities**

Detector stems express classes of measurable attributes.

- **Separation of physical structure from semantic role**

A detector's location (slot element) is distinct from its type (stem).

- **Reusability across datasets, partners, and domains**

Detectors and slot elements can be reused in multiple instruments.

Template Components

Info Sheet - Provides internal references, authorship, and versioning metadata.

Namespaces Sheet - Lists all ontology prefixes used in the template, including:

- hasco:, sio:, prov:, vstoi:, qudt:, unit:, xsd:, etc.

This ensures that each column maps to a known ontology.

Instruments Sheet - Registers each instrument, including:

- URI
- instrument class (rdfs:subClassOf)

- manufacturer
- version
- documentation links
- first slot element (if hierarchical)

SlotElements Sheet - Models the physical structure of an instrument:

- **belongsTo:** parent instrument or subcontainer
- **hasComponent:** detector or actuator installed in the slot
- **hasNext / hasPrevious / hasFirst:** ordering relationships

Slot elements may represent channels, connector arrays, internal boards, etc.

DetectorStems and Detectors - Detector stems describe general measurement classes (e.g., voltage sensing). Detectors represent actual components installed in an instrument. Each detector links:

- to a detector stem
- optionally to a codebook (categorical domains)

ActuatorStems and Actuators - Analogous to detectors, but representing actuation.

CodeBooks and ResponseOptions - Used for categorical variables or bounded modes (e.g., "ON", "OFF", "Error State").

Role in Ontology-Based Translation

The INS provides the semantic backbone required for:

- determining measurement provenance
- interpreting which component produced a value
- mapping instrument-specific variables to semantic abstractions
- supporting automated detection of equivalent sensors across partners

Without INS metadata, variable semantics would remain incomplete.

4.1.5.6 Study Data Dictionaries (SDDs)

While the INS template describes the *structure and semantics of instruments*, the **SDD template describes the semantics of the dataset variables themselves**. SDDs formalise the meaning of variables in tabular datasets so that each column becomes a machine-interpretable semantic record.

Purpose of the SDD Template

The SDD enables:

- machine-actionable interpretation of variables
- explicit definition of units, roles, entities, and attributes
- consistent alignment across datasets
- generation of semantic descriptions for individual values
- preparation for automated translation (DITAG agents)

It eliminates the ambiguity inherent in informal column labels.

The Dictionary Mapping Sheet

This is the core of the SDD. Each row describes one dataset variable. Common fields include:

- **Column:** The original column name
- **Attribute:** The semantic predicate describing what is measured
- **attributeOf:** The entity whose property is being measured
- **Unit:** QUDT unit (or categorical domain)
- **Role:** Entity role in the study (e.g., test subject, device)
- **Time:** Timeline element or procedural phase
- **Relation / inRelationTo:** Additional contextual links
- **Provenance:** Generation or derivation activity

Codebooks and Categorical Semantics

The Codebook sheet defines:

- codes
- labels
- meanings
- semantic classes

This ensures categorical semantics can be aligned across partners.

Timeline Specification

The Timeline sheet defines:

- cycle steps
- temporal phases
- event windows

This allows alignment of Sandia's cycle phases with AVL's EoL process steps.

Benefits for Arrowhead

SDDs:

- provides a machine-readable blueprint for semantic annotation
- support knowledge graph population
- enable automated reasoning over variables
- serves as the basis for semantic translations

4.1.5.7 Enhanced Semantic Services

Enhanced Semantic Services extend standard Arrowhead services by adding semantic metadata and assigning URIs to values produced during operation.

They provide a bridge between the Arrowhead SOA ecosystem and the semantic knowledge graph.

Motivation

Arrowhead services typically do not expose:

- meaning of each value
- provenance (how the value was produced)

- context or units
- relationships to other values

Enhanced Semantic Services introduce:

- **semantic profiles**
- **value-level URIs**
- **automatic insertion of values into the knowledge graph**
- **on-demand semantic explanations via SPARQL**

How Enhanced Services Work

1. A service receives a request.
2. It computes or retrieves a value.
3. The value is inserted into the knowledge graph and assigned a URI such as:
`http://arrowhead.org/kg/value/26752`
4. The service returns:
 - the raw value (e.g., "27.5"), and
 - the URI referencing its semantic description.
5. Consumers query the URI for semantic explanations.

This transforms Arrowhead from a syntactic messaging framework into a semantically enriched ecosystem.

Benefits

Enhanced Semantic Services provide:

- **explainability** (values become self-describing)
- **harmonization** across heterogeneous services
- **rich provenance tracking**
- **interoperability** through ontology-grounded semantics

4.1.5.8 URI-Based Knowledge Graph Integration

The assignment of a unique URI to each measured value is essential for semantic interoperability. A value becomes a first-class resource in the knowledge graph, capable of participating in semantic relations.

URI Assignment

Values are inserted into the knowledge graph with URIs of the form:

`http://arrowhead.org/kg/value/26752`

Each URI becomes an anchor for semantic metadata. Example triples:

```

1 <http://arrowhead.org/kg/value/26752>
2   a                               sio:MeasurementValue ;
3   sio:hasValue                    "27.5"^^xsd:float ;
4   sio:isAttributeOf               ah:BatteryCell_03 ;
5   qudt:unit                       unit:DegreeCelsius ;
6   ah:measuredUsing                ah:Channel_7 ;
7   prov:generatedAtTime            "2024-01-15T10:07:25Z" .

```

Advantages

- Every value is resolvable.
- Context is persistently linked.
- Downstream processes can inspect semantics at runtime.
- Translators can map values from different studies through shared ontology terms.

Integration with Arrowhead

This does **not** modify the Arrowhead core; instead, it augments it through a semantic overlay.

4.1.5.9 Querying Enhanced Services Using SPARQL

A major benefit of assigning URIs to values is the ability to query their meaning and provenance.

Basic Semantic Query

```
1 SELECT ?property ?unit ?entity ?device ?time
2 WHERE {
3   <http://arrowhead.org/kg/value/26752>
4     a                               ?property ;
5     qudt:unit                       ?unit ;
6     sio:isAttributeOf               ?entity ;
7     ah:measuredUsing                ?device ;
8     prov:generatedAtTime            ?time .
9 }
```

Provenance Chain Retrieval

```
1 SELECT ?activity ?input
2 WHERE {
3   <http://arrowhead.org/kg/value/26752>
4     prov:wasGeneratedBy ?activity .
5     ?activity prov:used ?input .
6 }
```

Cross-Dataset Queries

SPARQL can retrieve:

- all voltage values from both AVL and Sandia,
- all measurements taken during a specific cycle step,
- all values generated by a specific detector type,
- equivalence classes of variables derived from SDD alignment.

4.1.5.10 Case Studies: Sandia and AVL

The ontology-based approach in this deliverable was validated on two real datasets from the Arrowhead Battery Testing Use Case.

Sandia National Laboratories

Sandia's dataset includes:

- multi-chemistry lithium-ion cycling data

- voltage, current, temperature, resistance measurements
- safety and stress test conditions
- detailed test cycle phases

The SDD created for Sandia encodes:

- entity-role relations
- detector assignments
- timeline phase mappings
- provenance for derived metrics

AVL End-of-Life (EoL) Dataset

AVL's dataset focuses on:

- end-of-life assessment
- multi-channel diagnostics
- cycle-based degradation modelling
- extended metadata about ageing procedures

The SDD harmonises:

- variable labels
- units
- channel semantics
- timeline definitions

Semantic Harmonization Outcome

By mapping both partners' datasets to common ontology concepts:

- equivalent variables were aligned semantically
- cross-dataset querying became possible
- enhanced services provided identical semantic explanations across partners

This demonstrates robust semantic interoperability across heterogeneous industrial systems.

4.1.5.11 Conclusion

This section presents an ontology-based approach for automated translation of heterogeneous data models in the Arrowhead fPVN project. By introducing the Instrument Semantic (INS) and Study Data Dictionary (SDD) templates, this work establishes a structured, multi-ontology methodology for representing instruments, variables, and study contexts. These templates generate machine-actionable semantics that allow Arrowhead services to produce not just values, but explicit explanations of meaning, provenance, units, and relationships.

The approach integrates seamlessly into the Arrowhead architecture through **Enhanced Semantic Services**, which assign URIs to values and populate a knowledge graph for downstream querying. This enables consumers and translation agents to access detailed semantic explanations and perform automated harmonisation across datasets from different partners.

The Sandia and AVL battery datasets provided real-world validation, demonstrating that ontology-based harmonisation is feasible despite differences in naming conventions, units, structural assumptions, and measurement protocols. The resulting semantic alignment strengthens Arrowhead's capacity for interoperable analytics, predictive modelling, and cross-laboratory integration.

This deliverable forms the basis for subsequent approaches in WP4—rule-based and machine-learning-based translations—and contributes directly to the broader goals of the Arrowhead fPVN project: achieving scalable, cross-domain, semantically rich interoperability for the next generation of industrial ecosystems.

4.1.6 HASCO/Repo: Battery Testing Datasets as Foundation for Demonstrating Eclipse Arrowhead Semantic Integration (Prototype)

4.1.6.1 Battery Datasets as Training Environments for Eclipse Arrowhead Integration

The Sandia battery dataset serves not only as a scientific resource but also as a comprehensive training and demonstration environment for Eclipse Arrowhead's semantic integration capabilities. By encoding battery testing variables with rich semantic metadata through HASCO/Repo, the dataset enables several critical demonstrations:

4.1.6.2 Semantic Service Registration and Discovery

Each battery testing variable (voltage, current, temperature, state of charge) is registered with explicit semantic identifiers that resolve to formal ontological descriptions in HASCO/Repo. This transforms Arrowhead's traditionally syntactic service registry into a semantically aware discovery mechanism. When a consumer searches for battery monitoring services, the orchestrator can match requests not just on interface compatibility but on the semantic equivalence of measured attributes, enabling more intelligent service selection.

The semantic registration process enriches standard Arrowhead service metadata with payload schema references and HASCO/Repo URIs. For example, a voltage measurement service registered with semantic identifier http://hascorepo.org/sdd/sandia_battery_v1 enables the orchestrator to retrieve the complete semantic description of the voltage variable, including its physical entity reference (battery cell), measurement unit (volts), detector configuration (cyclor channel), and temporal context (cycle phase). This semantic enrichment enables orchestration decisions based on meaning rather than syntactic interface matching.

Demonstrating Payload Semantic Enrichment

The battery dataset's existing semantic annotations—including units, measurement contexts, detector configurations, and provenance chains—provide concrete examples of how HASCO/Repo enriches Arrowhead service payloads. A voltage measurement is no longer simply a floating-point number; it becomes a semantically grounded observation linked to its physical entity (battery cell), measurement instrument (cyclor channel), experimental context (charge/discharge cycle), and temporal position within the test protocol.

This semantic enrichment directly demonstrates Eclipse Arrowhead's capability to support knowledge-level interoperability rather than mere data exchange. When a service returns a voltage value, it includes both the numerical measurement and a HASCO/Repo URI that clients can resolve to retrieve:

- The ontological definition of voltage as a physical quantity
- The specific battery cell entity being measured
- The detector stem describing the voltage measurement capability
- The provenance chain showing how the value was generated
- The temporal anchor, placing the measurement within the test protocol
- The unit specification enabling automatic conversion if needed

This rich semantic context enables consumers to interpret, validate, and integrate battery data with a full understanding of its meaning and origin.

Training DITAG's Semantic Reasoning Engine

The diversity of variables in the Sandia dataset—ranging from direct measurements (voltage, current) to derived metrics (capacity fade, internal resistance) to categorical states (cycle phase, alarm conditions)—provides a rich environment for training and validating DITAG's ontology-based reasoning capabilities. DITAG learns to recognise semantic equivalence patterns, subsumption relationships, and transformation requirements by analysing how battery domain concepts are formally represented in HASCO/Repo.

For instance, DITAG analyses how Sandia represents voltage measurements with semantic annotations linking to SIO's `sio:Voltage` concept, QUDT's unit specifications, and VSTOI's detector stem hierarchies. These learned patterns generalise to other industrial domains where similar entity-attribute-measurement structures exist, enabling DITAG to apply battery-domain reasoning strategies to manufacturing, laboratory automation, and other Industry 4.0 contexts.

4.1.6.3 AVL Dataset Demonstrating Cross-Organisational Eclipse Arrowhead Integration

The AVL battery dataset serves a critical role in demonstrating how Eclipse Arrowhead, when enhanced with HASCO/Repo semantic infrastructure, enables true cross-organisational interoperability in industrial ecosystems.

Exercising Heterogeneous Service Integration

AVL's data collection practices, variable naming conventions, and experimental protocols differ significantly from Sandia's approach, reflecting real-world organisational diversity. AVL uses different battery cyclers equipment, employs distinct experimental procedures for end-of-life testing, and organises data using terminology specific to their laboratory practices. For example, AVL uses variable names like `U_Volt`, `I_Amp`, and `CycleNumber`, while Sandia uses `voltage_V`, `current_A`, and `cycle_id`.

By deploying both datasets as Arrowhead services with HASCO/Repo semantic annotations, we demonstrate that Eclipse Arrowhead's orchestration capabilities can discover and integrate services from different providers without requiring standardisation of internal data models. Each organisation maintains its own naming conventions, data structures, and experimental protocols, while HASCO/Repo's semantic layer provides the common conceptual foundation enabling interoperability. This preserves organisational autonomy while achieving semantic interoperability—a fundamental Industry 4.0 requirement.

Validating DITAG's Multi-Source Harmonisation

The combination of AVL and Sandia services explicitly exercises DITAG's ability to harmonise payloads from multiple providers within a single orchestration response. When a consumer requests battery testing data, DITAG must simultaneously assess semantic compatibility for both AVL's `U_Volt` representation and Sandia's `voltage_V` representation, mapping both to the consumer's expected `cell.voltage` field.

This multi-source harmonisation validates DITAG's scalability and demonstrates its capability to handle N-to-1 semantic translation scenarios common in federated industrial ecosystems. DITAG queries HASCO/Repo to retrieve the SDDs for both providers, analyses the semantic annotations to determine that both variables reference the same ontological concept (`sio:Voltage`), and generates translation mappings that preserve semantic correctness while adapting to each provider's syntactic representation.

Demonstrating Provider-Agnostic Consumer Applications

By consuming both AVL and Sandia services through DITAG-mediated orchestration, consumer applications achieve complete independence from provider-specific payload structures. The consumer application code remains unchanged whether it receives data from AVL, Sandia, or future battery testing partners, as long as those partners register their services with appropriate HASCO/Repo semantic identifiers.

This demonstrates Eclipse Arrowhead's capability to support scalable, evolvable service ecosystems where providers can join, leave, or modify their implementations without disrupting existing consumers. When a new battery testing laboratory joins the ecosystem, it only needs to create appropriate INS and SDD templates describing its instruments and variables, register its services with HASCO/Repo semantic identifiers, and DITAG will automatically enable interoperability with existing consumers.

4.1.6.4 Explicit Exercise of Harmonisation Capabilities Through Combined AVL and Sandia Services

The simultaneous deployment of AVL and Sandia battery testing services provides a concrete demonstration of how HASCO/Repo and DITAG enable cross-organisational harmonisation in Eclipse Arrowhead. This combined deployment explicitly exercises several critical harmonisation capabilities:

Vocabulary Harmonisation

AVL uses `U_Volt`, `I_Amp`, and `T_degC` to represent voltage, current, and temperature, while Sandia uses `voltage_V`, `current_A`, and `temp_C`. Despite these syntactic differences, HASCO/Repo's semantic annotations enable DITAG to recognise that both vocabularies reference identical domain concepts.

The SDD for each provider maps these variables to shared ontological entities: `sio:Voltage`, `sio:ElectricCurrent`, and `sio:Temperature`. DITAG analyses these ontological mappings and automatically derives vocabulary translation rules that preserve semantic meaning while adapting variable names. This demonstrates that Eclipse Arrowhead services can maintain organisational naming conventions while still achieving semantic interoperability — organisations need not abandon their established practices to participate in federated ecosystems.

Unit System Harmonisation

The AVL and Sandia datasets may employ different unit systems or unit representations. For example, one laboratory might report voltage in millivolts while another uses volts, or temperature might be expressed in Celsius versus Kelvin. HASCO/Repo's integration with QUDT (Quantities, Units, Dimensions, and Types) ontology enables explicit unit declaration in SDDs.

DITAG leverages these unit annotations to detect when unit conversion is required and to generate transformation logic that preserves physical correctness. For instance, if AVL reports voltage in millivolts while the consumer expects volts, DITAG derives the appropriate scaling factor ($mV * 0.001 = V$) from the QUDT unit relationships encoded in HASCO/Repo. This automatic unit harmonisation prevents integration errors and ensures that cross-laboratory comparisons are physically meaningful.

Temporal Harmonisation

Battery testing protocols involve complex temporal structures: charge/discharge cycles, rest periods, pulse sequences, and long-term ageing phases. AVL and Sandia organise these temporal elements differently, using different cycle numbering schemes, phase labels, and timestamp formats. AVL might use process-driven timeline representations where phases are labelled by experimental procedure (e.g., "formation_charge", "aging_discharge"), while Sandia might use cycle-based representations with numeric identifiers.

The SDD's Timeline sheet enables both organisations to explicitly define their temporal structures and map them to standardised temporal ontology concepts. DITAG uses these temporal annotations to align events across providers, enabling consumers to correlate measurements from AVL and Sandia even when they use different temporal reference frames. This temporal harmonisation is essential for comparative ageing analysis that requires synchronisation of battery states across datasets—for example, determining whether a voltage measurement from AVL's cycle 50 corresponds to a similar degradation state as Sandia's cycle 100.

Structural Schema Harmonization

Beyond individual variable mappings, AVL and Sandia services expose different payload structures. AVL may use nested JSON objects grouping related measurements by cycle phase:

```
1 {
2   "cycle": 50,
3   "charge_phase": {
4     "U_Volt": 4.2,
5     "I_Amp": 2.0
6   },
7   "discharge_phase": {
8     "U_Volt": 3.0,
9     "I_Amp": 2.0
10  }
11 }
```

while Sandia may use flat arrays with metadata headers:

```
1 [
2   {"cycle_id": 50, "phase": "charge", "voltage_V": 4.2, "current_A": 2.0},
3   {"cycle_id": 50, "phase": "discharge", "voltage_V": 3.0, "current_A": 2.0}
4 ]
```

DITAG's harmonisation extends beyond field renaming to include structural transformation. It generates translation logic that can reshape AVL's nested structures into Sandia-compatible flat representations or vice versa, based on the consumer's requirements. This structural flexibility demonstrates Eclipse Arrowhead's ability to accommodate diverse payload design philosophies while maintaining semantic consistency.

Provenance-Aware Multi-Source Integration

When a consumer receives harmonised data from both AVL and Sandia, each value retains complete provenance information indicating its origin, the transformation applied, and the semantic basis for harmonisation. This provenance tracking—enabled by HASCO/Repo's integration with PROV-O—allows consumers to maintain source attribution even when data from multiple providers is merged into unified analytical datasets.

For example, a harmonised voltage measurement includes PROV-O annotations specifying:

- `prov:wasGeneratedBy` — the original service (AVL or Sandia) that produced the measurement

- `prov:wasAttributedTo` — the laboratory agent responsible for data collection
- `prov:wasDerivedFrom` — the original variable name before harmonisation
- `prov:used` — the SDD version used for semantic translation
- Custom annotations describing the DITAG transformation applied

This provenance is critical for scientific reproducibility and for industrial traceability requirements where data lineage must be auditable. Regulatory compliance in battery manufacturing and automotive industries often requires complete documentation of data origins and transformations, which HASCO/Repo's provenance infrastructure provides automatically.

4.1.6.5 Retroactive Semantic Enrichment Workflow

The battery testing scenarios provided by AVL and Sandia demonstrate a critical methodological contribution: existing scientific and industrial datasets can be retroactively transformed into semantically enriched Eclipse Arrowhead services without requiring changes to data collection infrastructure or experimental protocols. This retroactive semantic enrichment is achieved through the following workflow:

Step 1: INS Template Creation

Battery testing instruments (cyclers, data acquisition systems, environmental chambers) are described using the INS metadata template, capturing detector hierarchies, actuator configurations, and measurement capabilities. For example, the Sandia PAT-Tester cycler is modelled with:

- Physical instrument classification in VSTOI hierarchy
- Detector stems for voltage, current, temperature measurements
- Slot elements representing individual cycler channels
- Codebooks defining operational states and alarm conditions

Similarly, AVL's end-of-life testing equipment is described with INS templates reflecting its specific detector configurations and channel arrangements. This instrument-level semantic modelling ensures that consumers can understand not just what was measured, but how and with what equipment.

Step 2: SDD Template Generation

Each variable in the battery dataset (voltage, current, temperature, derived metrics) is annotated with semantic metadata including entity references, attribute types, units, detector associations, and provenance information. The SDD creation process involves:

- Identifying the entity being measured (e.g., specific battery cell)
- Specifying the attribute type using ontology references (e.g., `sio:Voltage`)
- Declaring units using QUDT specifications
- Linking to the detector stem from the INS template
- Defining temporal anchors in test protocols
- Specifying provenance relationships

This systematic semantic annotation transforms raw battery data into ontology-grounded knowledge that can support automated reasoning and harmonisation.

Step 3: HASCO/Repo Ingestion

INS and SDD templates are ingested into HASCO/Repo's knowledge graph, generating persistent URIs for all semantic resources. The ingestion process creates RDF triples representing instruments, variables, entities,

timelines, and provenance relationships, all accessible via SPARQL queries. Each resource receives a globally unique identifier that can be referenced in Arrowhead service registrations.

Step 4: Service Wrapper Deployment

Arrowhead-compliant service wrappers are deployed around the battery datasets, exposing REST endpoints that return measurement data along with HASCO/Repo URI references. These wrappers:

- Implement standard Arrowhead HTTP-JSON interfaces
- Query the underlying battery data stores (CSV files, databases, streaming systems)
- Augment each returned value with its corresponding HASCO/Repo URI
- Support content negotiation for RDF-based semantic descriptions

The service wrappers require no modifications to the original data collection systems—they simply provide a semantic-aware view of existing data.

Step 5: Semantic Registration

Services are registered in the Arrowhead Service Registry with metadata linking to their HASCO/Repo semantic identifiers, enabling semantic-aware orchestration. The registration includes:

- Standard Arrowhead metadata (service definition, interface, endpoint)
- Semantic payload URI pointing to the SDD in HASCO/Repo
- Optional capability descriptors for DITAG-based translation support

Once registered, the services become discoverable through semantic queries and can participate in DITAG-mediated orchestration.

Significance of Retroactive Enrichment

This workflow demonstrates that legacy datasets and existing data collection systems can participate in semantic Arrowhead ecosystems without requiring re-instrumentation or protocol changes. Organisations can preserve their existing experimental infrastructure, data formats, and operational procedures while still achieving semantic interoperability with partners using different systems. This significantly lowers the barrier to adoption of semantic interoperability infrastructure, making it practical for industrial consortia, multi-laboratory research projects, and cross-organisational collaboration in Industry 4.0 contexts.

The battery testing use cases thus validate not only the technical feasibility of semantic Arrowhead orchestration but also its practical applicability to real-world industrial and scientific ecosystems where heterogeneity is the norm and standardisation is often infeasible or undesirable.

4.1.6.6 Sandia Battery Studies

Sandia conducted large-scale cycling experiments across several lithium-ion chemistries. Their datasets capture:

- charge/discharge cycles,
- voltage, current, temperature,
- efficiency metrics,
- safety behaviour under stress profiles.

The SDD generated for Sandia's dataset specifies detailed semantic mappings for these variables, including:

- detector stem classes (e.g., voltage, temperature, impedance),
- provenance from the PAT-Tester configuration,

- timeline anchors for cycle phases.

Simulating Arrowhead Services from Sandia Data

Using the semantically annotated Sandia dataset, we deployed simulated Eclipse Arrowhead services that expose battery testing data through standard Arrowhead interfaces. Each service was registered in the Arrowhead Service Registry with:

- Service Definition: battery-cycle-data
- Interface: HTTP-JSON REST endpoint
- Semantic Payload URI: http://hascorepo.org/sdd/sandia_battery_v1
- Metadata: References to HASCO/Repo-stored SDD and INS templates

The service endpoint returns structured JSON payloads containing voltage, current, temperature, and timestamp data. Each returned value includes both the measurement and its corresponding HASCO/Repo URI, enabling clients to retrieve full semantic context on demand. This demonstrates how existing scientific datasets can be retroactively transformed into semantically enriched Arrowhead services without modifying the original data collection infrastructure.

Battery Datasets as Training Environments for Eclipse Arrowhead Integration

The Sandia battery dataset serves not only as a scientific resource but also as a comprehensive training and demonstration environment for Eclipse Arrowhead's semantic integration capabilities. By encoding battery testing variables with rich semantic metadata through HASCO/Repo, the dataset enables several critical demonstrations:

Semantic Service Registration and Discovery. Each battery testing variable (voltage, current, temperature, state of charge) is registered with explicit semantic identifiers that resolve to formal ontological descriptions in HASCO/Repo. This transforms Arrowhead's traditionally syntactic service registry into a semantically aware discovery mechanism. When a consumer searches for battery monitoring services, the orchestrator can match requests not just on interface compatibility but on semantic equivalence of measured attributes, enabling more intelligent service selection.

Demonstrating Payload Semantic Enrichment. The battery dataset's existing semantic annotations—including units, measurement contexts, detector configurations, and provenance chains—provide concrete examples of how HASCO/Repo enriches Arrowhead service payloads. A voltage measurement is no longer simply a floating-point number; it becomes a semantically grounded observation linked to its physical entity (battery cell), measurement instrument (cycler channel), experimental context (charge/discharge cycle), and temporal position within the test protocol. This semantic enrichment directly demonstrates Eclipse Arrowhead's capability to support knowledge-level interoperability rather than mere data exchange.

Training DITAG's Semantic Reasoning Engine. The diversity of variables in the Sandia dataset—ranging from direct measurements (voltage, current) to derived metrics (capacity fade, internal resistance) to categorical states (cycle phase, alarm conditions)—provides a rich environment for training and validating DITAG's ontology-based reasoning capabilities. DITAG learns to recognise semantic equivalence patterns, subsumption relationships, and transformation requirements by analysing how battery domain concepts are formally represented in HASCO/Repo. These learned patterns generalise to other industrial domains where similar entity-attribute-measurement structures exist.

- aligning cycle timelines using the SDD Timeline sheet,
- ensuring cross-lab semantic equivalence (e.g., `voltage_V = U_Volt`).

4.1.6.7 AVL Battery Studies

AVL conducted complementary battery end-of-life testing with a focus on:

- accelerated ageing protocols,
- capacity degradation under various stress conditions,

- internal resistance evolution,
- thermal behaviour during high-rate cycling.

The AVL dataset employs different terminology, experimental protocols, and data structures compared to Sandia, yet describes fundamentally similar battery testing phenomena. This heterogeneity makes AVL an ideal partner for demonstrating cross-laboratory semantic harmonisation.

Simulating Arrowhead Services from AVL Data

Parallel to the Sandia deployment, AVL's dataset was exposed as Eclipse Arrowhead services with complementary semantic annotations:

- Service Definition: battery-eol-analysis
- Interface: HTTP-JSON REST endpoint
- Semantic Payload URI: http://hascorepo.org/sdd/avl_eol_v1
- Metadata: References to HASCO/Repo-stored SDD with AVL-specific timeline and variable mappings

Although AVL and Sandia services expose structurally different payloads—with different variable names (e.g., `voltage_V` vs. `U_Volt`), different JSON schemas, and different cycle phase representations—both services share semantic grounding through their HASCO/Repo annotations. This structural heterogeneity, combined with semantic alignment, provides the ideal testbed for demonstrating DITAG's harmonisation capabilities.

AVL Dataset as Demonstration of Cross-Organisational Eclipse Arrowhead Integration

The AVL battery dataset serves a critical role in demonstrating how Eclipse Arrowhead, when enhanced with HASCO/Repo semantic infrastructure, enables true cross-organisational interoperability in industrial ecosystems:

Exercising Heterogeneous Service Integration. AVL's data collection practices, variable naming conventions, and experimental protocols differ significantly from Sandia's approach, reflecting real-world organisational diversity. By deploying both datasets as Arrowhead services with HASCO/Repo semantic annotations, we demonstrate that Eclipse Arrowhead's orchestration capabilities can discover and integrate services from different providers without requiring standardisation of internal data models. This preserves organisational autonomy while achieving semantic interoperability—a fundamental Industry 4.0 requirement.

Validating DITAG's Multi-Source Harmonisation. The combination of AVL and Sandia services explicitly exercises DITAG's ability to harmonise payloads from multiple providers within a single orchestration response. When a consumer requests battery testing data, DITAG must simultaneously assess semantic compatibility for both AVL's `U_Volt` representation and Sandia's `voltage_V` representation, mapping both to the consumer's expected `cell.voltage` field. This multi-source harmonisation validates DITAG's scalability and demonstrates its capability to handle N-to-1 semantic translation scenarios common in federated industrial ecosystems.

Demonstrating Provider-Agnostic Consumer Applications. By consuming both AVL and Sandia services through DITAG-mediated orchestration, consumer applications achieve complete independence from provider-specific payload structures. The consumer application code remains unchanged whether it receives data from AVL, Sandia, or future battery testing partners, as long as those partners register their services with appropriate HASCO/Repo semantic identifiers. This demonstrates Eclipse Arrowhead's capability to support scalable, evolvable service ecosystems where providers can join, leave, or modify their implementations without disrupting existing consumers.

4.1.6.8 Harmonised Interpretation and Integration

By using SDDs and enhanced services:

- Sandia and AVL variables were mapped to shared ontology concepts;
- differences in naming, units, and schema were resolved semantically;
- values from each dataset became cross-queryable through SPARQL.

This demonstrates that Arrowhead interoperability extends beyond mere communication—it supports genuine knowledge-level integration across laboratories.

DITAG-Mediated Orchestration: Explicit Demonstration of Cross-Laboratory Harmonisation

To explicitly demonstrate DITAG's harmonisation capabilities within Eclipse Arrowhead orchestration, we constructed a cross-laboratory integration scenario involving both AVL and Sandia services. This scenario exercises the complete semantic interoperability stack: service registration with semantic identifiers, orchestration with semantic reasoning, and automated payload harmonisation. Critically, this scenario demonstrates how the battery testing datasets collected by AVL and Sandia — originally created for scientific analysis—can be repurposed to simulate, validate, and train Eclipse Arrowhead's semantic orchestration infrastructure.

Scenario Description

A battery analytics application requires aggregated cycle data from multiple sources to perform comparative ageing analysis. The consumer application is designed to work with a standardised payload schema referencing the following variables:

- `cell_voltage` (volts)
- `cell_current` (amperes)
- `cell_temperature` (Celsius)
- `cycle_number` (integer)
- `timestamp` (ISO 8601)

The consumer initiates an Arrowhead orchestration request for services matching the battery-cycle-data service definition.

Orchestration and Semantic Reasoning

The Arrowhead Orchestrator identifies two candidate providers:

- Sandia Service with semantic URI: `http://hascorepo.org/sdd/sandia_battery_v1`
- AVL Service with semantic URI: `http://hascorepo.org/sdd/avl_eol_v1`

Both services match the requested service definition but expose payloads with different structures:

- Sandia uses: `voltage_V`, `current_A`, `temp_C`, `cycle_id`, `time_utc`
- AVL uses: `U_Volt`, `I_Amp`, `T_degC`, `CycleNumber`, `TestTime`

The Orchestrator forwards the semantic identifiers to DITAG. DITAG queries HASCO/Repo to retrieve the SDD descriptions for all three schemas (consumer, Sandia provider, AVL provider). Using the semantic annotations, DITAG determines:

- Sandia's `voltage_V` is semantically equivalent to the consumer's `cell_voltage`
- AVL's `U_Volt` is semantically equivalent to the consumer's `cell_voltage`
- Both providers' current and temperature variables map to the consumer's corresponding fields
- Cycle number and timestamp fields are semantically compatible despite naming differences

DITAG generates translation mappings for both providers, specifying field-level transformations that preserve semantic intent. These mappings are returned to the Orchestrator, which includes them in the orchestration response.

Service Invocation and Harmonisation

The consumer receives orchestration results indicating that both Sandia and AVL services are semantically compatible, along with references to DITAG-generated translation artefacts. The consumer invokes both services:

- Retrieves Sandia payload and applies the Sandia-to-consumer translation mapping
- Retrieves AVL payload and applies the AVL-to-consumer translation mapping
- Merges harmonised data from both sources into a unified dataset for analysis

Critically, the consumer does not need to know the internal structure of either provider's payload. Translation logic is derived automatically from HASCO/Repo semantic descriptions, enabling zero-configuration integration. Furthermore, if AVL or Sandia update their payload schemas, only the corresponding SDD in HASCO/Repo needs to be updated; DITAG will automatically derive new translation logic without requiring consumer code changes.

Semantic Traceability and Provenance

Each harmonised value retains provenance information linking it to:

- The originating service (Sandia or AVL)
- The original variable name and structure
- The semantic transformation applied
- The SDD version used for translation

This enables downstream consumers to audit harmonisation decisions and verify that semantic transformations preserve the intended meaning. SPARQL queries against HASCO/Repo can retrieve the complete semantic lineage of any harmonised value.

4.1.6.9 Implications for Cross-Laboratory Integration

This scenario demonstrates that the combination of HASCO/Repo, DITAG, and Eclipse Arrowhead orchestration enables:

- Automated discovery of semantically compatible services across organisational boundaries
- Dynamic harmonisation of heterogeneous payloads without manual mapping
- Zero-configuration integration for consumers working with standardised semantic schemas
- Scalable interoperability that accommodates schema evolution without redeploying services
- Provenance-aware data fusion supporting auditable cross-laboratory analysis

By using real battery testing datasets from AVL and Sandia to simulate Arrowhead services, we demonstrate not only the technical feasibility but also the practical value of semantic orchestration in industrial and scientific ecosystems.

4.1.7 Ontology-based Translation in Business Processes (Proposal)

An ontology-based data model translation to improve the interoperability of information exchange processes for the process industry use case needs. This connects to the work in WP9, defining information exchange processes and further implementing them as executable, event-driven choreographies. For the ontology-based translation, the purpose is to develop an approach to identify mapping needs using super ontology concepts, to eventually invoke translations between different data modelling languages as part of these business processes.

Previously, ontology-based translation has been investigated to determine how it could be invoked as part of executable business processes in which technical information exchange takes place between partners of a complex value network. The conceptual approach has been developed primarily for the use case needs stemming from Task 9.3, Technical information exchange in process industry. In the use case, different business entities exchange information in processes that typically take days or even weeks, involving human activities. However, for this, the processes need to be digitalised, i.e. standardised in which order agreed tasks take place. This is illustrated in Figure 30, where for the technical information exchange needs, ontology-based translations have been identified as one of the necessary translation methods.

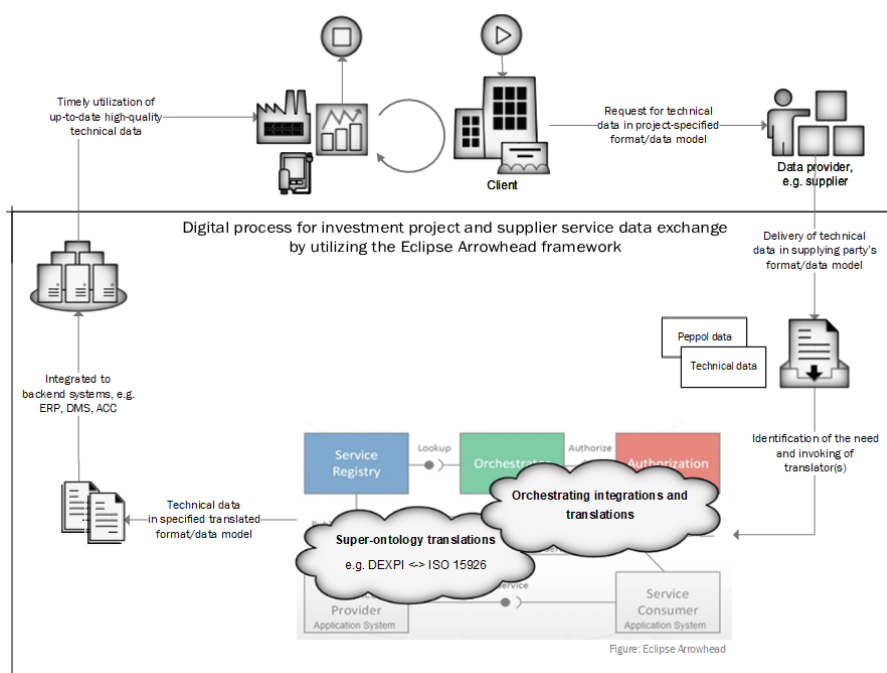


Figure 30: Ontology-based translation in business processes

In the work in WP9, defining interoperable information exchange, business process modelling is proposed as the method to define the interaction sequences. Secondly, as the business partners and thus also information systems change, dynamicity is needed for system integrations as the value network evolves and business partners change. Here, the Arrowhead framework plays an important role, providing available compatible services, but also translation adapters to be found and injected. To improve interoperability, an adapter architecture to inject translations into the business processes has been designed, and its implementation in BPMN 2.0 is presented in Figure 31.

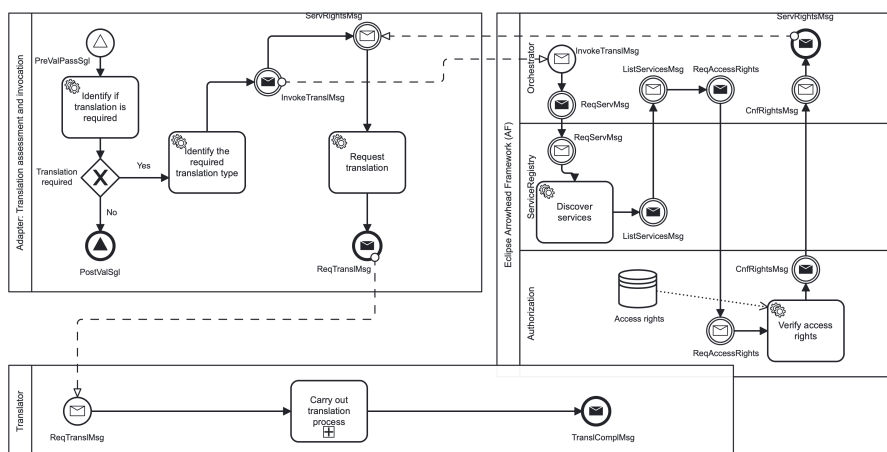


Figure 31: Business Process Model and Notation

The concept of identifying interoperability mismatch and injecting adapters has now also been identified by the Arrowhead framework and the translator service approach that has recently been developed. The implementation of the work presented, connected to process industry information exchange, continues to be implemented and explored as part of the use case demonstrators in WP9.

The translation approaches presented connect closely to WP9 translation needs in many of its tasks. The process industry use cases typically rely on different standard data models, many based on ontology definitions,

and approaches using both ontology mapping as well as AI-based approaches are relevant for the use cases. For the work in WP4, the use cases provide a challenging environment with a lot of different standards that have to be supported, thus providing relevant context and requirements for the developments.

4.1.8 Arrowhead fPVN Ontology–Driven Interpretation (Prototype)

4.1.8.1 Integration for Environmental Monitoring Data

In previous Deliverables, Task 4.1 emphasised the use of upper ontologies to systematically align heterogeneous concepts originating from different modelling languages such as ISO 10303, ISO 15926, IEC 81346, and S5000F.

In D4.3, this approach is extended to integrate environmental monitoring ontologies relevant to BEIA's dataset (water quality, air quality, weather conditions). The goal is to create a semantic bridge between:

- sensor-specific descriptors (e.g., "OM_E", "SA_C", "TN_D")
- domain concepts (e.g., *DissolvedOxygen*, *Salinity*, *Turbidity*)
- standards models (e.g., ISO 15926 property classes)
- the Arrowhead super ontology

4.1.8.2 Example Ontology Mappings

Table 4: Example Ontology Mappings.

Sensor Code	Meaning (from D4.2)	Ontology Concept	Target Standard Class
OM_E	Dissolved Oxygen (mg/L)	WaterQuality:DissolvedOxygen	ISO 15926 PhysicalQuantity
SA_C	Salinity (ppt)	WaterQuality:Salinity	ISO 10303 MeasureWithUnit
TC1_E	Temperature (°C)	Environmental:Temperature	IEC 81346 Property
PH_B	pH Level	WaterChemistry:pH	ISO 15926 PropertyType

4.1.8.3 Ontology Alignment for Environmental Variables

```

1 import re
2
3 def normalize_for_ontology(name: str) -> str:
4     return re.sub(r'[^a-zA-Z0-9]+', '', name).lower()
5
6 ENVIRONMENT_ONTOLOGY = {
7     "om_e": "WaterQuality:DissolvedOxygen",
8     "sa_c": "WaterQuality:Salinity",
9     "tcl_e": "Environmental:Temperature",
10    "ph_b": "WaterChemistry:pH",
11    "rx_b": "WaterChemistry:Redox",
12 }
13
14 def map_to_ontology(variable_code):
15     canonical = normalize_for_ontology(variable_code)
16     return ENVIRONMENT_ONTOLOGY.get(canonical, "UnknownConcept")

```

This ontology mapping forms the semantic input layer for downstream ML/AI translation tasks (Section 4.2.11).

4.2 AI-Based Translation

The following section outlines the efforts and progress regarding the technical solutions at the 30-month mark of the project concerning AI-based translations. These efforts include the analysis of requirements, design, development, validation, and integrated translation microservices of the proposed solutions.

4.2.1 AI Agents (Support Activity)

AI agents are autonomous systems designed to perceive their environment, process information, and take actions to achieve specific goals. The agents combine perception, reasoning, and decision-making capabilities to interact with both digital and physical contexts. Unlike traditional software, which follows rigid, predefined instructions, AI agents are adaptive. It means that they can learn from data, experiences, or feedback, and refine their behaviour over time.

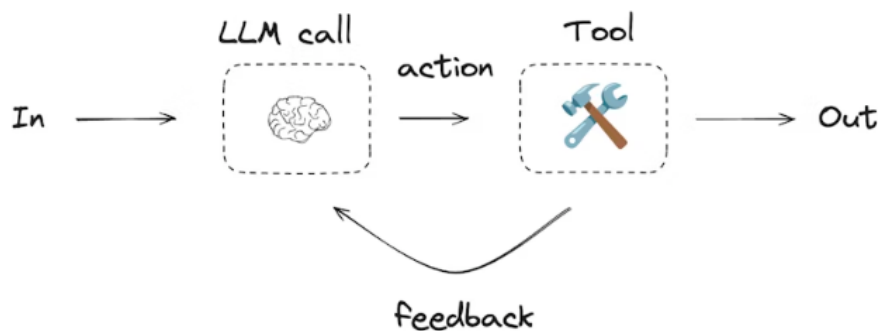


Figure 32: AI agent workflow [7]

An AI agent typically operates through a continuous cycle (see Figure 32) of observing its surroundings, reasoning about the state it perceives, and acting iteratively to achieve its goals. The significance of AI agents lies in their ability to handle complex tasks that are difficult, repetitive, or time-consuming for humans. For example, decision-support agents assist professionals in information mapping (e.g., translation between standards) by analysing vast amounts of information and suggesting solutions.

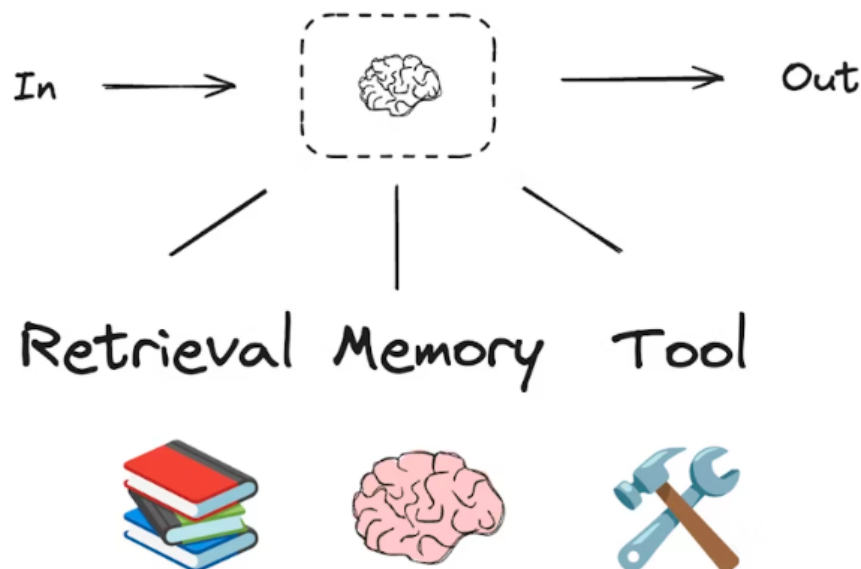


Figure 33: AI agent toolkit [7]

AI agents often rely on external tools to expand their capabilities beyond their built-in reasoning and decision-

making processes. Tools can be thought of as specialised interfaces or services that an agent can call when it needs information or when it must perform an action that exceeds its own computational limits. For example, an agent might query a searching tool to access up-to-date knowledge (e.g., retrieve information from a standard-related RAG database), use a software component to connect to databases, or manipulate structured data (See Figure 33). By integrating tools in this way, agents become more versatile, bridging the gap between their abstract reasoning abilities and concrete, task-specific actions.

From an architectural perspective, AI agents can be designed in various ways depending on the complexity of their intended role and environment. At the simplest level, they may operate as reactive systems, directly mapping inputs to outputs without maintaining memory or long-term planning. More complex architectures can apply memory modules for storing past interactions, or feedback loops for continuous learning and improvement.

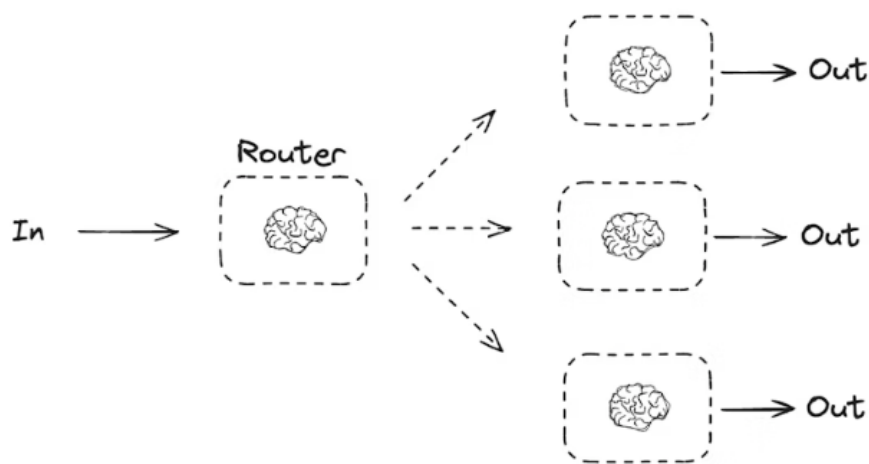


Figure 34: AI agent toolkit [7]

A tool can be another agent, or agents can collaborate with other agents (or humans as well), resulting in a multi-agent architecture to further extend the possibilities. For example, Figure 34 presents a multi-agent architecture.

In recent years, the concept of AI agents has become even more relevant with the rise of generative AI, which provides them with advanced communication and reasoning skills, enabling more natural and effective interaction. The integration of memory, planning, field-specific tools and long-term learning into agent architectures allows them to automate processes in an adaptive way. Their potential impact is vast, as they promise to extend human capabilities and transform industries

4.2.2 MCP – Model Context Protocol (Support Activity)

AI has rapidly evolved from isolated models to interconnected ecosystems involving multiple models, tools, and applications working together. As systems grow more complex, a major challenge has emerged: enabling different AI models, data sources, and tools to communicate effectively without creating custom integrations for each case. The Model Context Protocol (MCP) addresses this challenge by providing a standardised, open-source framework for seamless interoperability among AI systems, allowing AI applications to connect to external systems, tools, and workflows efficiently.

By introducing a standardised communication layer, MCP allows any model to interact with any tool that supports the protocol, eliminating the need for proprietary APIs or unique connectors. This standardisation simplifies development and encourages interoperability across the AI landscape.

Figure 35 presents the components of an MCP communication: Host, Client and Server. The Host is an AI-based application that applies one or more MCP clients. The MCP Client maintains a connection to an MCP server and provides context-related information to the Host. The MCP Server provides context (e.g., available REST-API resources, results of a resource call, etc.) to the MCP Clients.

MCP is a JSON-based protocol (JSON-RPC 2.0 message format), and applies a stateful communication between the clients and servers.

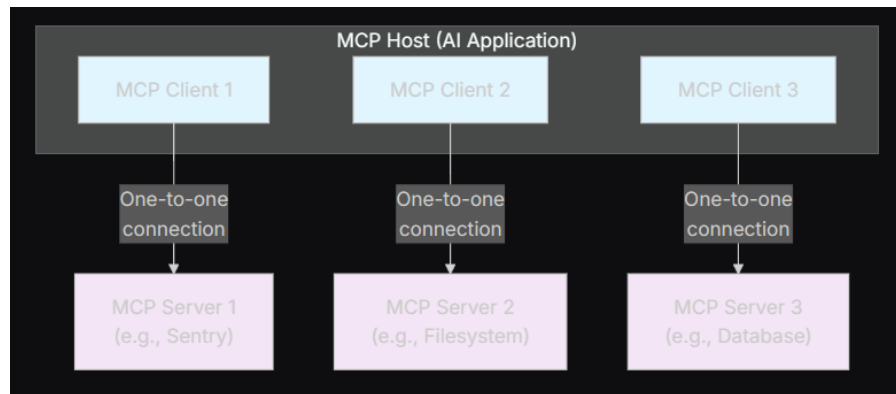


Figure 35: MPC Architecture [8]

As a typical workflow, an AI application applies MCP to connect with multiple external systems. For example, it can retrieve customer data from a CRM, summarise files from a cloud drive, and log conversation summaries back into a support database — all through MCP's unified communication format that removes the need for custom-coded integrations.

For more information, please refer to the appendix: [Appendix/1_Model_Context_Protocol.pdf](#)

4.2.3 Prompting Techniques (Support Activity)

To apply LLM-based decisions integrated into Python code or to retrieve relevant information from split document chunks, a task-specific, well-formed prompt should be used in each case. This subsection summarises the prompting techniques that were collected in the recent period, regarding document processing.

When we would like to apply an LLM call with a specific request (prompt), it can contain several parts (sub messages), based on the type of the message (role):

- system
- user
- assistant

The “system” message defines general rules and contexts, and how the LLM should behave regarding our question. E.g.: “You are an assistant who analyses text differences and maps them to CSV keys.”, “You search for data from a text fragment based on the keywords and descriptions provided.”

The “user” message defines precisely the tasks and the rules, often separated into more parts. The format of the “user” message should be in Markdown format, because LLMs can efficiently handle and understand the Markdown-formatted contents.

The “assistant” message is an example answer for the current “user” message. It is often used to show example question-answer pairs within the LLM call, so more “user” and “assistant” message pairs can be defined within one LLM call. It helps the LLM to understand the task.

The “user” message is the most relevant part of an LLM call because it contains all the relevant information to generate the answer. As previously mentioned, the Markdown format is a recommended format to work with. Each part of the “user” message should be split with a ### mark, e.g.,

```
1 ### Task ###
2
3 ### Important rules ###
4
5 ### Output rules ###
6
7 ### Synonyms ###
8
9 ### Key descriptions ###
10
11 ### Document subsection ##
```

We often have to work with `csv` files, or `key - description` files as a dictionary to retrieve information for keys based on the description. When the size of this description is below 10, the `csv` format is usable. However, when we have to work with a list of more than 10 elements, a JSON format is a better choice. Especially, when we have to add other extra information, or we have more columns in our `csv`.

The details of the task descriptions (and the rules as well) depend on the size of the models. When we apply smaller models (e.g., `gpt-4o-mini`), the descriptions of the task should be more detailed (even the process step-by-step). We also have to specify how the descriptions should be used to identify a relevant key.

Since the LLMs are pretrained models, some keywords could mean different contexts in different scenarios, depending on the model training data sets. To avoid mismatches because of the pretrained knowledge, synonyms should be defined for relevant keywords.

The size of the text (document part) should depend on the size of the applied LLM's context window. In general, we should count on 30% of the maximum context window size. This is because LLMs mainly consider the first 30% of the maximal context window.

LLM could decide between processing steps. As an example, we applied a router function within the mapping table creation procedure (see Section 4.2.2). The router had to select the relevant 1st layer IPC 2581 XML Element based on the description. The prompt was structured based on the Markdown format and was forced to generate only one output category. The output category was the return value of a Python function, which could be further used in the source code. The example prompt is:

```
1  "system": "You are a router agent responsible for selecting the correct category based on the description."
2
3  "user":
4
5  ### Description ###
6
7  Electronic Component's Family (e.g., Resistors, Capacitors, Diodes, Piezo-Electric Devices, Inductors, Microcircuits, Transistors)
8
9  ### Available Categories ###
10
11  1. Avl - The AVL (Approved Vendor List) contains manufacturer and supplier approval data for parts and materials used in
12     production. Each entry typically associates a part number with one or more authorized vendors, including manufacturer names,
13     part numbers, status (e.g., approved, qualified, obsolete), and sourcing restrictions.
14  2. Bom - The BOM (Bill of Materials) defines the hierarchical structure of a product - listing all assemblies, subassemblies, and
15     components required to manufacture or build it. Each item may include quantity, reference designators, sourcing info, and
16     version or revision details.
17  3. Content - Content encompasses all descriptive, textual, and document-based information related to parts, assemblies, and
18     processes. It includes metadata (e.g., part descriptions, categories, lifecycle states) and associated digital assets such as
19     datasheets, drawings, and specifications.
20  4. LogisticHeader - The LogisticHeader contains high-level logistics and transaction metadata used to manage the physical or
21     digital flow of goods and data. It typically includes shipment identifiers, routing IDs, source and destination locations,
22     carrier information, shipment dates, and package counts.
23  5. HistoryRecord - The HistoryRecord logs changes, transactions, or routing events applied to data objects such as BOMs, AVL
24     entries, or ECAD integrations. Each record typically captures timestamp, user or system identifier, action type, and data deltas
25     .
26  6. Ecad - ECAD data includes electronic design information from PCB layout and schematic tools such as Altium, Cadence, or Mentor.
27     It may contain component symbols, footprints, netlists, reference designators, and layout constraints.
28
29  ### Output Rules ###
30
31  Respond should contain only the category name that is relevant.
32  You must choose one and only one category!
33  The response must be one of the six categories above, and it is case-sensitive.
```

4.2.4 AI-based Translation System (Prototype)

Task 4.2 continued the work on the AI-based translation topic. After re-examining the steps and internal milestones regarding the AI-based translations, we still came to the previous conclusion: a precise mapping table is a key part of the translation process. This is because we need a conceptual correspondence on how to represent the value of model #1 regarding model #2. The connections, hierarchies, and domain-specific representations have to be defined, and this step can be effectively supported by AI methods. This is because LLMs can be applied to select categories based on human-readable descriptions, can search context-specific information, and can offer element or attribute pairs regarding two independent standards.

Figure 36 presents the architecture of the implemented translation system. The implemented methods are related to UC1.7, as an example application.

To automatically generate a mapping table between two data models, the following steps should be performed:

- The key parameters or elements of the source data model have to be defined with a precise, human-readable description. This description will be used to find relevant pairs regarding the destination data

Multi-RAG Architecture to create mapping definitions

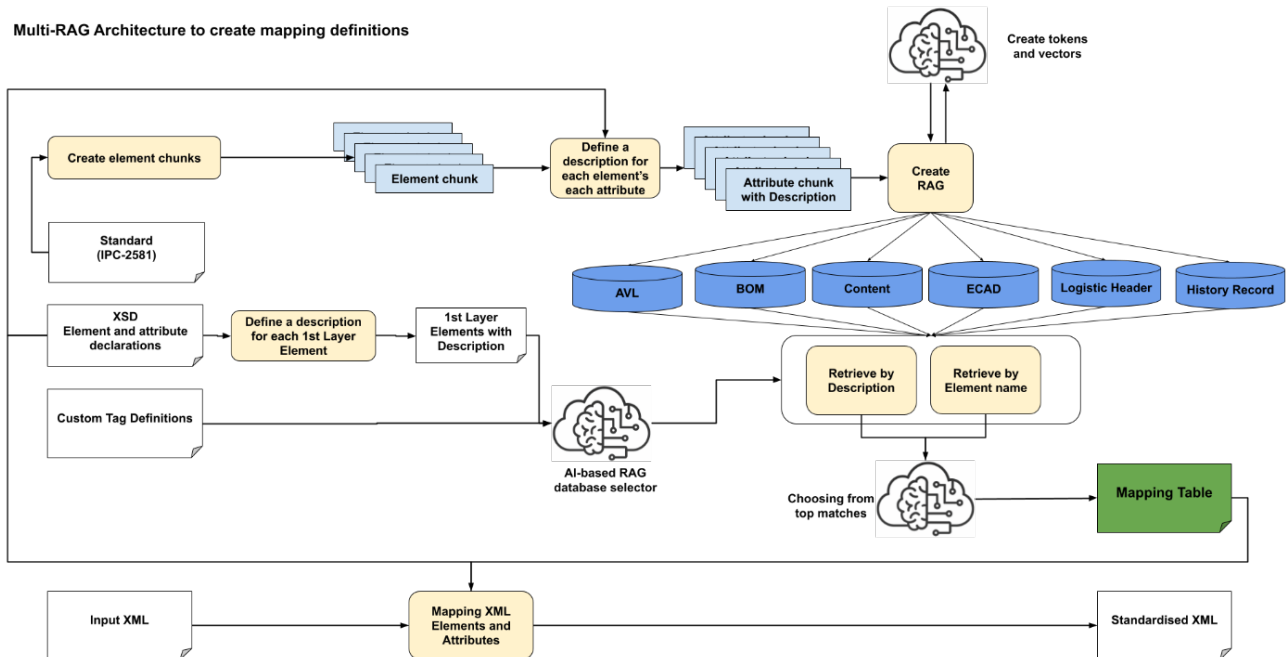


Figure 36: Architecture of the AI-based translation system

model.

- The document or standard related to the destination model should be converted to Markdown format, because LLM methods can effectively work with this format. As a general rule, each subsection or table should be split into different text chunks for further processing. These text chunks will provide the domain-specific knowledge regarding the destination model.
- When the destination data model is related to an XML format, each attribute should be defined independently with an element-specific description.
- To create a domain-specific knowledge database, one or more RAG (Retrieval-Augmented Generation) vector stores should be created from the attribute descriptions of the destination data model.
- In the previous deliverable, the duplications in the mapping table were allowed. After further examining the Mapping Table contents, we came to the conclusion that duplication shouldn't be allowed. To support this decision, Task 4.2 changed the format of the mapping table, and instead of storing the element name and the attribute name, the whole graph is represented. The reason is that there are element-attribute pairs which appear in different contexts within the destination model.
- To support the more efficient mapping definition and avoid destination element duplication in the table, we concluded that multiple parallel RAG vector stores should be created based on the root element categories. These categories were chosen based on the 1st level elements of the destination model's XML tree.
- To find a pair between the source and destination models' elements and attributes, we applied two retrieval steps: 1) a description-based retrieval, and 2) a name-based retrieval. The reason for this two-step method is that sometimes a name-based retrieval can achieve better accuracy, because models can apply similar names. Since the RAG method supports similarity search as well, the results will be relevant.
- To select between the relevant top matches, a final LLM call was applied.
- As a final translation step, applying the previously created mapping table, values were translated to the destination data model, based on the requirements of the destination data model's XSD file.

It should be noted that there are parameters, elements or attributes that couldn't be handled by the destination model. These parameters should also be recognised during the mapping table generation process.

Figure 36 presents that we identified 4 main steps where AI methods could be applied effectively:

- Domain-specific vector store creation.
- Routing based on the destination model's 1st layer elements.
- Retrieving context-relevant element and attribute names.
- Selecting and pairing the source and destination element and attribute names based on the top similar matches.

The following subsection presents the results of the above-mentioned method, and also examples regarding UC1.7 data samples.

The source code is available on github: https://github.com/Aitia-IIOT/ah-ai-translation-poc/tree/master/IPC_translate.

4.2.5 Integration with the Arrowhead 5.0 Microservice Paradigm (Prototype)

Task 4.2 has integrated the EC – IPC2581 translation (see Section 5.2.2) solution into the Arrowhead 5.1 framework. An overview of the integration can be seen in Figure 37. The integrated system was demonstrated during the fPVN Deep Tech Lisbon Event 2025.

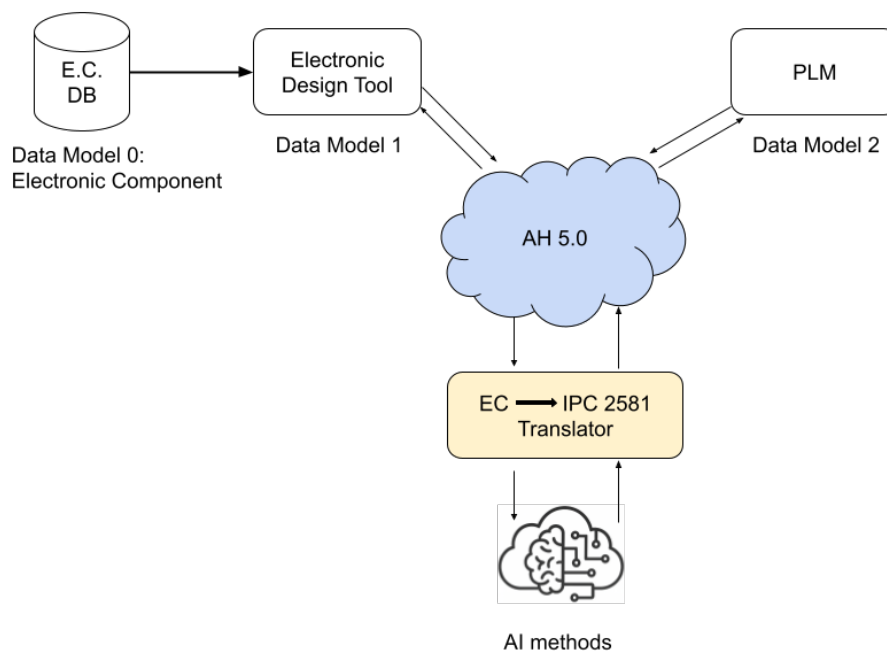


Figure 37: Integrated EC – IPC 2581 translation with the Arrowhead 5.1 framework.

The details of the integration steps are available as an Appendix, in:
2_AH5_Data_Model_Translation_Integration.pdf.

4.2.6 Measurable Progress of the AI-based Translation System (Support Activity)

During the translation process, we can distinguish the following steps:

1. Read the target standard and understand each element and each attribute:

- Review the target standard and analyse each element and attribute to understand its meaning, structure, and constraints defined in the schema definition (e.g., in XSD), to establish the target model's logic.

- Automation possibilities: Partially automatable, requires human understanding to define how the knowledge base can be created, and what the structure of the documents is.
- AI applicability: Partial applicability, AI can summarise sections, subsections, or extend the element descriptions to support information retrieval phases.

2. Collect all possible target attributes: formal declaration and textual description:

- Extract all element and attribute definitions and their descriptions from the target documents (e.g., standard pdf, white paper, XSD, etc.) into a structured list. This step grants a reference knowledge base for the mapping table definition phase.
- Automation possibilities: Can be fully automated, e.g., a script can parse the XSD file to extract all elements and attributes.
- AI applicability: Can be applied. AI can help to define, collect and enrich the descriptions for each element or attribute.

3. Read the source standard and understand each element and each attribute:

- Same as 1), but with the source data set.

4. Collect all possible source attributes: formal declaration and textual description:

- Same as 2), but with the source data set.

5. Create mapping for each target attribute, find all possible source attributes (Iterative work, required time: Target attributes x Source attributes):

- Establish potential mappings between target and source attributes based on semantics, datatype compatibility, and naming similarities. This often requires iterative work. The required processing time is related to the number of target attributes and source attributes.
- Automation possibilities: Can be automated with iterative steps.
- AI applicability: Can be applied. AI can propose semantic mappings, similarity search, and select the best match between multiple matches:
 - (a) **If there are multiple mapping candidates for an attribute, pick one:**
 - Resolve conflicts by selecting the most appropriate source attribute when multiple potential mappings exist. The decision is often context-dependent.
 - AI can analyse semantic similarity and context-specific matching.
 - (b) **Cleaning and correction:**
 - Review and correct inconsistencies or errors in the mapping table to ensure accuracy and consistency.
 - It can be partially automatable with validation rules that can be applied programmatically. AI can detect anomalies or inconsistent mappings.

6. Identify source attributes which couldn't be mapped:

- The mapping table or other relevant document should list all source attributes that do not have a corresponding target attribute.
- Automation possibilities: It can be derived from the mapping table, or the result of the 5.2 cleaning and correction step.

7. Mapping Table is ready

8. Identify the mandatory elements, attributes and default values which should also be added:

- Detect elements or attributes required by the target schema that are missing in the source data and assign default or constant values.
- Automation possibilities: Partially automatable, schema definitions (e.g., XSD) can define mandatory fields, attributes and structures (element embeddings).

- AI applicability: AI can offer suitable defaults based on the context.

9. Map all records based on the predefined rules (Number of records x Number of attributes per record):

- Perform the actual data transformation from the source data model to the target data model according to the mapping table and transformation rules.
- Automation possibilities: Fully automatable. Transformation can be scripted or executed with tools.
- AI applicability: not relevant.

10. Validate the output XML file using the XSD file:

- Check the transformed XML files against the target data model to ensure structural and semantic compliance.
- Automation possibilities: Fully automatable. Schema validation tools can be applied automatically.
- AI applicability: not relevant, AI can examine errors and suggest fixes.

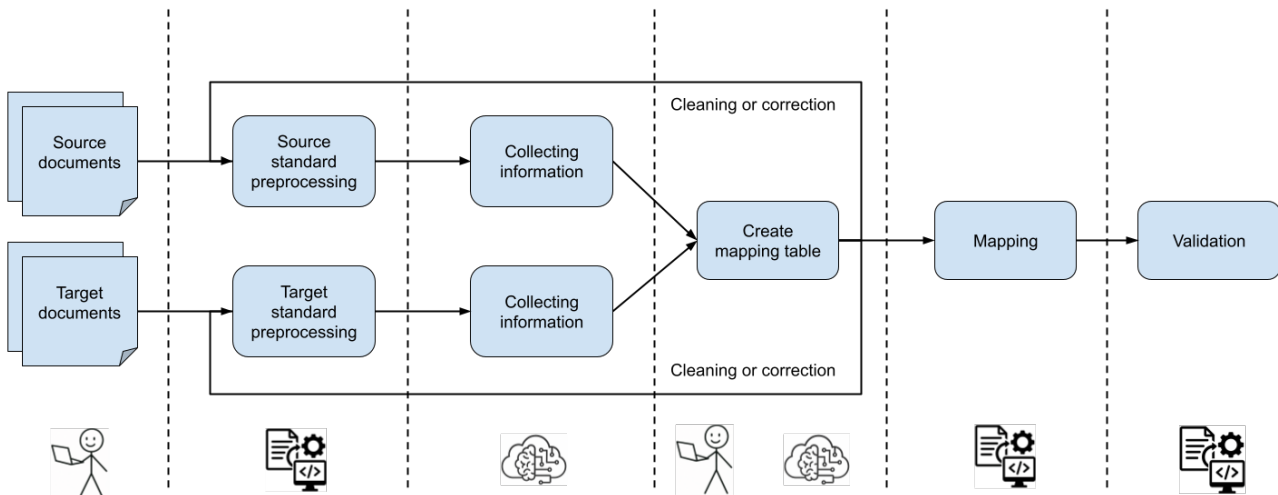


Figure 38: High-level steps of an AI-based translation process.

Figure 38 presents the high-level steps of a translation process, indicating automatable steps. The following automation possibilities can be distinguished:

- Automation with general algorithms and scripts:
 - Document preprocessing (Standards, white papers, etc).
 - Chunk creation (splitting documents automatically based on sections, subsections, topics, etc).
 - XML → XML conversion based on a mapping table that contains the target xml graph.
 - Validation based on XSD or custom rules.
- Automation with AI methods:
 - Knowledge base creation from dictionaries, document parts (e.g., from subsections, from tables, etc).
 - Information retrieval, top matches based on descriptions.
 - Decisions to apply multiple knowledge bases.
 - Decisions from multiple matches.
 - Mapping table generation.

- Human in-the-loop:
 - Collecting the source and target documents.
 - Investigation of the mapping table, optional corrections.

For the automated steps, the following manual cleaning/correction/post-processing is needed:

- Dictionary enrichment, which contains the attribute – description pairs.
- Mapping table cleaning, dropping duplications.

What is the overall speedup when using automated translation vs manual translation?

- Speed up the information retrieval:
 - Seconds for the AI.
 - Minutes or even hours for a human, depending on the standards.
- Speed up for creating the mapping table:
 - Around 2 minutes for the AI.
 - More days for a human (3-6 days, depending on the standards).

To measure the consistency of a mapping table generation procedure, Task 4.2 has generated the EC – IPC mapping table 10 times. The measured consistency was 95%.

Definition of translation quality in percentage:

- Currently, we are mapping from the EC data model to the IPC standard. As a test, we also implemented the mapping table generation the other way, from the IPC standard to the EC data model. The accuracy of the round-trip conversion could be a measure of the quality of the translation.
- Calculating with this algorithm, the currently achieved quality is 60%.

4.2.7 AI-translation Evaluation Pipeline (Support Activity)

Reference data: In order to validate the automated translation process and assess its quality, a reference baseline translation needs to be established. This is a set of source—target mappings, which are verified and known to be correct. The automated process is assessed by comparing its output with the reference output. The automated process is refined until it matches the desired reference output.

Round-trip mapping: If possible, target—source reference mappings can also be created. This allows round-trip verification, when the data is converted from the source representation to the target, and then back to the source data format. One expects that such a round-trip conversion restores the original data. The verification can be executed in the target—source—target direction as well, expecting that the data is identical in the original target data format, too.

Syntactic validation: The mapped data is validated at multiple levels. The XML data files are validated using XML Schema Definition (XSD), which ensures that each XML element and attribute value conforms to a specific data type and format. The validation is run on both the source and target data.

If the data formats change (by adding, deleting, or modifying the type of an element or attribute), the XSD files need to be updated, and the validations need to be re-run.

Semantic validation: Semantic validation ensures that the data values are consistent in relation to each other. This includes checking that a value is plausible in light of other values, or if there are no orphan or invalid references among data elements.

```
<xsd:complexType name="IPC-2581Type">
  <xsd:sequence>
    <xsd:element ref="Content"/>
    <xsd:element ref="LogisticHeader"/>
    <xsd:element ref="HistoryRecord"/>
    <xsd:element ref="Bom" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Ecad"/>
    <xsd:element ref="Avl" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="revision" type="xsd:string" use="required"/>
</xsd:complexType>
```

Figure 39: Exerp from IPC-2581C.xsd

4.2.7.1 Creation of IPC-2581 reference mapping

In order to produce a reference mapping from the UC 7.1 data formats to the IPC-2581 standard, we created a program [9], which implements the reference mapping.

The data mapper program is based on the `xsddata` Python package, which takes an XML/XSD file as input (See Figure 39), and outputs a Python file containing `dataclasses` which correspond to the XML elements and attributes (See Listing 1). An actual example can be seen in Listing 2 where the mapping is implemented as small hard-coded rules.

```
1 @dataclass
2 class Ipc2581CType:
3     class Meta:
4         name = "IPC-2581Type"
5
6     content: Optional[Content] = field(
7         default_factory=Content,
8         metadata={
9             "name": "Content",
10            "type": "Element",
11            "namespace": "http://webstds.ipc.org/2581",
12            "required": True,
13        },
14    )
15     logistic_header: Optional[LogisticHeader] = field(
16         default_factory=LogisticHeader,
17         metadata={
18             "name": "LogisticHeader",
19             "type": "Element",
20             "namespace": "http://webstds.ipc.org/2581",
21             "required": True,
22        },
23    )
24     history_record: Optional[HistoryRecord] = field(
25         default_factory=HistoryRecord,
26         metadata={
27             "name": "HistoryRecord",
28             "type": "Element",
29             "namespace": "http://webstds.ipc.org/2581",
30             "required": True,
31        },
32    )
33     bom: list[Bom] = field(
34         default_factory=list,
35         metadata={
36             "name": "Bom",
37             "type": "Element",
38             "namespace": "http://webstds.ipc.org/2581",
39        },
40    )
41     ecad: Optional[Ecad] = field(
42         default_factory=Ecad,
43         metadata={
44             "name": "Ecad",
45             "type": "Element",
46             "namespace": "http://webstds.ipc.org/2581",
47             "required": True,
48        },
49    )
```

```

49 )
50     avl: Optional[Avl] = field(
51         default=None,
52         metadata={
53             "name": "Avl",
54             "type": "Element",
55             "namespace": "http://webstds.ipc.org/2581",
56         },
57     )
58     revision: Optional[str] = field(
59         default="C",
60         metadata={
61             "type": "Attribute",
62             "required": True,
63         },
64     )

```

Listing 1: Excerpt from ipc2581c.py.

```

1 def map_bom_item(root: Ipc2581C, src_record: plm.Row) -> BomItem:
2     dst = BomItem()
3     dst.characteristics = map_characteristics(root, src_record) if src_record else None
4     dst.oemdesign_number_ref = src_record.ein
5     # dst.quantity = src.quantity
6     # dst.pin_count = src.pin_count
7     # dst.category = src.category
8     # dst.internal_part_number = src.internal_part_number
9     dst.description = src_record.description
10    return dst
11
12 def map_bom(root: Ipc2581C, src: plm.SAE_PLM) -> Bom:
13     dst = Bom()
14     dst.bom_item = [map_bom_item(root, bi) for bi in src.row] if src else []
15     # dst.name = src_record.name
16    return dst

```

Listing 2: Excerpt from the map_sae_plm_to_ipc2581c.py file containing the mapping rules.

The mapping rules convert a SAE format XML data file into an IPC-2581 XML file. This file serves as a reference mapping to evaluate the automated translation program.

The mapping rules themselves can also be used as training data for automatically creating the mapping rules.

4.2.8 Quantitative Evaluation of a Semantic RAG-Driven XML Translation (Support Activity)

The purpose of this evaluation was to systematically assess the quality and reliability of an automated XML-to-XML translation pipeline designed to map variables from a proprietary, inconsistently formatted source schema into the IPC-2581 industry standard. Given the heterogeneity of the input data, the complexity of the IPC-2581 hierarchical model, and the fact that the translation process is driven by a combination of retrieval-augmented generation (RAG), semantic similarity search, and schema-guided XML reconstruction, traditional syntactic validation methods are insufficient for measuring correctness. Instead, a semantic accuracy metric was developed to determine whether each variable from the source data is placed into the semantically appropriate attribute or structural location in the output IPC-2581 document. To achieve comprehensive coverage, the evaluation was conducted over the dataset of 169 input/output XML file pairs from the IPC translate scripts¹, enabling both per-file and aggregate analysis of the translation pipeline's performance across a wide range of component descriptions and structural variability. The resulting analysis provides a quantitative foundation for understanding system behaviour, identifying gaps in the mapping table, and guiding future refinement of the translation modules.

4.2.8.1 Methodology for Calculating Semantic Translation Accuracy

The objective of the evaluation was to measure the semantic correctness of the translation between two heterogeneous data models:

- the source model represented by an IPC-like, non-standard XML with free-form tags, and
- the target model, an IPC-2581-compliant XML structure produced by the translation pipeline (RAG-based mapping + tree-guided XML embedding).

¹Available here: https://github.com/Aitia-IIOT/ah-ai-translation-poc/tree/master/IPC_translate/v1

Unlike syntactic comparison—which checks for literal tag equivalence—semantic evaluation has the role of determining whether each source variable was successfully placed into the correct semantic location according to the schema of the target standard. Because the input data is malformed (invalid XML tag names, inconsistent spacing, missing namespaces) and the output uses XSD-governed hierarchical paths with attributes rather than nested elements, a syntactic diff is not meaningful. Instead, a semantic validator was designed.

Variable Extraction from Source XML

Each `<row>` in the source XML represents a logical data record. Due to malformed syntax (e.g., tag names containing spaces such as `<Code Eng COM>`), standard parsers cannot be used. Instead, a custom regular expression that extracts:

- full opening tag name
- full closing tag name
- associated content

was developed.

The extracted tag names are canonicalised using the following rules:

- convert to lowercase
- remove all non-alphanumeric characters
- collapse the result into a canonical variable name

For example:

Table 5: Example variable extraction from the XML source

Raw XML Tag	Canonical
EPPL Family	epplfamily
Quality type	qualitytype
Code Eng COM	codeengcom
Footprint1	footprint1

This produces the set of input semantic variables.

Mapping Table (Ground Truth for the Evaluation)

The mapping table `mapping.csv` contains manually curated or RAG-generated variable mappings:

```
1 EC_tag -> IPC_element
```

Each mapping describes the intended placement of a source variable in the IPC-2581 data model. The `IPC_element` field encodes the full target path, for example:

```
1 IPC-2581/Content/DictionaryLineDesc.units
2 IPC-2581/Content/DictionaryStandard/EntryStandard/Circle.diameter
```

Only the final semantic component (e.g., `units`, `diameter`, `id`, `mirror`) is used for accuracy evaluation, because that corresponds to the actual XML attribute name present in the translated output. This field is also canonicalised identically to the source variable names.

Variable Extraction from the Translated IPC-2581 XML

The translated XML uses attributes rather than elements to represent most values, as required by the IPC-2581 schema. Therefore, both element names and attribute names are extracted:

```
1 <Circle diameter="0.3pF"> -> variable: "diameter"
2 <Xform mirror="COM"> -> variable: "mirror"
3 <Package type="EEE-01"> -> variable: "type"
```

Leaf elements (those without children) are extracted as well, though most values appear as attributes in this standard. All extracted variables are canonicalised using the same procedure as above. The result is the set of actual output semantic variables.

Semantic Matching and Accuracy Computation

For each variable in the source `<row>`:

1. Canonicalised source variable name $\rightarrow$ `src_var`
2. Validator looks up its expected target variable:

```
1 expected_var = mapping_table[src_var]
```

3. Semantic match occurs if:

```
1 expected_var ∈ output_vars
```

4. Accuracy for a row =

```
1 # correctly mapped variables) / (total variables in that row)
```

Per-variable accuracy is computed as:

```
1 for each variable v:
2     accuracy(v) = (number of rows where v was correctly placed) / (number of rows where v appears)
```

4.2.8.2 Implementation Overview

A dedicated Python validator was implemented to operationalise the methodology. It consists of four stages:

Canonicalisation Layer

A function:

```
1 def canonicalize_varname(name):
2     ...
```

normalises all variable names by:

- converting to lowercase
- stripping whitespace
- removing all characters except `a-z` and `0-9`

The aim is to ensure uniform comparison between:

- malformed source tags
- mapping table entries
- XML tag names
- XML attribute names

Source Variable Extraction

A custom regex:

```
1 < <full-tag-name> > content </ <full-tag-name> >
```

extracts variable names even with illegal syntax, whitespace, or multi-word identifiers. This is necessary because:

- `<Code Eng COM>` is invalid XML
- standard parsers break
- the validator must reconstruct semantic intent regardless of syntax

Target Variable Extraction

The target XML is well-formed; therefore, an ElementTree parser is used. For every element:

- attributes are extracted (most semantic variables are attributes)
- leaf-element names are optionally extracted

For example:

```
1 <DictionaryLineDesc units="01 CAPACITORS"/>
```

produces the variable:

```
1 units
```

after canonicalization.

Semantic Accuracy Computation

The validator:

- loads `mapping.csv`
- maps source variables → expected IPC attributes
- compares expected vs. actual occurrences
- aggregates results per row and overall
- generates:
 - bar charts
 - heatmaps
 - detailed per-variable correctness

The computation is fully deterministic and independent of formatting or whitespace.

4.2.8.3 Interpretation of Results

The computed **overall semantic accuracy was 0.533 (53.3%)**. This value represents the proportion of source variables in the `<row>` that were successfully translated to their correct semantic attribute in the IPC-2581 output.

Variables Successfully Translated

A substantial number of variables—especially high-confidence fields:

```
1 EPPL Family
2 EPPL Group
3 Quality type
4 Description
5 SPN
6 Component Number
7 Commercial Part Number
8 Engineering Package
```

```

9  Generic specification
10 Detail specification
11 Manufacturer
12 Link files
13 Library Ref
14 Value
15 Tol
16 Footprint1
17 Application

```

were correctly mapped.

This indicates to us that the RAG-based variable selection and the tree-guided XML assembly logic successfully place many fields in the correct IPC-2581 locations.

Variables Not Successfully Translated

Some variables consistently failed to map:

- “Code Old ...” fields
- “PPL EBOM / MBOM”
- “Quality Level / Status”
- “Moisture Sensitivity Level”
- “SIM SubKind”
- “Volt”
- “Footprint2 / Footprint3”

Suspected reasons for this are:

1. No mapping exists for these fields in `mapping.csv`.
2. The output XML does not instantiate the required attribute (e.g., `qualitylevel` does not appear anywhere).
3. Some IPC-2581 elements are optional, and the translator correctly omitted them when no target variable is available.
4. The LLM filtering model may determine that some source variables have no safe semantic equivalent in the IPC schema.
5. Some IPC-2581 substructures are deeply nested, making placement more challenging.

Achieved Results

Figure 40 presents the per-variable semantic accuracy obtained across all 169 translated files. The results clearly distinguish three categories of system behaviour. First, a substantial subset of variables achieves perfect accuracy, reflecting stable and unambiguous mappings that the translation pipeline consistently resolves (e.g., EPPL Family, Description, Component Number, Value, Tol, and multiple footprint fields). Second, a smaller group of variables shows partial or inconsistent performance, generally corresponding to attributes that appear less frequently in the dataset or for which contextual cues vary across records. Third, several variables exhibit zero accuracy; these are typically fields for which no semantic mapping is currently defined, or for which the IPC-2581 schema offers no direct structural counterpart.

The semantic accuracy score of approximately 53% should be interpreted as a realistic baseline for the current stage of system development. This value does not reflect a failure of the translation pipeline; rather, it captures the natural limitations imposed by several factors in the current configuration. First, the mapping table used for this test is still evolving and does not yet define target correspondences for a number of fields present in the input data. Second, the IPC-2581 schema itself contains structures and attributes that are only partially covered by the current translation logic, creating unavoidable blind spots in the output.

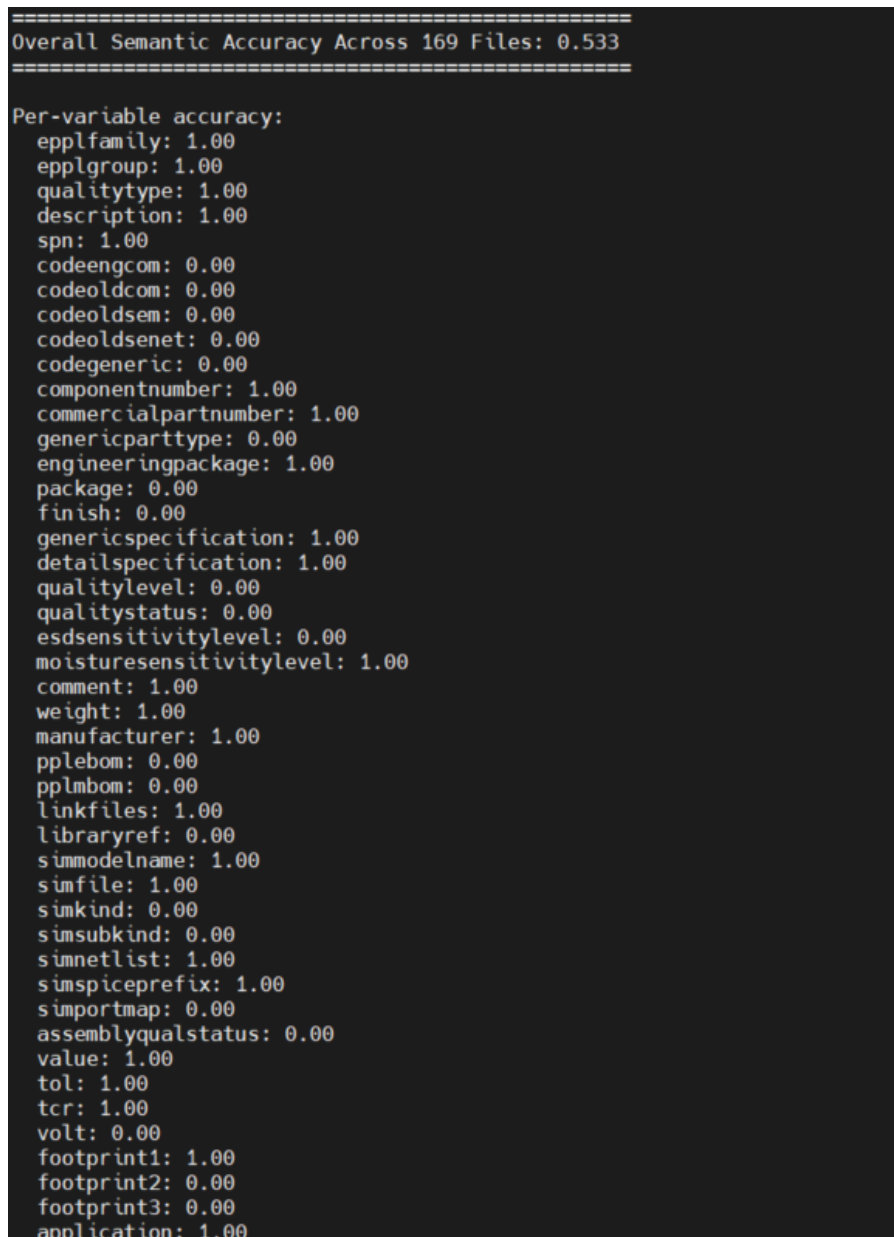


Figure 40: Per-variable semantic accuracy across 169 translated XML files

Additionally, certain source fields have no direct or unambiguous semantic equivalent within IPC-2581, leading the system—correctly—to leave them unmapped rather than risk forcing an inappropriate or misleading correspondence. The use of a retrieval-augmented generation mechanism introduces another dimension: although this approach significantly reduces hallucinations, the model may still default to fallback behaviours when contextual information is insufficient or when it encounters variables with limited semantic anchoring in the target schema.

Taken together, a score around 0.53 should therefore be seen not as an anomaly but as a meaningful quantitative reflection of the system's current maturity. As the mapping table is expanded, the coverage of the schema improves, and the translation rules become more precise; this score is expected to rise. The present result provides a solid, interpretable baseline against which future iterations can be measured.

What the Accuracy does *not* Measure

It is also important to clarify what lies outside the scope of the current accuracy metric. The validator focuses

exclusively on whether each semantic variable from the input appears in the correct structural location—typically as an attribute—in the IPC-2581 output. It does not attempt to verify the correctness of the values assigned to these attributes, meaning that it does not check whether enumerations, numerical types, or unit constraints are respected according to the schema or domain rules.

The metric also does not assess path-level correctness beyond confirming that the expected attribute name appears somewhere in the output XML. Whether the attribute is embedded at the precise hierarchical depth prescribed by the IPC-2581 standard is not evaluated, nor does the validator check whether parent and sibling elements are arranged in the exact order required by the XSD.

Furthermore, this approach does not verify the formatting quality of the output XML—such as indentation, namespace declarations, or compliance with canonical serialisation practices—and it does not perform any form of round-trip validation to ensure that the translated XML could be losslessly converted back into the original representation. These aspects are intentionally treated as out of scope for this phase, as the primary focus is on measuring semantic placement accuracy rather than full schema conformance or bidirectional fidelity.

4.2.9 Next Steps - AI-based Translation

This approach sees the following tasks for the next period:

- Work together with WP7 on UC1_7, to finalise the mapping table contents and measure the translation quality. It needs human resources to validate the table components iteratively.
- Finalise the syntactic validation step based on the XSD file, and extend the generated XMLs with the mandatory elements.
- For further semantic validation of the IPC-2581-compatible XML files, we will use the Vu2581 visualisation software. Our goal is to open a translated XML without errors.
- To compare the translation results with the ontology-based solutions, we will work together with Task 4.1 and apply the DITAG tool to translate back to the original input. After that, we can investigate the accuracy.
- Extend and investigate the STEP (ISO 10303) standard to have a precise, validated mapping table to translate from the EC data model to STEP. STEP is also related to WP3, WP7 and WP9.

4.2.10 Dissemination

The work done by this approach has resulted in two published papers:

- [10]: Csaba Hegedűs, Ádám Tóth, Máté Bancsics, Pál Varga, “Multi-agentic Plan-and-solve Engine for Managing System of Systems via Natural Language”, 6th Workshop on Management for Industry 5.0 (MFI5.0), 2025, Honolulu, HI, USA
- [11]: Tamás Tóthfalusi, Zoltán Csiszár, Pál Varga, “TACO – A copilot with Generative AI Actions for Telecommunication Core Network Analysis”, IEEE/IFIP Network Operations and Management Symposium, 2025, Honolulu, HI, USA

4.2.11 ML/AI Translation for Environmental Sensor Data Streams (Prototype)

Following the objectives of Task 4.2 described in D4.2 (exploring ML algorithms for cross-model translation), D4.3 extends this work by applying ML/AI reasoning to real-time IoT data conforming to BEIA’s environmental monitoring pipeline (Libelium → Meshlium → InfluxDB → Grafana).

The AI model learns to:

- interpret sensor field names,
- infer semantic units,
- link them to standard-compliant attributes,
- translate them into structured models (e.g., ISO 15926 templates or IPC-style XML structures)

4.2.11.1 LLM-based Translation of Sensor Payloads

```

1 def ai_translate_sensor_record(record, ontology_mapper, llm):
2     """
3     record: dict {sensor_code: value}
4     ontology_mapper: maps sensor codes to ontology concepts
5     llm: local LLM interface
6     """
7     translated = {}
8
9     for code, value in record.items():
10         concept = ontology_mapper(code)
11
12         prompt = f"""
13         Sensor code: {code}
14         Ontology concept: {concept}
15         Raw value: {value}

```

Translate this into a standardised metadata object including unit, normalised name, and recommended schema location. Output JSON only.

```

1 result = llm(prompt)
2 translated[code] = result
3
4 return translated

```

This script produces structured, interpretable translations from raw sensor readings.

4.2.12 Percepta (Prototype)

While artificial intelligence (AI) models require large volumes of historical data for training, their deployment depends on real-time structured data. However, whereas training datasets are typically organised, accurate, and complete, real-world data often originates from multiple sources using different protocols and formats, and may be unreliable due to network failures, sensor errors, or other disruptions.

For AI models to operate and infer correctly, incoming data must be reliable – incorrect or inconsistent data can lead to poor inferences and compromise the entire system in which the model is embedded.

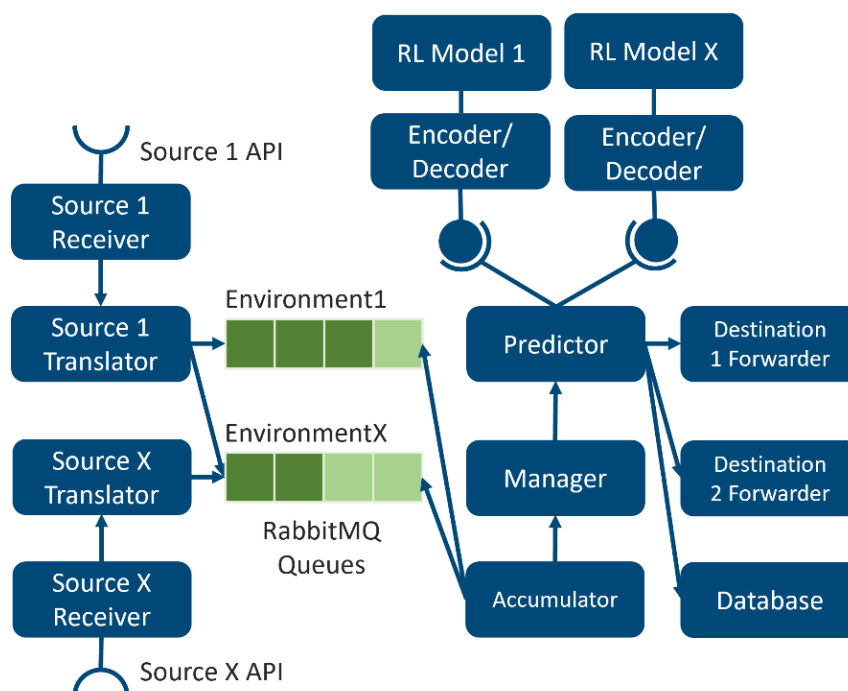


Figure 41: Percepta architecture

Percepta (Figure 41) emerges as a modular and extensible solution to this problem, acting as a bridge between the real world and the AI model. Its goal is to be modular enough to adapt to any data transmission protocol, data format, AI model, or use case.

To receive data, Percepta includes multiple receivers, each designed for a specific protocol. If a particular protocol is not natively supported, the system is designed to be open and easily extendable for seamless integration. Regarding data formats, Percepta introduces the concept of a Translator, a module that adapts to the format of incoming data and converts it into the system's standard representation.

Once data is received, Percepta provides several processing operations such as missing-data detection and replacement, and frequency harmonisation (for example, if data arrives every 5 minutes but the system operates on a 15-minute cycle, it adjusts accordingly). Furthermore, Percepta is extensible – users can integrate their own modules for data replacement, validation (e.g., correcting contextually inconsistent data), and harmonisation. In addition, Percepta supports the integration of any model that consumes the collected data, as well as modules for decision validation and concept drift detection.

Ultimately, the core of Percepta lies in its versatility, flexibility, and modularity. To improve understanding, a detailed description of each component that constitutes the Percepta architecture is presented below.

Receivers – The input components of the system are the Receivers. For each data source, there is a Receiver that adapts to how the assets' information is sent.

Translators – For each data source, there is also a Translator that adapts to the format of the received data, extracting only the relevant information for the algorithm. Whenever the Receiver receives/requests data, it forwards what it received to the Translator. The Translator, after extracting the relevant information and formatting the data coherently for the system, submits the altered data to an internal RabbitMQ queue.

Accumulator – The Accumulator abstracts itself from the data providers. The Accumulator listens to the queues of all environments, and whenever it receives a message, it forwards it to the Manager of that environment.

Manager – The Manager receives messages from the Accumulator and, at the end of each time interval (e.g., every 15 minutes, if that's the frequency at which the algorithm operates), it checks for missing data and handles any necessary substitutions. Once the dataset is complete, the Manager creates the final message containing all the environment data treated. Finally, it sends the structured data to the Predictor component of the environment.

Predictor – This component is responsible for forwarding the formatted data to the appropriate AI environment model. When the model returns its decisions, the component validates them with the fail-safe module to ensure they are correct and within safe operational limits. Once validated, the Predictor routes the decisions to the corresponding Forwarder component.

Forwarder – The Forwarder component is responsible for sending the AI decisions to the appropriate recipients.

It is also important to note that all data received by Percepta, along with the decisions made by the AI model, are stored in a database for future model retraining and analysis.

4.2.12.1 Autonomous Driving

The rise of autonomous driving is transforming the vehicle landscape into a cooperative and interconnected environment where multiple agents share data with each other. The development and testing of autonomous driving applications is very expensive, as it entails significant hardware costs and the cost of renting testing spaces, as well as the risks of breaking equipment due to failures during tests. Simulation allows us to better prototype or applications and test our applications, reducing room for errors before deployment in real-world tests.

In our work, we leverage Arrowhead within our simulation framework to provide dynamic service discovery, authorisation, and orchestration in multiple urban scenarios and understand its capabilities in such environments. The Service-oriented Architecture of Arrowhead allows us to build modular, scalable and interoperable Intelligent Transportation Systems (ITS) applications.

We developed two test scenarios with the intention of understanding Arrowhead's capabilities and limitations. The first one is a highway lane merge scenario, where autonomous vehicles that are trying to enter the highway will request nearby edges for cooperative perception services. The highway is equipped with sensors that allow the detection and tracking of vehicles crossing through the highway. The edge will request a local cloud equipped with Arrowhead for those services and authenticate the vehicle, then the vehicle will receive the list of services available in its area and choose the right service to perform the highway merge. This scenario

intends to understand how latencies in the Arrowhead cloud might affect the performance of cooperative driving applications.

Additionally, we developed another scenario that's situated in an urban city where the service we're testing this time is intersection assistance, where a sensorial infrastructure will provide additional perception data to vehicles circulating throughout the city, helping them do manoeuvres such as traversing intersections by providing them with information on the cars that are driving near. This scenario intends to test Arrowhead's capabilities in dealing with many vehicles sending requests to the network.

Our simulation architecture integrates both ROS 2 and Gazebo with the Omnet++ and Arrowhead Frameworks. Omnet++ is a network simulation tool, and we use it together with the libraries from the Artery framework, which allows us to simulate Vehicular Communications with protocols such as ITS-G5. Our Framework leverages ROS2 and the underlying DDS as the foundation for the interaction between the different tools that compose our application. The Omnet++ Interface is composed of an Omnet++ transmitter that creates a publisher, whose topics are subscribed to by the Omnet++ module, which then relays the information to the correct nodes within the Omnet++ simulation environment. The interface also has an Omnet++ receiver, which captures the packets from the Omnet++ simulation environment and publishes them into the respective ROS topics. The Arrowhead Interface is a module that sends and handles RESTful requests to the Arrowhead Framework from the nodes within our simulation environments. Within our network, Arrowhead allows for service discovery and authentication. As already mentioned, we intend to test the capabilities of Arrowhead, evaluating the impact service delays may have on the performance of cooperative driving tasks. In our network architecture, we decompose services into smaller services capable of interoperating together in many scenarios. In this case, we have Arrowhead Core Services, Service Registry, Orchestrator and Authorisation, along with some services that are going to be provided by Roadside Units (RSU).

By leveraging Arrowhead's service-oriented architecture, we enable dynamic service discovery, orchestration, and secure communication between distributed vehicular and infrastructure nodes. Our initial implementation focuses on validating the feasibility and responsiveness of Arrowhead services in two representative scenarios. Early results suggest that while Arrowhead offers promising mechanisms for structuring and managing ITS services, its performance in real-time vehicular contexts, especially under high-speed conditions, must be thoroughly assessed.

4.3 Model-Based Translation

The following section outlines the efforts and progress regarding the technical solutions at the 30-month mark of the project concerning model-based translations. These efforts include the analysis of requirements, design, development, validation, and integrated translation microservices of the proposed solutions.

4.3.1 Arrowhead v5.1 Translation Support System (Prototype)

Supported by the development and release of Eclipse Arrowhead v5.1 a translation support system has been added to the Arrowhead technology stack. Eclipse Arrowhead Translation system documentation is available at <https://aitia-iiot.github.io/ah5-docs-java-spring/home/welcome/> while code is available at <https://github.com/eclipse-arrowhead/ah5-translation-manager-java-spring>.

This Arrowhead support system is also modelled in the Arrowhead DSL (SysML v1.6 version). It's available in the Arrowhead v5.1 microsystem library available at <https://github.com/eclipse-arrowhead/ah5-core-java-spring>. The SysML v1.6 has been translated to SysML v2, utilising results from D4.2 section 4.3.1 and 4.3.2.

SysML v2 has a formal base and is primarily a declarative language. To enable an autonomous tool chain work by the Verum company and Karol Kubeila, who made translations from SysML v2 to the Dezyne language, as well as having a formal semantics. The Verum² company already have a tool chain for formal verification of Dezyne³ based executable code.

4.3.2 SysML-based Translation - Towards a Model Governance Approach (Support Activity)

As a key goal in the Arrowhead fPVN project, WP4 aims to bridge functional interoperability gaps by automatically mapping between non-equivalent but functionally similar ports, such as those complying with different standards.

²<https://www.verum.com/>

³<https://www.verum.com/verumdezyne>

These mappings might allow Eclipse Arrowhead to align systems with differing interface formats or standards, enabling a level of interoperability and integration within complex systems of systems that was not achievable before. Validation, i.e., automated conformance, correctness, and completeness guarantee over such scenarios, is a vital engineering activity to the success of such automation approaches.

In addressing interoperability and validation challenges, WP4 leverages the capabilities of SysML v2 as an emerging language for defining and validating systems architecture. SysML v2 introduces enhanced modelling constructs and validation support, which are beneficial for defining rigorous well-formedness criteria within the Arrowhead fPVN context. By implementing a SysML v2-based ruleset, WP4 can validate port configurations to ensure they meet specific requirements, such as valid system-port relationships, interface consistency, and compatibility checks. This includes both basic well-formedness rules (e.g., detecting systems without ports, incorrectly configured ports or interface mismatches) and complex validations that identify systems with ports unsatisfiable by others, either directly or through mapped interfaces.

As a preparatory step, an Arrowhead Library was devised in SysML v2, as a translation of the already existing SysML profile for SysML v1.6. This has been complemented by the translation of the SysML v1.6 library of most released Eclipse Arrowhead core and support systems. In addition, a set of template microsystems, devices, networks, local cords and a system of local clouds has been translated to SysML v2. Finally, an extensive use case on Smart City lightning systems has been modelled using Arrowhead DSL based on SysMLv1.6. This use case has then been translated to the Arrowhead DSL based on SysML v2.

Thus, the SysML v2 Arrowhead DSL library consists of three major parts:

- Library: Corresponds to the SysML v1.6 profile for the Arrowhead DSL.
- Arrowhead core and support system library: Making most Arrowhead v5.1 core and support systems, plus most Arrowhead v4.6 support systems, available as library models.
- Use case examples: Smart city street lighting. This is based on the published paper [12].

These parts are not yet fully released within the Eclipse Arrowhead project, but are available as a release candidate at: <https://github.com/eclipse-arrowhead/profile-library-sysml>. Further, these parts are now being communicated with the standardisation organisation OMG at their Dec 10-11, 2025 meeting in San Francisco, USA. The ambition is to form an OMG working group on SysML SoA modelling based on the Arrowhead DSL.

4.3.3 Object Detection in Engineering Diagrams with Scarce Training Data (Prototype)

Engineering diagrams, such as piping and instrumentation diagrams (P&ID), are these days intended to be maintained as intelligent diagrams and linked to applications such as operation and maintenance systems, as well as computer-aided engineering (CAE) systems. However, a significant portion of existing diagrams is still in the form of raster images. Converting raster images to more easily processed and modified vector graphics can be performed utilising neural networks, as the diagrams largely consist of repeating symbols. The drawbacks of machine learning methods include erroneous detection and a large time investment in both the training data collection and the training itself. Furthermore, disparate diagram sets apply slightly different symbols even if the general form of a symbol is universal. Symbol classes also differ. As a result, current solutions are limited to individual diagram sets, requiring a significant portion of the material to be applied to train the detection model. This is time-consuming and isn't viable for small diagram sets. This task aimed to explore solutions to improve the performance of machine learning-based object detection in engineering diagrams. Advanced machine learning techniques, such as few-shot learning, could minimise the time investment and the required training data size in individual cases. Additionally, generating synthetic training data by augmenting existing diagrams can aid in addressing the lack of material.

Three ML algorithms were tested in the task to assess symbol detection accuracy in P&ID diagrams when training material is scarce. Two of them, FSCE and MPSR, were based on few-shot learning. Few-shot learning in object detection essentially aims to teach the model object classes with only a few examples from each class. The process with the chosen few-shot algorithms was to first train a base model with large quantities of fully labelled data that is similar to the target data and subsequently finetune that base model to adjust to the target data by exploiting the few-shot examples picked from the target material. This finetuning process is a form of transfer learning. With the third algorithm, YOLO, only base training was performed. The resulting model was applied as a comparison baseline for the few-shot algorithms to assess how impactful the finetuning

process is. The datasets for the training, validation, and testing of the ML algorithms were constructed utilising diagrams provided by Semantum and some of its customers, notably Afry. All diagrams were vector graphics and rasterised to be able to use in ML training. One diagram set was chosen for testing, and the few-shot examples were picked from this material. The rest of the diagrams were constructed as training and validation datasets. Data augmentation was utilised by scaling the diagrams to different sizes before rasterisation. Scaling enabled increasing both symbol volume and diversity in the training data. In total, the training and validation datasets consisted of 4580 diagrams that contained 1129700 symbols from 62 classes. 22 symbol classes were present in the test dataset. Mean average precision (mAP) and recall (mAR) were low to moderate with all three algorithms. YOLO and 1-shot FSCE obtained very similar results, below 0.2, while 1-shot MPSR was slightly better with above 0.2 values. 2-shot MPSR increased both mAP and mAR to around 0.3, and with higher scale test images, MPSR obtained above 0.4 mAP and mAR. Class-specific accuracies varied significantly with all algorithms, with notable differences in which symbol classes were well detected. YOLO detected gate valves and process instrumentation functions well with a 0.5 IoU threshold, obtaining above 0.8 accuracy. However, more stringent thresholds dropped the values significantly, and the rest of the symbol classes were mostly poorly detected, with many classes at 0.0 accuracy. FSCE handled off-page connectors best at 0.8 around accuracy. Gate valves were detected very poorly, and process instrumentation functions moderately. Like YOLO, many classes had 0.0 accuracy. Some of the rarer classes, such as tank and heat exchanger, were detected better than with YOLO. MPSR's accuracy was low to moderate with most symbol classes with the 1-shot variation, while increasing the number of few-shot examples increased accuracy across the board, with some classes significantly. Unlike with the other two algorithms, almost all values were higher than 0.0. MPSR performed clearly the best with the rare classes, yet both gate valves and process instrumentation functions were detected poorly. Both FSCE and MPSR predicted bounding boxes better than YOLO, as increasing the IoU threshold didn't notably lower accuracy.

Overall, the algorithms performed better than in studies where they had been tested with photo data, yet the results were still only poor or moderate. Mean average precision and recall were under 0.5 with all tested algorithms. When considering individual symbol categories, the variation between them was sizable. With most symbols, YOLO performed worse than either of the few-shot algorithms but managed to get better results with gate valves and process instrumentation functions, the two most common symbols. This is unsurprising, as both are relatively uniform across most diagram sets. Compared to YOLO, FSCE, and MPSR reached similar or higher precision scores for the other symbol classes. The results with MPSR were largely better than those with FSCE, with four of the 22 categories clearly favouring FSCE and nine categories favouring MPSR. Comparing the results from the few-shot algorithms directly to YOLO results is somewhat misleading, as base training with them utilised Faster R-CNN instead, and that base model was clearly inferior to the YOLO model. Training a potentially better base model with Faster R-CNN or finetuning the YOLO model with a few-shot algorithm capable of that could improve the results. Although the models tested in the task did not achieve the detection accuracy of models specifically trained for individual diagram sets, the results for individual symbol classes suggest that few-shot learning can achieve comparable performance. Moreover, for small diagram sets, these approaches offer the only viable alternative to manual conversion.

4.3.4 Payload Recognition and Mapping (Support Activity)

Data models are essential for data organisation and storage, while information models enable semantic understanding and interoperability across systems and domains. The "translation" of data models and information models refers to the process of converting or mapping one model to another, typically to facilitate interoperability, integration, or communication across different systems or domains. This translation enables systems with different representations, terminologies, or structures for data and information to understand and work with each other's data. Translating data models focuses on structural alignment, while translating information models focuses on semantic alignment, allowing systems to communicate both data and its intended meaning effectively. However, recognising what needs to be translated is a critical first step before any effective translation can occur. Recognition involves identifying and understanding the structure, format, and meaning of each system's data or information model elements. Without recognising and understanding the underlying components, any translation attempt could misinterpret or misalign elements, resulting in a loss of meaning or functionality. Recognition ensures that each element, whether a data field or a semantic concept, is accurately identified, making the translation process both feasible and meaningful. A successful initial approach to payload recognition within the

Arrowhead framework has been implemented using forms in the `mbaigo library`⁴. This approach emerged from the need for backward compatibility and is straightforward: each payload includes a single attribute called version. This attribute makes it easy to identify the type of payload being processed. This method greatly simplifies the code, allowing the same functions to “pack” and “unpack” data models into and from byte arrays, whether they are registration or measurement payloads.

These functions handle different serialisation formats, such as JSON and XML, seamlessly. The natural next step is to enable payload recognition without relying on version attributes. The technical implementation uses Google Go’s interface to ensure that all “forms” conform to a specific contract. A form represents the schema of a data model and includes a few associated methods. The pack function accepts a form and a specified serialisation content type, generating a corresponding byte array, which is then transmitted to a waiting receiving system. Upon receiving the byte array, the system examines the content type from the header of the incoming request or response and passes this information to the unpack function. The unpack function then deserialises the byte array and returns the form to the calling function. The calling function only needs to confirm that the form is of the expected type, ensuring it is correctly mapped.

The library’s forms package contains some basic forms. If a new application requires a unique or customised form, it can define its own, as long as it complies with the form interface (i.e., meets the interface contract). If the form is undocumented, the developer can still retrieve its structure from a received payload. For example, upon receiving a JSON payload, the developer can quickly use a JSON-to-Go converter to generate a corresponding Go struct. After implementing the standard methods, the application can then map and process the form seamlessly.

4.3.5 Arrowhead DSL (SysMLv1) to Arrowhead SysML v2 Library (Prototype)

In the previous deliverable, Task 4.3 detailed the SysMLv1 to SysMLv2 model-to-model translation that focuses on the SysML v1 BDD and IBD elements such as Block, Part, Port and Connector. Those elements are widely used in the Arrowhead DSL.

In this deliverable, we will detail the model-to-model translation from Arrowhead SysMLv1 (based on SysMLv1) to SysMLv2 Library (based on SysMLv2). The Arrowhead DSL is extending the SysMLv1 profile. To move to SysMLv2, we have reused our previous model-based translation (SysMLv1 to SysMLv2).

The Table below shows the mapping that we have identified to move from Arrowhead DSL elements, such as SD, SysD, IDD, to SysMLv2 elements.

Table 6: Arrowhead DSL to SysMLv2 elements

Arrowhead DSL Element	SysML v2 Element
Any stereotype <S> S= SD, SysD, IDD, ...	PartDefinition with semantic metadata @S
FullPort	ItemDef
Stereotype <S> applied on Operation	Action with semantic metadata @S
Part	PartUsage
Generalization	: > relationship

The translation is implemented in Xtend, a Java-like language that permits defining a template and implementing the mapping between the source model element and the target model element. The tooling is implemented in Java inside the Papyrus tool. The translation was tested on several different Arrowhead DSL examples, such as the Arrowhead Platform core elements: ServiceRegistry, Orchestration and SmartGrid examples. The following figures illustrate the ServiceRegistry translation from Arrowhead DSL to SysMLv2. First of all, the user should select the Arrowhead DSL Project and right-click to select the import menu (Figure 42).

The user should select the “ArrowheadDSL to Sysmlv2” item and click Next (Figure 43):

The user then selects the UML file to be transformed to SysMLv2. The target file (.sysml) will be generated in the newly created Project “AFPVNSysMLv2Project” as shown in the Figure 44.

The SysMLv2 generated from the ServiceRegister example is shown in Figure 46.

The translation is available in this github repository: <https://github.com/asmaasmaoui/profile-library-sysml/tree/main/distribution/Papyrus/>

⁴<https://github.com/sdoque/mbaigo>

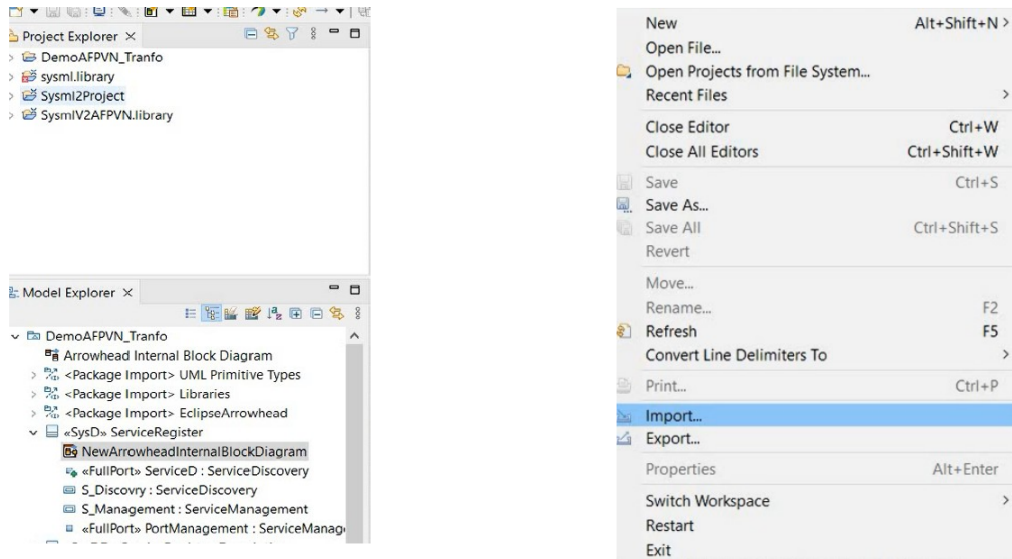


Figure 42: Right Click on the PapyrusProject and select the Import Menu

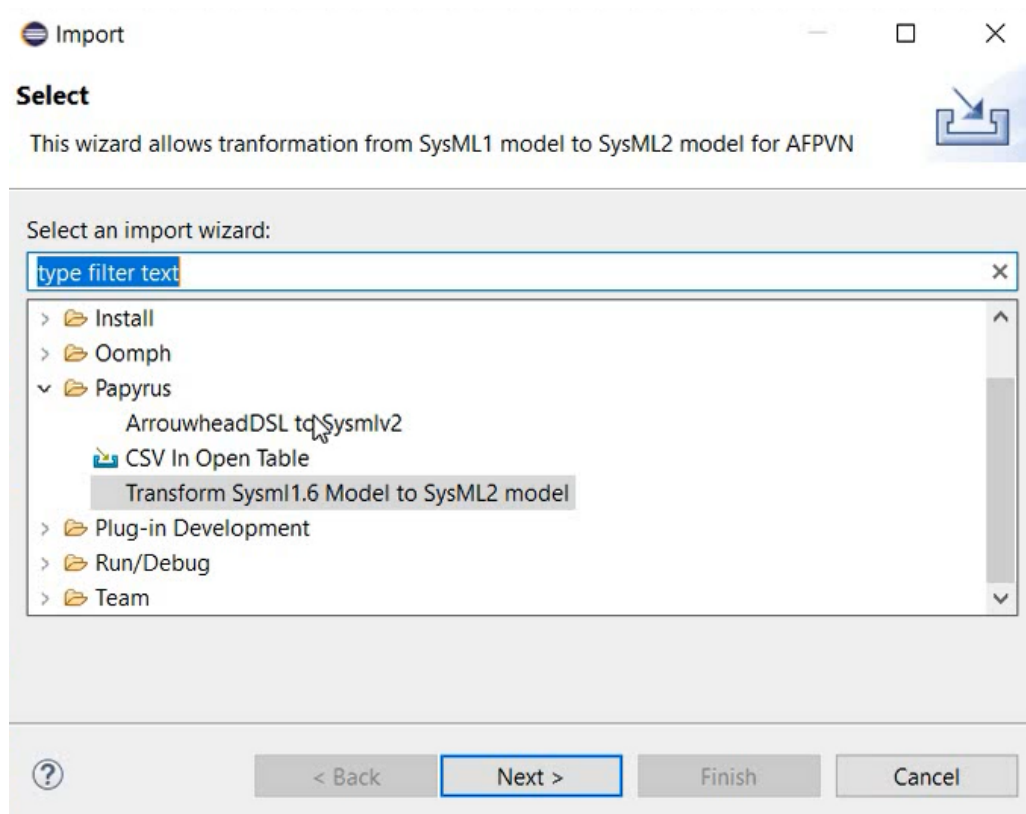


Figure 43: Select "Transform Sysml1.6 Model to SysML2 model"

4.3.6 Using LLM to Generate Arrowhead DSL (SysMLv1) Models (Prototype)

A collaboration between Task 4.3 and Task 4.2 about using LLM for model-based translation was detailed in the previous deliverable. The result was the development of a new Papyrus UML class diagram design technique using LLMs. The LLM helps the Papyrus users in designing a UML class diagram from scratch. The user specifies in the prompt his request about a UML class diagram, for example, "Please generate a UML Class

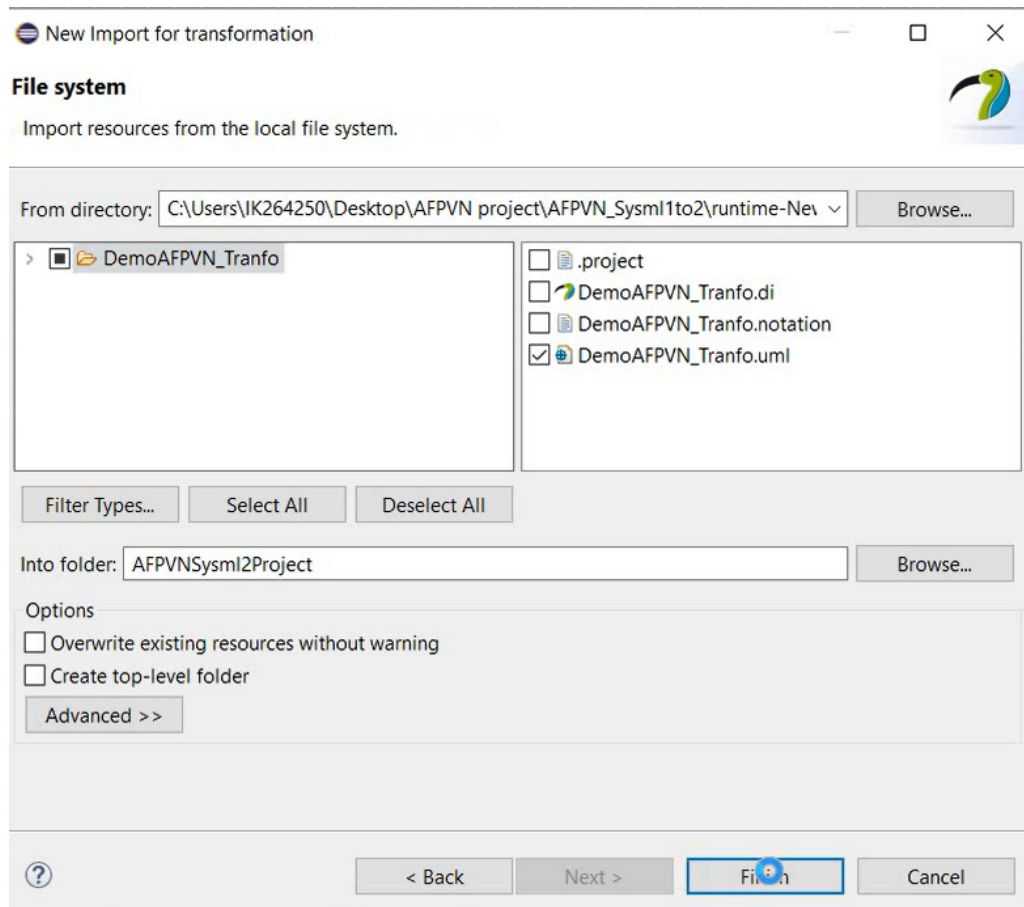


Figure 44: The user should select the Arrowhead DSL model based on SysML1 (uml file) and Click on Finish

Diagram of a Library system”. The LLM will generate the semantic model automatically. We have extended this translation based on LLM to generate an Arrowhead DSL model from the prompts. For example, the user can ask the LLM to produce an Arrowhead DSL for a Light System with an HTTP service.

A video explaining the Design of the Arrowhead DSL-compliant model using LLM is available here: https://www.youtube.com/watch?v=F8MHTjm_btg.

Figures 47, 48, 49, and 50 illustrate how we can use this translation: First of all, the user should create an Empty Arrowhead Project, right-click on the UML file and select the “Modelise with LLM” menu.

A new HMI is open: the user should select the “Arrowhead:LLM:latest” and right in the prompt field, the request. For example: “Please design a Light System with a Monitoring service that has an HTTP operation named TurnOn”.

Figure 50 shows the response of the Lama LLM, which is a good design compliant with Arrowhead DSL.

4.3.7 Rely to SysML (Prototype)

Task 4.3 is developing a solution to actively advance the translation of data from its Rely tool to the Arrowhead Requirements Model. This effort has already reached a significant milestone: the first demo is ongoing, demonstrating the initial integration of Rely into the Papyrus environment. This marks a concrete step forward in ensuring compatibility with broader industry frameworks and validating the approach in practice.

4.3.7.1 Translation from Rely to Arrowhead Requirements Model and SysML

The translation process has been successfully extended to include not only the Arrowhead Requirements Model but also SysML. Several key inputs from the Rely format are now automatically converted into these models, enabling seamless integration into Papyrus for system modelling and validation. This dual translation capability

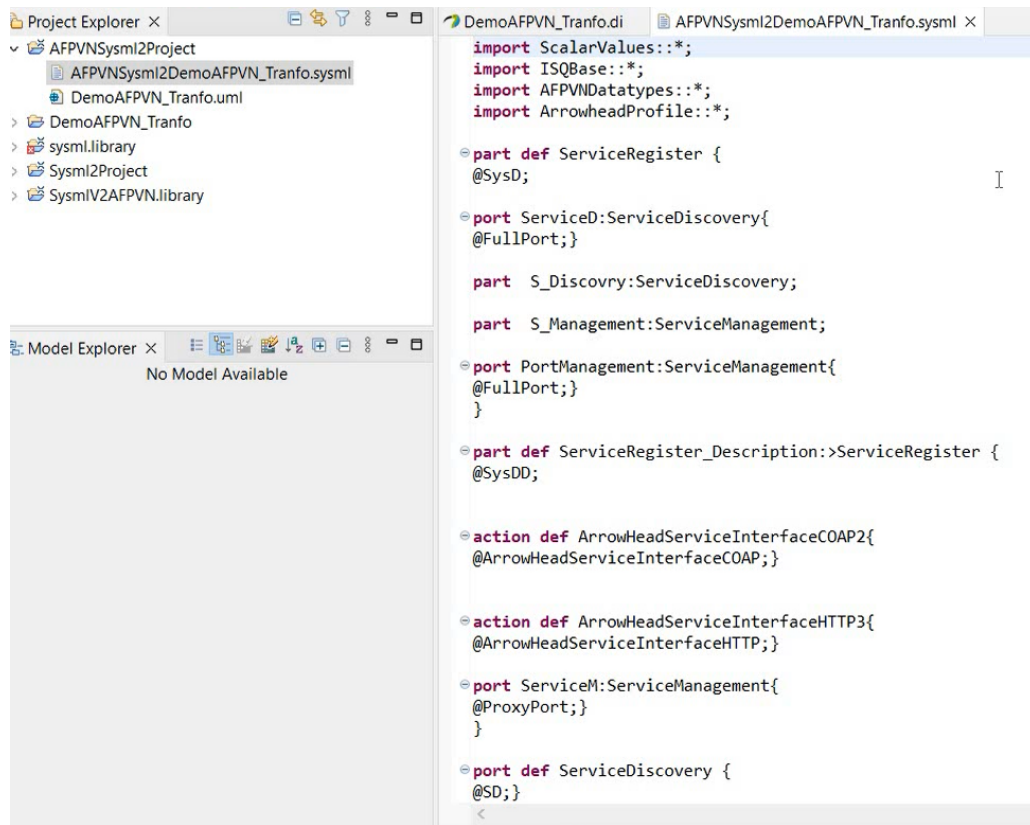


Figure 45: The user should select the Arrowhead DSL model based on SysML1 (uml file) and Click on Finish

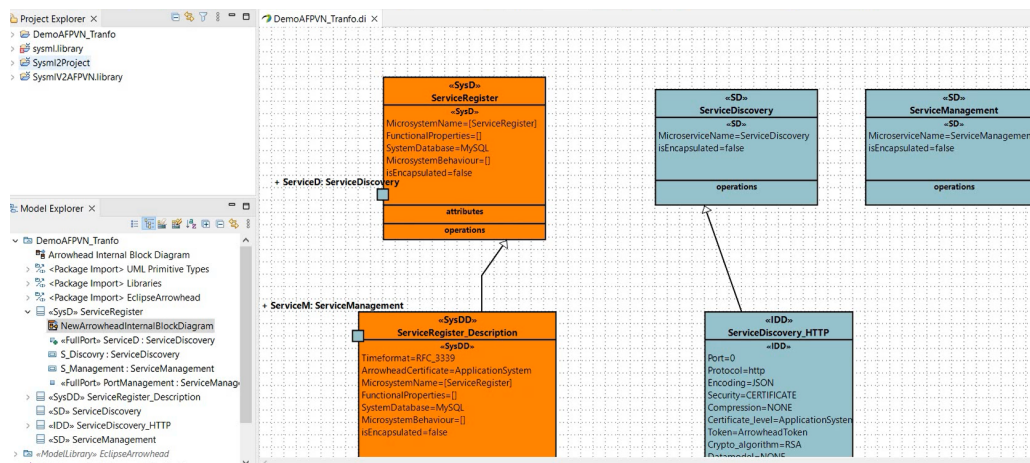


Figure 46: The Arrowhead DSL design of the ServiceRegister model

strengthens interoperability and provides a more comprehensive foundation for system design and requirements management.

4.3.7.2 Specific Use Case Development

Building on the completed translation workflows, the team is currently working on a specific use case to showcase the results. This use case will highlight how Rely-to-SysML and Rely-to-Arrowhead translations can be applied in real-world scenarios, demonstrating the efficiency of test case generation and system validation in Cyber-Physical Systems (CPS). The prototype serves as tangible evidence of progress and will help refine the

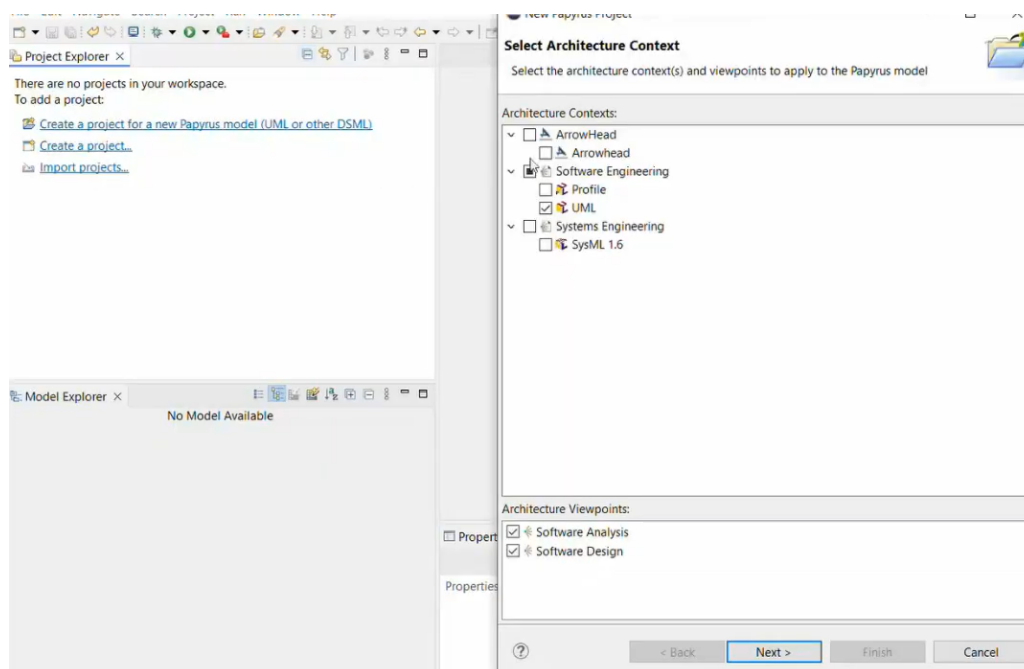


Figure 47: Create a new Arrowhead Project in Papyrus

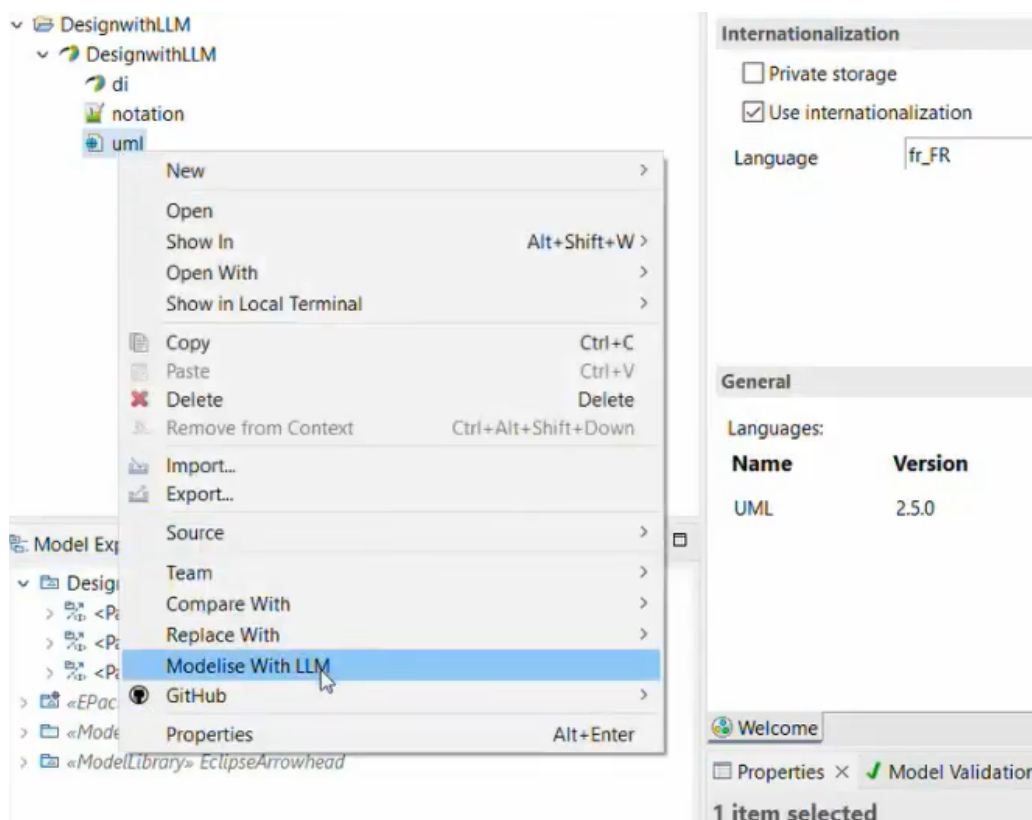


Figure 48: Right-click on the UML file and select the "Modelise with LLM"

methodology for future deployments.

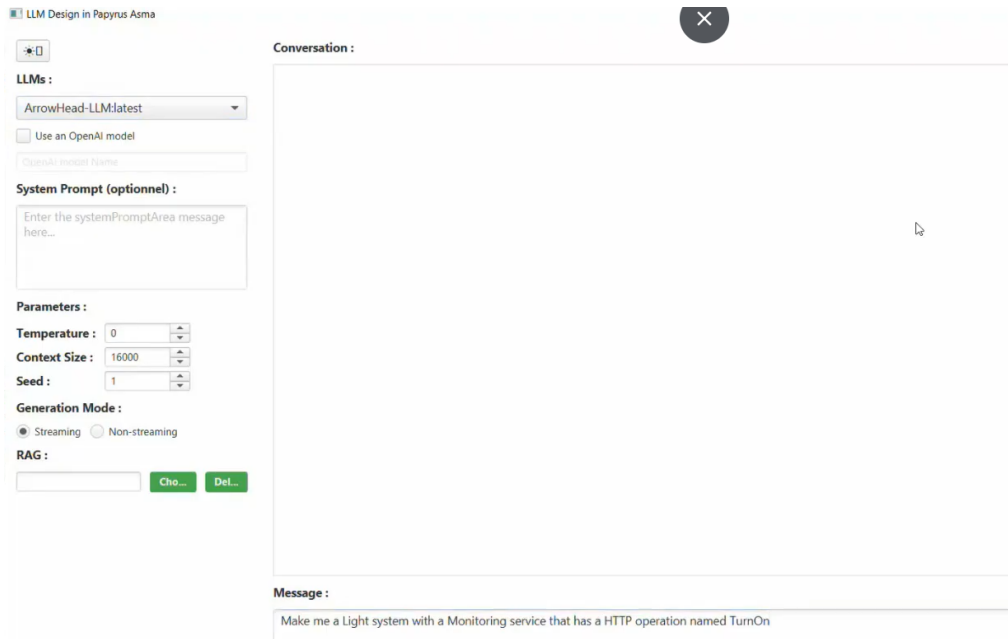


Figure 49: An example of a Prompt to create an Arrowhead DSL model

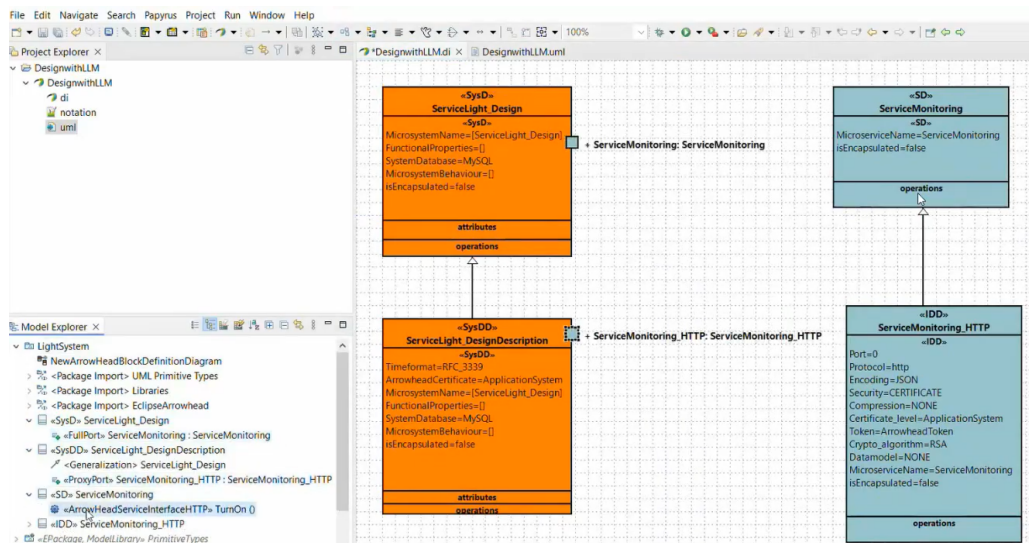


Figure 50: The Arrowhead DSL design generated from LLM

4.3.7.3 Ongoing Collaboration and Integration

This work continues in close collaboration with Task 4.3 and the use-cases, with the integration of Rely into Papyrus remaining a central focus. The ongoing demo and use case development illustrate how the Rely translation tools are being aligned with the Arrowhead framework, paving the way for improved interoperability, streamlined workflows, and enhanced system design capabilities.

4.3.8 Exploring Deep-learning-based Meta-modelling for Data Translation (Support Activity)

This section concludes the research conducted under Deliverable 4.3, focusing on the development and evaluation of a deep-learning-based meta-modelling framework for automated data translation across heterogeneous industrial standards. The core challenge addressed is the critical need for seamless interoperability in modern digital ecosystems like smart manufacturing and transportation, where the diversity of data standards (e.g.,

IEEE 1451, ISO 15926, IEC 61499) creates significant barriers to data integration and system communication.

Our research progressed through two distinct phases. The initial phase, presented at the SMAP 2024 [13] and SDF 2024 [14] conferences, established a foundational hybrid AI framework that integrated Ontology Learning (OL), Knowledge Graphs (KGs), and Large Language Models (LLMs). This initial model demonstrated the viability of the approach, achieving a promising precision of 0.92 and recall of 0.88. It confirmed that combining semantic structuring, relational modelling, and linguistic processing could overcome many limitations of traditional, rule-based translation methods.

Building upon these foundational insights, the second phase focused on creating a significantly enhanced framework, the subject of a forthcoming high-impact journal publication. This state-of-the-art model incorporates advanced techniques to address the limitations of the original approach, including scalability, reasoning depth, and contextual accuracy. Key innovations include the integration of Graph Retrieval-Augmented Generation (GraphRAG) for superior context retrieval from KGs, Chain-of-Thought (CoT) prompting to elicit complex, step-by-step reasoning from LLMs, the use of hypergraph structures to model complex multi-entity relationships, and community detection algorithms for optimising knowledge graph analysis.

Comprehensive empirical evaluation of the enhanced framework revealed substantial performance gains over the original model. We recorded a 10.59% increase in precision (to 0.94), a 12.20% increase in recall (to 0.92), a 12.05% increase in F1-score (to 0.93), and a 10.47% increase in overall accuracy (to 0.95). Rigorous ablation studies confirmed that the new components were critical to this success, with CoT reasoning and GraphRAG integration being the most impactful contributors. Qualitatively, the enhanced framework produced vastly more comprehensive and useful translations, with response lengths increasing by over 1000%, providing not just mappings but also explanations, examples, and contextual clarifications.

This research successfully demonstrates a clear path from a viable conceptual model to a powerful, empirically validated, and industry-ready solution for automated data translation. The enhanced framework sets a new benchmark in the field, offering a scalable, accurate, and robust methodology for achieving true interoperability in complex, data-intensive environments. The findings lay the groundwork for future advancements in explainable AI, user-in-the-loop systems, and the handling of dynamic, evolving standards.

4.3.8.1 The Critical Challenge of Data Interoperability in Modern Industry

The Fourth Industrial Revolution (Industry 4.0) is fundamentally transforming sectors such as smart manufacturing, transportation, and industrial automation. This transformation is driven by the convergence of physical and digital systems, generating unprecedented volumes of data from a multitude of sources, including sensors, actuators, control systems, and enterprise platforms. The ability to seamlessly integrate, interpret, and leverage this data is paramount for enabling advanced functionalities like predictive maintenance, autonomous operations, real-time supply chain optimisation, and intelligent decision-making. However, this vision is profoundly hindered by a persistent and complex challenge: the lack of data interoperability. Industrial ecosystems are characterised by a heterogeneous landscape of data standards, each developed for specific domains and purposes. For instance:

- IEEE 1451 provides a standardised interface for smart transducers (sensors and actuators), defining communication protocols and data formats (TEDS).
- ISO 15926 offers a data model for the integration and lifecycle management of process plant information, crucial in industries like oil and gas.
- IEC 61499 specifies an architecture for distributed industrial process measurement and control systems, using function blocks to model automation logic.

While each standard is invaluable within its domain, their differing semantics, structures, and terminologies create "data silos," making it exceptionally difficult for systems based on different standards to communicate and share information effectively. Traditional solutions to this problem, such as manual mapping or rule-based middleware, are notoriously labour-intensive, error-prone, brittle, and incapable of scaling to the complexity and dynamism of modern industrial data. They struggle to capture the nuanced semantics and implicit relationships essential for accurate translation, leading to data loss, misinterpretation, and system failures.

This interoperability gap represents a major bottleneck to innovation and efficiency. Addressing it requires a paradigm shift towards more intelligent, automated, and scalable solutions for data model translation—a process of accurately mapping data from a source model to a target model while preserving its structural and semantic integrity.

4.3.8.2 The Evolution of the Research: From Foundational Concept to State-of-the-art Framework

This section documents the culmination of a research journey aimed at developing such a solution. Our work progressed systematically, evolving from an initial conceptual framework to a highly sophisticated and rigorously validated system.

- **Phase 1: The Foundational Hybrid Framework.** Our initial research, presented at the IEEE SMAP 2024 and SDF 2024 conferences, proposed a novel hybrid AI framework. The central hypothesis was that by synergistically combining three distinct AI methodologies, we could overcome the limitations of any single approach. The framework is integrated:
 1. **Ontology Learning (OL):** To systematically extract and structure domain concepts and relationships from standards documentation, creating a formal semantic foundation.
 2. **Knowledge Graphs (KGs):** To represent the extracted knowledge in a flexible, graph-based format, modelling the intricate relationships within and between standards.
 3. **Large Language Models (LLMs):** To process the natural language text of the standards, handle linguistic nuances, and drive the translation process. The preliminary results from this phase were highly encouraging, validating the core concept and demonstrating a significant improvement over traditional methods. However, this initial work also illuminated areas for improvement, particularly regarding the depth of LLM reasoning, the efficiency of knowledge retrieval from the KG, and the representation of highly complex, multi-entity relationships.
- **Phase 2: The Enhanced Hybrid Framework:** Building on the lessons learned, we developed an enhanced framework designed to push the boundaries of performance and capability. This advanced model, which is the subject of a forthcoming journal publication, incorporates state-of-the-art techniques to create a more robust, intelligent, and accurate system. This report provides a detailed exposition of this enhanced framework, its underlying technologies, its comprehensive evaluation, and its profound implications for the field of data interoperability.

4.3.8.3 Foundational Research: the Initial Hybrid AI Framework

The first phase of our research focused on establishing the viability of a hybrid AI approach to data model translation. The work, detailed in our publications for SMAP 2024 and SDF 2024, resulted in a foundational framework that served as the baseline for all subsequent enhancements.

Conceptualisation and Core Components

The central premise of the initial framework was that no single AI technology could adequately address the multifaceted challenge of data translation. LLMs, while powerful in processing language, lack the structured, verifiable knowledge base needed for high-fidelity industrial applications. KGs excel at representing structured relationships but struggle with interpreting unstructured text. OL provides a formal semantic backbone but can be rigid and difficult to scale. Our hypothesis was that their combination would be synergistic, with each component compensating for the weaknesses of the others. The three core components were assigned distinct roles:

1. **Ontology Learning for Semantic Structuring:** The process began with OL techniques to parse standards documentation and extract key entities (e.g., TransducerChannel, FunctionBlock), attributes, and relationships. This step transformed unstructured and semi-structured text into a formal, machine-readable ontology (using standards like OWL), providing a canonical representation of each data model's semantics. This created the foundational layer of understanding.
2. **Knowledge Graphs for Relational Modelling:** The concepts and relationships extracted by OL were then used to construct a Knowledge Graph using a platform like Neo4j. The KG served as a flexible and scalable knowledge base, capturing not just direct relationships but also enabling the inference of indirect connections. It modelled the data standards as an interconnected graph of knowledge, making it possible to query for complex cross-standard pathways.
3. **Large Language Models for Linguistic and Contextual Processing:** An LLM (specifically, an 8B parameter LLaMA model) was employed as the core translation engine. It was tasked with taking a source

data concept and, using the knowledge contained within the KG as context, generating the corresponding concept in the target standard. The LLM's role was to handle the linguistic nuances, variations in terminology, and contextual subtleties that are difficult to capture with purely symbolic methods.

Initial System Architecture and Workflow

The workflow of the foundational framework (See Figure 51) was designed as a sequential pipeline, reflecting a logical progression from data ingestion to final translation.

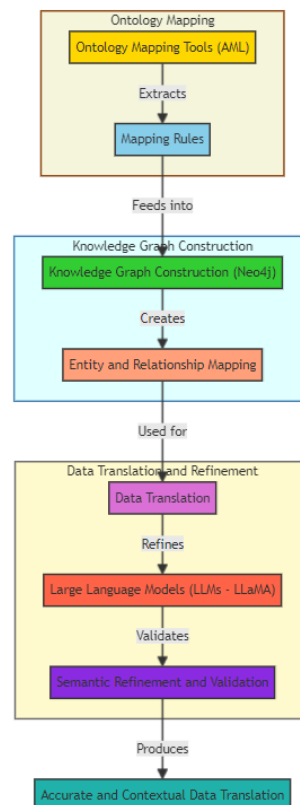


Figure 51: Hybrid Framework Integrating Ontology Learning, Graph Neural Networks, and Large Language Models for Data Model-Driven Translation

The process unfolded as follows:

1. **Ontology Mapping:** An Alignment Mapping Language (AML) tool was used to automatically identify initial correspondences between ontologies derived from the different standards. This step generated a set of preliminary mapping rules.
2. **Knowledge Graph Construction:** These rules and the full ontology definitions were ingested into a Neo4j graph database. Each concept became a node, and each relationship became an edge, creating a unified graph representing the knowledge from all relevant standards.
3. **LLM-Powered Translation:** For a given translation task, the LLM was prompted with the source concept. Crucially, its prompt was augmented with relevant contextual information retrieved from the KG. For example, to translate an IEEE 1451 TransducerChannel, information about its properties and connections would be retrieved from the KG and provided to the LLM.
4. **Output Generation:** The LLM would then process this context-rich prompt and generate the translation, identifying the equivalent entity and its properties in the target standard (e.g., a specific data object in ISO 15926).

Preliminary performance and Key Insights

The initial framework was evaluated on its ability to translate concepts between IEEE 1451, ISO 15926, and IEC 61499. The evaluation yielded promising results, establishing a strong proof-of-concept.

Table 7: Evaluation of Different Methods of Data Translation

Criteria	Model Hybrid	OL	KGs	LLMs
Precision	0.92	0.85	0.75	0.90
Recall	0.88	0.60	0.75	0.85
Scalability	500K entires	10K entires	1M nodes	500K entires
Efficiency	12 ms/entry	2 ms/entry	5 ms/node	10 ms/entry
Robustness	35% noise	20% noise	30% noise	25% noise

Key Insights and Identified Limitations:

- **Validation of the Hybrid Approach:** The results strongly confirmed that the hybrid model was effective. The accuracy achieved (Precision 0.92, Recall 0.88) was a significant improvement over what could be expected from any single technology or traditional rule-based systems.
- **Scalability Concerns:** While the KG backend could handle up to 1 million nodes, the integration with the LLM and the sequential nature of the pipeline raised concerns about performance on extremely large-scale, real-time datasets.
- **Superficial Reasoning:** The LLM, while context-aware, exhibited relatively superficial reasoning. It could perform direct, one-to-one mappings well, but it struggled with complex, multi-hop translations that required synthesising information from disparate parts of the KG. The translations often lacked depth and explanation.
- **Inefficient Context Retrieval:** The method for retrieving context from the KG was basic. It often pulled in either too much irrelevant information or missed crucial, indirectly related concepts, leading to suboptimal prompts for the LLM.
- **Representational Limits of Standard KGs:** Standard graph models (nodes and binary edges) were sometimes insufficient to capture complex industrial realities, such as relationships that involve more than two entities (e.g., a calibration procedure that links a sensor, a standard, and a technician).

These insights were invaluable. They not only validated our initial direction but also provided a clear roadmap for the necessary enhancements. The goal for the next phase was clear: to build upon this solid foundation to create a system with deeper reasoning, more intelligent context awareness, and a more expressive knowledge representation.

4.3.8.4 The Enhanced Hybrid Framework: A State-of-the-Art Approach

The second phase of our research was dedicated to systematically addressing the limitations of the foundational framework. The result is an enhanced hybrid framework that incorporates several cutting-edge AI techniques to deliver a new level of performance in accuracy, robustness, and qualitative output.

Rationale for Enhancement

The primary drivers for the enhancement were the needs for:

1. **Deeper Reasoning:** To move beyond simple mappings to complex, inferential translations that mimic the problem-solving process of a human domain expert.
2. **Smarter Knowledge Retrieval:** To ensure the LLM receives precisely the right context—no more, no less—to perform its task accurately, emulating an expert who knows exactly which part of a manual to consult.

3. **More Expressive Knowledge Representation:** To model the true complexity of industrial data relationships, which are often not simple pairs but multi-faceted interactions.
4. **Improved Robustness and Automation:** To create a more resilient and automated pipeline from raw data to structured knowledge.

To achieve this, we re-architected key parts of the framework and integrated several novel components.

Architectural Advancements and Core Innovations

The enhanced framework is not merely an incremental update; it represents a fundamental redesign of the knowledge processing and reasoning core.

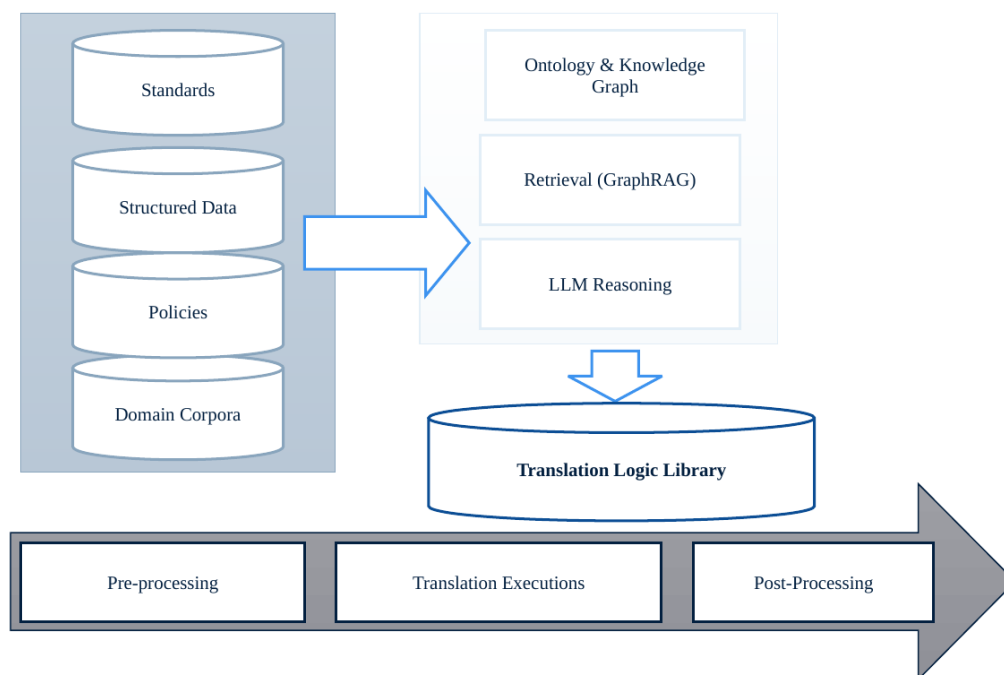


Figure 52: Enhanced System architecture of Hybrid Framework for data translation

Instead of relying on simpler extraction tools, we adopted a pipeline inspired by frameworks like **OntoGenix**. This provides a structured, multi-stage workflow for ontology development that is both more automated and more robust. The pipeline includes stages for rigorous data pre-processing (normalisation, cleaning, named entity recognition tailored for technical language), concept extraction, validation (logical consistency checks, redundancy removal), and refinement, allowing for an iterative loop of improvement with expert feedback. This ensures that the knowledge base is built on a high-quality, clean, and logically sound semantic foundation.

We made two major enhancements to the knowledge graph component:

1. **Hypergraph Structures:** To overcome the limitations of traditional graphs, we introduced the concept of hypergraphs. In a standard graph, an edge connects exactly two nodes. In a hypergraph, a single "hyperedge" can connect an arbitrary number of nodes. This is perfectly suited for modelling complex industrial relationships. For example, a single "Maintenance Event" hyperedge could connect the nodes for `Asset_ID_123`, `Technician_JaneDoe`, `Procedure_Manual_Rev4`, and `Timestamp_XYZ`. This provides a much richer and more accurate representation of reality than trying to model the same event with a series of pairwise edges.
2. **Community Detection:** To improve query efficiency and understand the high-level structure of the standards, we integrated the **Leiden algorithm** for community detection. This algorithm groups densely interconnected nodes into clusters or "communities." In our context, these communities naturally correspond to major functional areas within a standard (e.g., a community for "Timing and Synchronisation" in

IEEE 1451). This structural insight allows the reasoning engine to narrow its search space, leading to faster and more relevant context retrieval.

This is the most significant innovation in the enhanced framework, transforming the LLM from a simple translation tool into a sophisticated reasoning engine.

1. **Graph Retrieval-Augmented Generation (GraphRAG):** We replaced the basic context retrieval mechanism with GraphRAG. Standard RAG retrieves chunks of text from a document database. GraphRAG, by contrast, leverages the structure of the knowledge graph itself. When a query is made, GraphRAG doesn't just find the starting node; it traverses the graph to retrieve a relevant subgraph—a network of interconnected concepts. This retrieved context is far more powerful because it includes not just the entity's definition but also its relationships, its neighbours, and its place within the broader community structure. This provides the LLM with a rich, multi-dimensional view of the concept, mirroring how a human expert draws on a network of knowledge. We implemented parallel retrievers (semantic, structural, and keyword-based) to ensure the retrieved context is comprehensive.
2. **Chain-of-Thought (CoT) Prompting:** To unlock deeper reasoning, we implemented **Chain-of-Thought (CoT) prompting**. Instead of asking the LLM for a direct answer, CoT prompts instruct the model to "think step by step" and lay out its reasoning process before giving the final answer. For a translation task, the LLM might first identify the core function of the source concept, then search for similar functions in the target standard's context provided by GraphRAG, then compare attributes, and finally conclude with the most likely mapping, explaining why it made that choice. This simple-sounding technique has a profound impact: it dramatically reduces errors, makes the model's output more reliable, and, crucially, makes the translation process **explainable**.

The enhanced framework also supports multiple state-of-the-art LLMs (e.g., GPT-4o, Gemini 2.0, Claude 3.5) with a dynamic selection mechanism to choose the best model for a given task. It also enables multimodal processing to handle inputs like tables and code snippets found in standards, and parallel model execution to improve robustness through ensembling and cross-validation of results.

The Enhanced System Workflow

The workflow is now more dynamic and iterative. A query initiates a GraphRAG process to build a contextual subgraph. This subgraph, along with a CoT prompt, is fed to the selected LLM. The LLM generates a detailed, reasoned response, which is then parsed to extract the final translation logic. This logic is stored and applied in a robust, three-stage execution pipeline: pre-processing, translation, and post-processing (validation and formatting).

4.3.8.5 Comprehensive Evaluation and Empirical Findings

A cornerstone of the second research phase was a rigorous, multi-faceted evaluation of the enhanced framework. This evaluation was designed to quantify its performance improvements over the original model, dissect the contributions of its new components, and assess the qualitative leap in its output.

Experimental Setup

The evaluation was conducted using a dataset of documents and data models from the IEEE 1451, ISO 15926, and IEC 61499 standards. A ground truth of correct mappings was generated to serve as the benchmark for accuracy. The experiments were run on a server to support the demanding computational needs of the advanced models.

The following metrics were used:

- **Precision, Recall, F1-Score, and Accuracy:** Standard metrics for classification tasks to measure the correctness of the generated mappings.
- **Property Accuracy:** A specific metric to evaluate how well the framework maps not just the concepts but also their associated attributes.
- **Translation Validation Score:** A composite score (from 0 to 1) measuring the overall quality and reliability of the translation between standard pairs.

- **Qualitative Metrics:** Response length and comprehensiveness, measured via character count and qualitative analysis.
- **Time Performance:** Processing, translation, and query times.

Performance Comparison: Original vs. Enhanced Framework

The results demonstrate a clear and substantial superiority of the enhanced framework across every quantitative metric.

Table 8: Performance Comparison of extended system

Metric	Original Framework	Enhanced Framework	Improvement (%)
Precision	0.85	0.94	+10.59%
Recall	0.82	0.92	+12.20%
F1 Score	0.83	0.93	+12.05%
Accuracy	0.86	0.95	+10.47%

The most notable gain was in recall, which increased by 12.20%. This indicates that the enhanced framework is significantly better at identifying all relevant mappings, reducing the risk of "silent failures" where a necessary translation is simply missed. The improvement in property mapping accuracy (a 25% increase) also confirms that the translations are not just conceptually correct but also more complete and usable.

Ablation Studies: Deconstructing the Performance Gains

To understand which of the new components contributed most to this success, we conducted a series of ablation studies. We systematically disabled key features of the enhanced framework and measured the resulting drop in performance (specifically, F1-score).

Table 9: Ablation study of extended system

Framework Configuration	F1 Score	Performance Drop
Full Enhanced Framework	0.93	-
Without Chain-of-Thought (CoT)	0.88	-5.38%
Without GraphRAG	0.89	-4.30%
Without Parallel Retrievers	0.90	-3.23%

The results were unequivocal:

- **Chain-of-Thought Reasoning** had the largest impact. Removing it caused the F1-score to drop by over 5%, confirming that forcing the LLM to perform structured, step-by-step reasoning is critical for achieving high accuracy in complex tasks.
- **GraphRAG** was the second most important component, with its removal leading to a 4% drop in F1-score. This proves the superiority of graph-based structural retrieval over simpler context-providing mechanisms.
- Advanced graph analysis features like **Community Detection** also made significant contributions, demonstrating their value in optimising the reasoning process.

Qualitative Analysis: A Paradigm Shift in Output Quality

Beyond the quantitative metrics, the difference in the *quality* of the output was staggering.

- **Original Framework Response:** A typical response would be a direct mapping, e.g., "IEEE1451:MaxPhysVal maps to ISO15926:MaximumValue."

- **Enhanced Framework Response:** A response from the enhanced framework using CoT is a detailed explanation: "Step 1: The source concept is IEEE1451:MaxPhysVal, which represents the maximum physical value a sensor can measure. Step 2: Searching the ISO 15926 context for concepts related to limits and measurements, I find MaximumValue. Step 3: Comparing their definitions and usage, both serve to define the upper bound of an operational range. Therefore, the most accurate translation is ISO15926: MaximumValue. This property is essential for ensuring measurements stay within the sensor's calibrated limits."

Time and Resource Efficiency

The enhanced framework is more computationally intensive, with the total processing time per transaction being slightly higher than that of the original model.

While the upfront processing to build the sophisticated knowledge base is significant, the actual query and translation times remain highly efficient, making the system suitable for interactive applications once the initial setup is complete. The substantial gains in accuracy, reliability, and explainability represent a highly favourable trade-off for the modest increase in computational resources.

4.3.8.6 Discussion, Implications, and Future Directions

Synthesis of Findings

This research successfully designed, implemented, and validated a state-of-the-art hybrid AI framework for data model translation. The journey from the foundational concept to the enhanced model demonstrates a clear evolutionary path, where each advanced component was strategically integrated to address specific, identified limitations. The final framework's performance, proven through rigorous quantitative and qualitative evaluation, establishes a new benchmark for automated interoperability solutions. We have shown that by deeply integrating structured knowledge (in advanced KGs) with the reasoning capabilities of cutting-edge LLMs (via GraphRAG and CoT), it is possible to create a system that approaches expert-level performance in the complex task of cross-standard data translation.

Implications for Industry

The implications of this work for industries reliant on data interoperability are significant:

- **Accelerated System Integration:** The framework can dramatically reduce the time and cost associated with integrating disparate systems, replacing months of manual mapping effort with a highly automated process.
- **Enhanced Reliability and Reduced Errors:** By providing accurate, context-aware translations, the framework reduces the risk of data misinterpretation that can lead to operational failures, safety incidents, or flawed business intelligence.
- **Enabling Advanced Applications:** True interoperability unlocks the potential for more sophisticated cross-domain applications. For example, a smart factory could seamlessly integrate data from its IEC 61499-based control systems with lifecycle data from an ISO 15926-based asset management platform and real-time sensor data from IEEE 1451-compliant devices to create a comprehensive digital twin.
- **Democratisation of Expertise:** The explainable nature of the framework can serve as a knowledge and training tool, helping engineers understand the complex relationships between different standards.

Limitations and Future Research Directions

Despite its successes, the framework is not without limitations, which point toward exciting avenues for future research.

1. **Limited Standard Coverage:** Our evaluation focused on three major industrial standards. Future work should expand this to a much broader range of standards across different domains (e.g., healthcare, finance) to test the framework's generalizability.

2. **Handling Ambiguity:** While significantly improved, the framework can still struggle with highly ambiguous or context-dependent terminology. Future research could explore more advanced semantic disambiguation techniques.
3. **Full Multimodal Understanding:** The current framework has basic support for tables and code. A key future direction is to integrate advanced computer vision models to interpret and translate information directly from diagrams, schematics, and P&ID charts found in standards documentation.
4. **User-in-the-Loop Refinement:** The next logical step is to develop an interactive user interface that allows domain experts to review, correct, and refine the framework's translations. This feedback could be used to continuously fine-tune the models, creating a virtuous cycle of improvement.
5. **Temporal Dynamics:** Standards evolve. Future versions of the framework should be able to handle versioning and change management, automatically identifying differences between standard revisions and updating the translation logic accordingly.
6. **Enhanced Explainability (XAI):** While CoT provides a degree of explainability, deeper integration with formal XAI methods could provide even more robust and verifiable explanations for the framework's decisions, which is critical for adoption in safety-critical systems.

4.3.8.7 Conclusion

The challenge of data interoperability remains a significant barrier to realising the full potential of Industry 4.0. This report has detailed the successful development and validation of an enhanced hybrid AI framework that provides a powerful, automated solution to this challenge. By moving from a foundational concept to a state-of-the-art system integrating GraphRAG, Chain-of-Thought reasoning, and advanced knowledge graph structures, we have demonstrated unprecedented gains in translation accuracy, robustness, and qualitative depth.

The final framework represents a significant contribution to the field of applied AI, offering a tangible pathway to breaking down data silos in complex industrial environments. It not only solves a critical engineering problem but also paves the way for future research into more intelligent, explainable, and collaborative AI systems. The source code and detailed implementation accompanying our journal publication [15] will enable other researchers and practitioners to build upon this work, accelerating the journey towards a truly seamless and interconnected digital world.

4.3.9 Translating Industrial Data Into System Models with SysML v2 (Proposal)

Modern engineering projects require seamless integration of heterogeneous industrial data into coherent system models. SysML v2 offers advanced capabilities for semantic consistency, traceability, and lifecycle management, making it a cornerstone for Model-Based Systems Engineering (MBSE).

Industrial environments produce vast datasets in diverse formats, such as (taking a project focus example, process industry as an example) Piping and Instrumentation Diagrams (P&IDs), process flow diagrams, and component specifications. Translating these into formal SysML v2 models ensures interoperability, automation, and digital twin readiness. SysML v2 supports ontology-driven mapping, enabling semantic traceability and reducing manual transformation errors.

4.3.9.1 Validation and Verification in SysML v2

SysML v2 introduces layered quality management for MBSE. Validation ensures well-formedness of models, checking syntax and semantics to prevent unreachable actions or inconsistent flows. Verification confirms that models meet mission goals and system requirements. Automated workflows integrate with CI/CD pipelines, enabling continuous quality assessment, technical debt tracking, and KPI dashboards. These practices boost interoperability and reliability across projects.

Key aspects include:

- Built-in validation rules for authoring success.
- Governance policies for multi-project consistency.

- Automated quality gates and dashboards.
- AI-assisted rule generation and analysis for future scalability.

4.3.9.2 DEXPI Background

DEXPI (Data Exchange in the Process Industry) is a standardisation initiative aimed at enabling interoperable data exchange across the process plant lifecycle. Initially focused on P&IDs, DEXPI now supports process flow diagrams and block flow diagrams through its UML-based information model and XML serialisation format introduced in DEXPI 2.0. This shift from document-centric to data-centric approaches aligns with Industry 4.0 principles, ensuring long-term accessibility and integration of engineering data.

Benefits of DEXPI:

- Seamless integration between design and simulation tools.
- Standardized XML serialization for P&IDs, PFDs, and BFDs.
- Reduced vendor lock-in and improved interoperability.
- Foundation for AI-assisted process design and digital twins.

4.3.9.3 Industrial Data Ontology (IDO) Background

IDO is a W3C OWL 2 ontology designed for semantic interoperability across industrial domains. It serves as a top-level ontology for ISO 23726, enabling machine-interpretable data exchange throughout asset lifecycles. IDO provides formal semantics for reference data libraries, supporting automated reasoning, quality assurance, and AI-driven applications. Its adoption ensures consistent vocabularies and unambiguous meaning across engineering systems, reducing manual mapping and enhancing automation.

Key features:

- Based on ISO 15926 principles.
- Enables ontology-based interoperability for Industry 4.0.
- Supports lifecycle traceability and digital twin initiatives.
- Facilitates AI and automated V&V through formal logic.

4.3.9.4 Results and Next Steps

Translating industrial data into SysML v2 system models is essential for achieving interoperability, automation, and lifecycle traceability. V&V practices ensure model integrity and compliance, while standards like DEXPI and IDO provide the semantic backbone for integration. Together, these technologies pave the way for advanced MBSE ecosystems and future AI-driven engineering workflows.

We have achieved identifying a layered co-modelling ecosystem, with industrial data and knowledge graph ontologies (encoding domain-specific knowledge) that are used in conjunction with system models, again, being organised via SysML extension mechanisms. We've been focusing on showcasing a unique key aspect of this framework, i.e., adding V&V as a service on top of multi-domain, multi-modal engineering scenarios (demonstrated via DEXPI models), serving as a well-founded correctness engine behind automated translation activities happening in complex systems engineering.

Towards finalisation, we'll be building an industrial demonstration for showcasing the framework based on XML interpretations of P&ID diagrams.

4.3.10 Data Representation and Translation with Emerging Paradigms (Support Activity)

Task 4.3 efforts were put to align with the partners' analysis of ontology-based translation tools, machine learning techniques for addressing complex mapping challenges, and model-based approaches for evaluating translations across diverse data modelling languages.

This collaborative effort can be further enriched by exploring the emerging use of forensic digital twins—virtual replicas designed to capture provenance, traceability, and evidence-based validation throughout translation workflows. Task 4.3 has recently experimented with such tools and produced some prototypes, presented at

academic events and to stakeholders. We plan to synchronise with the partners with the purpose of taking advantage of our gained knowledge in this field.

In parallel, we have developed the production of graphical digital twins through Implicit Neural Representations (INRs), which offer a promising direction for visualising and interacting with physical objects in a simulated digital environment, easing the integration with other emerging workflows such as agentic systems. Together, these extensions provide not only a technical foundation for domain translation pipelines but also a framework for transparency, reproducibility, and cross-domain interoperability.

4.3.11 UML/SysML-driven Translation for IoT Environmental Data (Proposal)

Task 4.3 in D4.2 emphasises model-based approaches (UML, SysML, formal semantic reasoning) for translation across heterogeneous modelling languages. This is part of the solution presented in Section 4.1.8 and 4.2.11. In D4.3, this capability is extended to environmental monitoring systems by:

- modelling sensor stations as SysML blocks,
- modelling data streams as UML activity diagrams,
- mapping physical measurements to model-based metaclasses (e.g., Measurement, Signal, Observed-Property).

Example SysML abstractions

- Block "SensorNode" - attributes: id, location, measurementSet
- Block "Measurement" - attributes: timestamp, value, unit, semanticType
- Block "EnvironmentalStation" - composed of SensorNode blocks

4.3.11.1 Model-based Serialisation Into a Standard Schema

```

1 import xml.etree.ElementTree as ET
2
3 def serialize_measurement_to_xml(measurement):
4     """
5     measurement: { "concept": "WaterQuality:Salinity", "value": 18.2, "unit": "ppt" }
6     """
7     root = ET.Element("Measurement")
8     ET.SubElement(root, "SemanticType").text = measurement["concept"]
9     ET.SubElement(root, "Value").text = str(measurement["value"])
10    ET.SubElement(root, "Unit").text = measurement["unit"]
11
12    return ET.tostring(root, encoding="utf-8").decode()

```

This script demonstrates the model-based generation of standard-compliant representations.

4.3.12 Aerobridge Translator (Prototype)

Based on the S5000F, the "AeroBridge", an Aerospace S5000F automatic translator, aims at defining a model-based translation system that converts local data to an XML message in compliance with S5000F, and vice versa.

Aerobridge should be used by companies that want standardised data flow with Suppliers and Customers.

The use of this translator must be preceded by a Guidance Phase between the parties wishing to exchange data. During this phase, a guidance meeting and document have to be developed, where technical data exchange has to be defined in terms of data to be exchanged, their format, business rules and for which analysis they will be used.

"Aerobridge" is a Model-based translator. As described in D4.1 and D4.2, Model-based translation involves translating data between different models by mapping elements and attributes from one model to another. This approach focuses on understanding the structure and semantics of each model and developing mappings that preserve the meaning and integrity of the data during translation.

This software is a Python-based algorithm solution designed to generate and read XML messages compliant with the S-Series standards.

To run the algorithm, the following requirements must be met:

- Python Environment: Python 3.6 or higher.
- Java: Java version 8 or higher
- System: The algorithm can be deployed on both servers and embedded systems, provided they meet the necessary computational and storage capabilities.

The translation process is illustrated in the following table:

Table 10: Translation Process

#	Transmission (TX)	Reception (RX)
1.	UC (Use Case) Identification	
	Identification of the correct use case mapped into SF5000F or development of a new UC to transfer the relevant data.	
3.	Develop UML (Unified Modelling Language)	
	Generate in Enterprise Architect (EA) (or similar software) the UML associated with the UC.	
5.	Mapping	
	The data (object of TX/RX communication) are mapped to the corresponding attributes in the S-Series XML schema.	
7.	Data Extraction	Input XML Message
	The algorithm connects to the legacy system and retrieves the necessary data. <i>Note: the connection is not automatic; each legacy system requires implementing a specific model-based procedure.</i>	The XML message is received and stored.
9.	XML Generation	XML Reading
	The mapped data is structured into an XML document compliant with the S-Series standards. The sub-steps are: 1. Generation of XML 2. Validation against valid values and xsd 3. Full message validation 4. Verification business rules (Schematron)	The XML message is read and de-structured. The sub-steps are: 1. XML parser 2. Full message validation 3. Validation valid values and type 4. Verification business rules (Schematron)
11.	Output	Data Extraction
	The generated XML message is saved or transmitted as required.	The algorithm connects to the legacy system and stores the extracted data.

4.4 Validation and Verification Processes

In deliverable D4.2, we introduced the Verification and Validation (V&V) processes defined for the three types of translators developed in the project: ontology-based, AI-based, and model-based. Although each translator relies on different underlying technologies, the proposed V&V workflows share a common structure aimed at ensuring functional correctness, semantic accuracy, interoperability, and overall robustness of the translation results across the project use cases.

In this section, we provide a summary of the V&V processes (see D4.2 for more details). Furthermore, how these V&V processes have been applied in the use cases to validate the translators is presented in Section 5.4.

4.4.1 V&V Process for the Ontology-based Translators

The V&V workflow for the ontology-based translator consists of nine phases, starting from requirement verification to final reporting. The process begins with the verification of functional and technical requirements, ensuring alignment with interoperability needs. It proceeds with unit testing of ontology mappings and integration testing using real or synthetic data, aimed at validating the correctness of semantic alignment rules under controlled conditions.

Interoperability is assessed through compatibility validation with consuming systems, focusing on data completeness, structural coherence, and semantic accuracy.

Performance evaluation and semantic consistency validation are carried out in parallel to assess scalability and the ability of the translator to preserve meaning across heterogeneous data models. The translator is subsequently evaluated in end-to-end scenarios reflecting the project use cases, followed by a phase dedicated to collecting user and expert feedback to refine the translation logic. The process concludes with comprehensive documentation and reporting to ensure transparency and traceability of the V&V results.

This process is summarised in Figure 53.

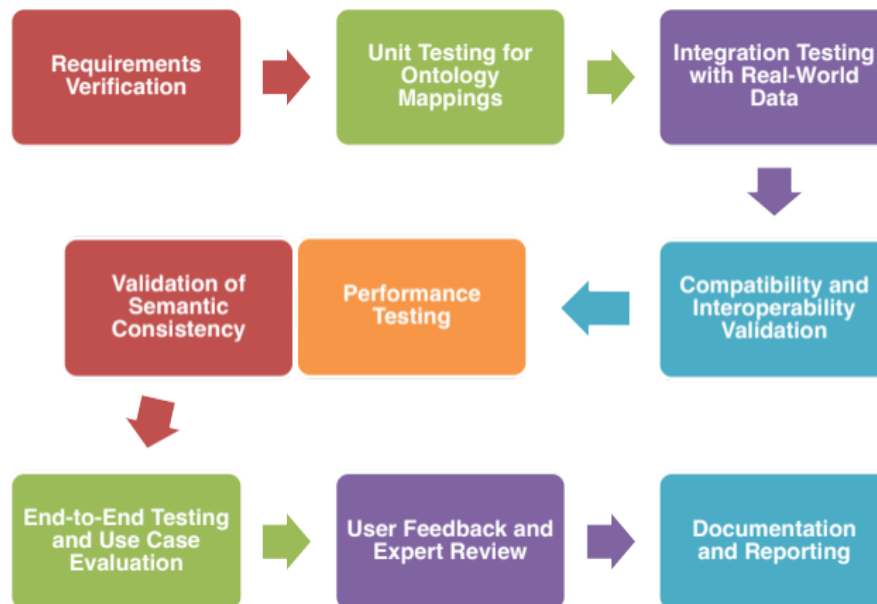


Figure 53: V&V process for the ontology-based translator

4.4.2 V&V Process for the AI-based Translators

The V&V process defined for the AI-based translator is structured into seven phases centred on data quality, model performance, and iterative improvement. The workflow starts with the preparation and evaluation of datasets used for training, testing, and validation, ensuring representativeness and high data quality. The translator model is then trained and fine-tuned using relevant AI approaches—such as LLM-based or neural machine translation models—followed by component-level testing to validate the correctness of internal modules (e.g., context recognition, clustering, label extraction).

End-to-end translation testing evaluates translation accuracy using real-world scenarios inspired by the project use cases, supported by both quantitative metrics (e.g., BLEU, ROUGE) and expert assessments.

Performance and scalability testing verify the system's responsiveness and resource efficiency under different load conditions. A dedicated phase ensures alignment with domain-specific terminology and semantic requirements, involving expert review.

The process concludes with systematic documentation, user feedback collection, and iterative model refinement, allowing continuous improvement of translation accuracy and robustness.

This process is summarised in Figure 54.

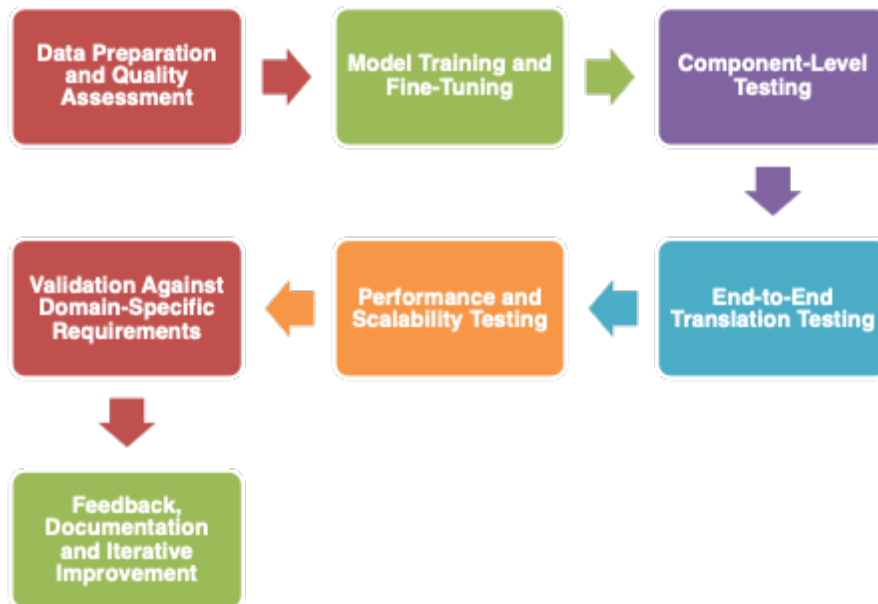


Figure 54: V&V process for the AI-based translator

4.4.3 V&V Process for the Model-based Translators

The model-based translator is validated through an eight-phase process aimed at ensuring correctness of model transformations, consistency of meta-model mappings, and conformance to domain semantics. The process begins with requirement verification, ensuring traceability and comprehensive coverage of functional and technical expectations.

Subsequently, meta-model mapping verification and model consistency testing assess whether model elements, relationships, and constraints are correctly transformed according to the defined mapping rules. Automated tools are used to verify structural coherence, inheritance rules, and inter-element dependencies.

The preservation of semantic integrity, particularly for domain-specific elements, is validated in collaboration with domain experts.

Performance and scalability are evaluated both on individual components and within full end-to-end workflows that reflect real use case scenarios. These tests assess the translator's ability to handle increasing model size and complexity while maintaining acceptable performance levels.

Finally, user and expert feedback inform adjustments to improve usability and translation accuracy, and the workflow concludes with comprehensive documentation of the entire V&V process, including tests executed, issues identified, and achieved results.

This process is summarised in Figure 55.

5 Use Cases Collaboration and Support Activity

The following section presents the current progress of the collaboration between WP4 and the UCs in terms of integrating the translator solutions into industrial scenarios. This section aims to cover UC collaboration that has not already been presented in Section 4. It is important to note that this work will be the focus and continue throughout the remainder of the project.

5.1 UC 1_6

UC 1_6 applies technologies from Arrowhead fPVN, specifically data translation and data interoperability, to improve the quality and efficiency of processes along the life cycle of EV batteries. The focus in the collaboration with WP4 is on providing unified access to data living in different data formats in "silos", especially data from previous tests. The integration of heterogeneous data formats from these tests presents significant challenges.

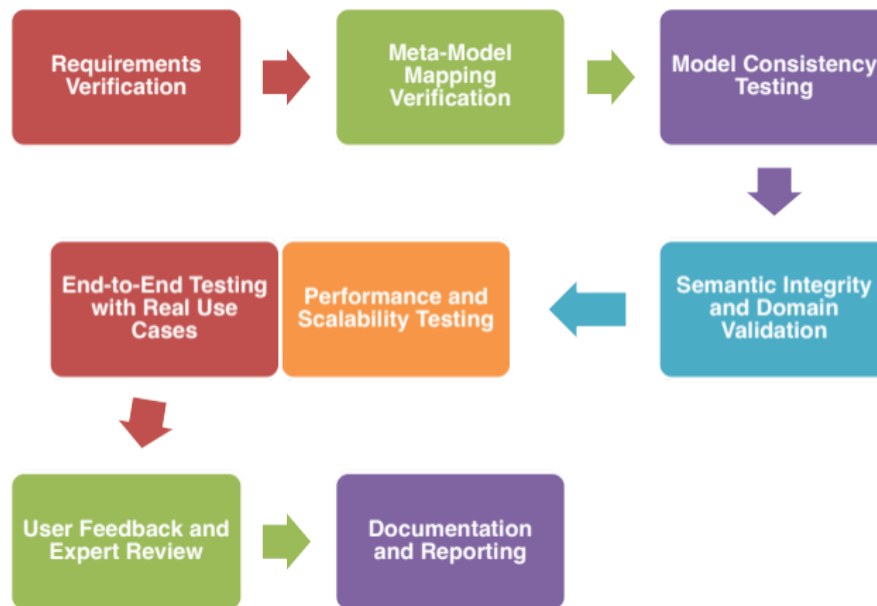


Figure 55: V&V process for the model-based translator

Current approaches suffer from manual mapping, repetitive configuration, and error-prone processes. An AI-based approach for automated data mapping is therefore being developed, leveraging semantic similarity techniques and historical data.

5.1.1 Main Challenge

Battery development and testing require the ingestion of data from diverse sources, often in unstandardised formats. Traditionally, engineers manually map external signals to internal data models, a process that is labour-intensive, prone to errors, and difficult to scale across multiple test facilities. These inefficiencies lead to delays and inconsistencies in data handling, ultimately impacting costs and quality.

Current workflows demand deep domain knowledge of data structures, metadata, and internal models. Each new test setup requires repetitive configuration, increasing complexity and reducing agility. While organisations possess historical data and the capability to ingest common formats, the lack of an automated, user-friendly process remains a bottleneck.

5.1.2 Approach

The prototype developed employs AI-driven semantic analysis to automate data mapping. By computing dense vector embeddings for metadata and parameter names using the **paraphrase-multilingual-mpnet-base-v2** model, the system captures semantic meaning in a 768-dimensional space. Similarity between data points is determined via cosine similarity, enabling robust matching even in cases of non-exact correspondence.

The workflow begins with the user uploading a data file and specifying integration intent. The AI model extracts metadata and norm names, which are then matched against an internal data model. A mapping proposal is generated, allowing user validation before final execution. Confirmed mappings enable automated creation of test configurations, such as load ramps, reducing manual intervention.

The solution builds on **Sentence-BERT (SBERT)** for semantic representation, as described in [16]. This architecture supports multilingual embeddings and facilitates clustering and semantic search. Historical data serves as a reference for parameter translation, where exact matches are applied directly, and approximate matches are computed through similarity scoring.

A graphic of this approach can be seen in Figure 56.

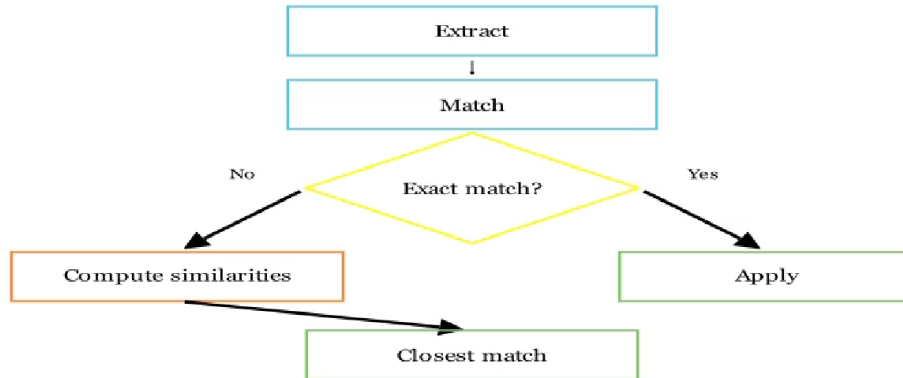


Figure 56: High-level approach for UC 1.7

5.1.3 Validation in the Use Case

In a typical scenario, an engineer uploads a dataset and requests integration with existing test equipment. The system analyses metadata, proposes mappings, and, upon confirmation, generates the required configuration automatically. This approach minimises onboarding time for new equipment and accelerates project execution.

5.1.4 Conclusion

AI-based semantic mapping represents a significant advancement in test data management. A screenshot from an example validation run is Figure 57.

	InputAttributeName	TargetAttributeName	ApplicationElement	CustomMapping	MappingType
1	PROJEKT	iName	Project	@(Project)	Closest Match
2	VER_REIH	rUnit	MeasurementAttr	(rUnit)	Closest Match
3	VER_E_NM	iSegmentNr	ExternalComponent	(iSegmentNr)	Closest Match
4	VERSIONV	softwareVersion	ETLTestBed	(softwareVersion)	Closest Match
5	Result Creation Date	iCreationDate	DC_Measurement	(iCreationDate)	Closest Match
6	Test Description	iDescription	Versuch	First("Test Description")	Exact Match
7	Description_Component	Description_Component	Versuch	First("Description_Component")	Exact Match
8	ID_Mounting_Place	ID_Mounting_Place	Versuch	First("ID_MOUNTING_PLACE")	Exact Match
9	Part_Nr	Part_Nr	Versuch	First("PART_NR")	Exact Match

Figure 57: Validation of the semantic mapping of UC 1.7

5.2 UC 1.7

This use case demonstrates the application of Model-Based Enterprise (MBE) principles to enable interoperable and intelligent management of Product Value Networks throughout the full life cycle of electromechanical components for aerospace. The primary objective is to implement digital and automated strategies that enhance manufacturing productivity and ensure comprehensive traceability of all assets across design, production, and operational phases. By integrating MBE with standardised modelling and translation tools, the use case supports increased efficiency, reliability, and cross-platform interoperability in aerospace systems engineering.

Figure 58 presents the final architecture of the demonstrators developed within the scope of this use case, illustrating the integration of tools, models, and services across the design and validation workflow.

5.2.1 DiTAG IPC-2581 to Electronic Component Data Model

The DiTAG tool, developed by Task 4.1, performs ontology-based translation from IPC-2581 to the UC 1.7 Electronic Component Data Model. This approach enables semantic alignment between standardised design

Use Case Description

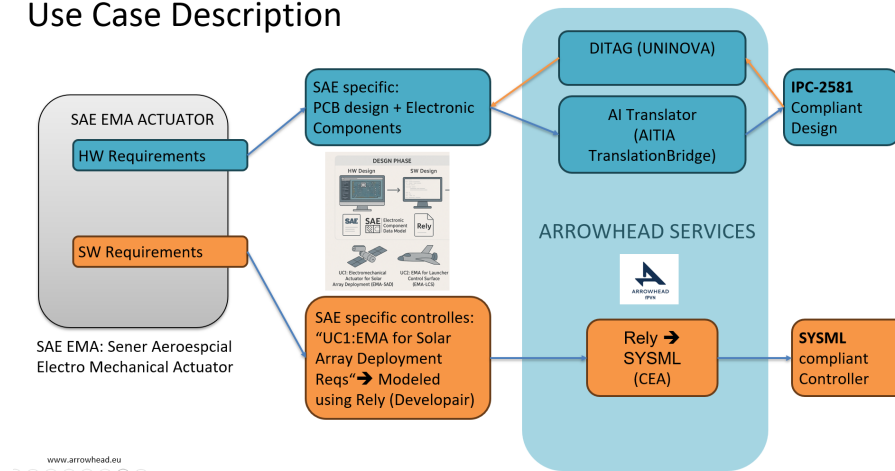


Figure 58: Use-case 1.7 description

formats and domain-specific models, facilitating consistent data exchange and integration.

5.2.2 Electronic Component Data Model to IPC 2581 translation

The task 4.2 translator uses AI techniques to support the reverse translation—from the UC1.7 Electronic Component Data Model back to IPC-2581. This bidirectional capability expands the scope of model interoperability and strengthens collaboration with task 4.2 in the context of intelligent data transformation.

The AI tool provided by task 4.2 for Electronic Component Model translation has moved beyond the planning stage. It is now part of the ongoing activities, supporting the ontology-based translation process. While it was not included in the earliest prototype, it is currently being integrated alongside IPC-2581 and UC 1.7 data models, strengthening collaboration with task 4.2 and expanding the scope of translation capabilities.

WP4 applied the same model as presented in Section 4.2.4. The following examples show samples from the relevant steps.

```
<EPPL_Family>01 CAPACITORS</EPPL_Family>
<EPPL_Group>02 CERAMIC CHIP</EPPL_Group>
<Quality_Type>F</Quality_Type>
<Description>Capacitor, Ceramic Chip, CC-I, C0603CF, 0.3pF, 0.05pF, 250V</Description>
<SPN>E01001027.F01</SPN>
<Code_Eng_COM>E0102M1341A03</Code_Eng_COM>
<Code_Old_COM>E0102M1341A03DLF</Code_Old_COM>
```

Figure 59: Sample from an input EC XML file. Appendix/EC_input1.xml presents a full example EC XML file.

```
<Avl name="DEFAULT">
  <AvlHeader title="U" source="U" author="U" datetime="2025-10-16T08:44:40.706518+00:00" version="1" comment="U" modRef="USERDEF"/>
  <AvlItem OEMDesignNumber="E01001027.F01">
    <AvlVmpn evplMpn="E0102M1341A03DLF" evplVendor="01-CAP-CERAMIC" qualified="false" chosen="false">
      <AvlMpn other="C06CF0R3A-9UN-C0TC0T" name="DEFAULT" rank="0" cost="0.0" moistureSensitivity="UNLIMITED" availability="false"/>
      <AvlVendor enterpriseRef="U"/>
    </AvlVmpn>
  </AvlItem>
</Avl>
```

Figure 60: Sample from a translated (output) IPC 2581 XML file. Appendix/IPC2581_translatedoutput.xml presents an example IPC 2581 XML file.

Table 11: Chunk examples for each specific RAG. Each row is one chunk within the RAG vector store.

1st level XML Element	Element name	Attribute name	Description from the standard
AVL	Avl	name	A unique name assigned to a group of approved sources of supply for the materials used in building the electronic assembly.
AVL	AvlVmpn	qualified	An enumerated string is either "true" or "false" (part of the 3WC standard – must be lower case). True equals that the external vendor part library has been qualified; False indicates that it has not been qualified. If the attribute is not present, the qualification is unknown.
BOM	BomItem	category	The category of the BomItem as an enumerated string being either ELECTRICAL — PROGRAMMABLE — MECHANICAL — MATERIAL — DOCUMENT.
BOM	Enumerated	enumeratedCharacteristicName	A unique name applied to the characteristic.
Content	LayerRef	name	The unique name of all Layer segment descriptions.
ECAD	Property	refValue	A second Value number that is to be used as a reference Value, considering the properties of the basic attribute requirement.
LogisticHeader	Role	Authority	The access level associated with this role as defined by the systemreferenced by externalConfigurationControlEntryPoint.

Table 12: Examples from the source data model tag descriptions. `Appendix/TagDefinitions.csv` presents all tag descriptions regarding the source data model.

Element name	Description
SPN	Sener Part Number (Unique ID). Data type: Identifier in string format. E.g., E01001027.E01.
Engineering Package	Type of the package that has the component (physical way to package the component, it is represented as an id) data type: Identifier (string). E.g., C0603.
Moisture sensitivity level	Predefined classes for moisture sensitivity level (e.g. None, 1, 2, 3) Unit: g. Data type: integer.
Weight	Weight of the component.

Table 13: Examples from the EC – IPC 2581 mapping table. Appendix/EC-IPC.MappingTable.csv presents the AI-defined mapping table.

EC Element name	IPC 2581 Element name and attribute name
SPN	IPC-2581/Avl/AvlItem.OEMDesignNumber
Engineering Package	- (Could not be mapped)
Moisture_sensitivity_level	Avl/AvlItem/AvlVmpn/AvlMpn.moistureSensitivity
Weight	Ecad/CadData/Step/Component.weight
Value	Ecad/CadData/Step/Package/SilkScreen/ NonstandardAttribute.value

5.2.3 ISO 10303 translation

As another relevant standard, WP4 also investigated ISO 10303 regarding translation possibilities. We found the following relevant data sources to work with ISO 10303 and with the EXPRESS language: <https://stepcode.github.io/> and <https://github.com/stepcode/stepcode/tree/develop>

STEPCode tries to collect relevant C++ and Python applications to parse EXPRESS, test and build schemas. The github project of STEPCode summarises the relevant EXPRESS schemas.

To apply the AI-based mapping table generation methods, we also have to define descriptions for the relevant parameters. As a first try, we applied the schema definitions from the following sources:

- Geometry schema: https://www.steptools.com/stds/smrl/data/resource_docs/geometric_and_topological_representation/sys/4_schema.htm
- Topology schema: https://www.steptools.com/stds/smrl/data/resource_docs/geometric_and_topological_representation/sys/5_schema.htm
- Geometric model schema: https://www.steptools.com/stds/smrl/data/resource_docs/geometric_and_topological_representation/sys/6_schema.htm
- Scan data 3d shape model schema: https://www.steptools.com/stds/smrl/data/resource_docs/geometric_and_topological_representation/sys/7_schema.htm

Table 14: Examples from the EC – ISO 10303 Mapping Table. Appendix/ISO10303.MappingTable.csv presents the AI-defined mapping table.

EC Element name	ISO 10303 name
Commercial Part Number	label
Generic Part Type	topology_schema
Package	shell
Weight	weights_positive
Sim Netlist	wire_shell

5.2.4 Rely to SysML

The translation from the Rely tool to SysML has progressed into active implementation. In collaboration with Task 4.3, this model transformation workflow is being integrated into the Papyrus environment, enabling automatic conversion of Rely inputs into SysML for system modelling and validation. The process is aligned with the Arrowhead Requirements Model, which complements SysML by supporting structured requirements management and traceability across Cyber-Physical Systems (CPS). In this setup, the translation pipeline enables the automatic conversion of Rely inputs into SysML for system modelling and validation, while the

Arrowhead Requirements Model provides the complementary framework for requirements management and traceability.

In the next text, we are going to explain the steps to follow during this process:

- Component specification in the rely-based tool:
 - We specified a software component within the rely tool.
 - The **inputs and outputs** of the component were defined.
 - We also captured the **requirements** describing the expected **behaviour** of the component

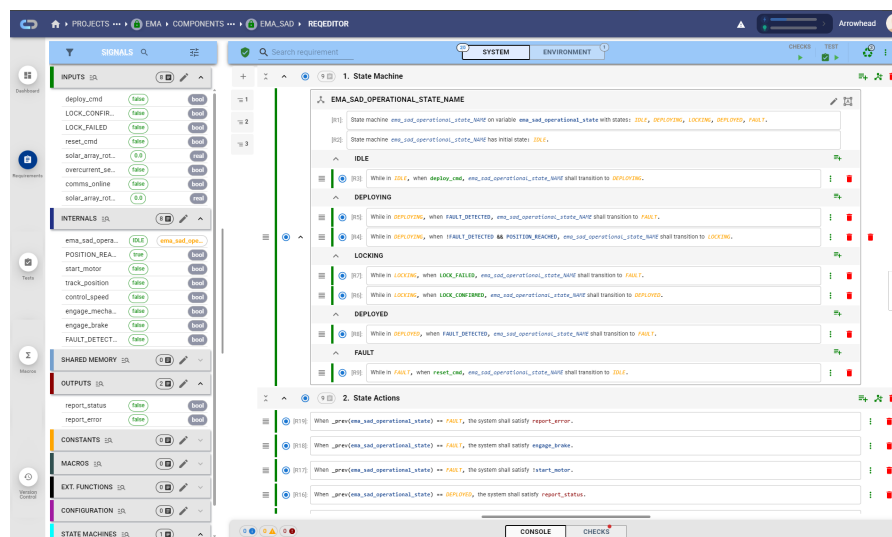


Figure 61: Rely tool GUI

5.2.4.1 Outcome

A structured component definition using the Rely language.

- SysML Integration:
 - A **translation between Rely and SysML** was created, designed to **maximise interoperability** between both languages and retain all the information possible.
 - A **connector** was developed to **transform component definitions and requirements** from **Rely** into **SysML format**.
 - This allows **integration with tools that support SysML diagramming** and keeps **traceability with the rely tool**.

5.2.4.2 Ongoing Developments

- **Fully translate** all types of requirements existing in the tool.
- **Translate state machines** from **Rely** into **SysML** formalism.
- **Implement a bidirectional connector** to enable transformation from Rely to SysML and from SysML back to Rely.

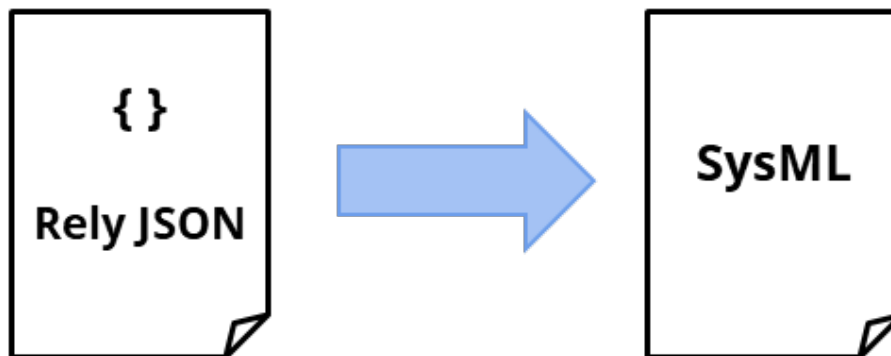
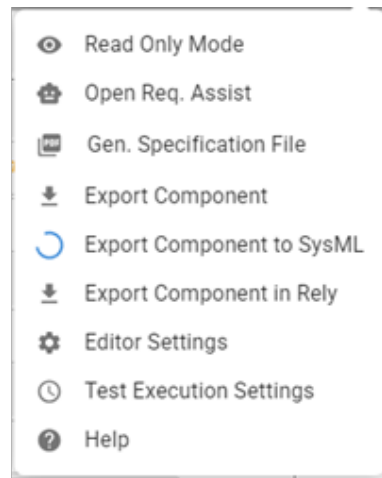


Figure 62: Rely to SysMLv2 process.

5.2.4.3 Goal

Achieve interoperability between the Rely and SysML tools, ensuring consistency and traceability across both environments.

This modular translation strategy—combining ontology-based, AI-driven, and model transformation techniques—ensures robust interoperability across design, requirements, and validation workflows. It lays the foundation for scalable, standards-compliant integration in aerospace systems engineering.

5.3 UC 2_9

In the Process industry demonstrator in WP9, there is a need to transform the cardboard mill Process and Instrumentation diagrams from dumb drawing format into intelligent Dexpi XML format. Furthermore, the intelligent Dexpi diagrams are further transformed into a Graph format in a graph database. This way, it is possible to carry out intelligent queries from the graph database.

Stora Enso has selected the pilot process from one of its cardboard mills. WP4 has developed ontology and rule-based transformation methods to convert the old drawings. This approach has been applied to the P&IDs, and the resulting Dexpi files are handed over to other demonstration partners. Demonstration implementation is still ongoing, but the first results look promising.

5.4 V&V Process Adoption in the UCs

This section presents how the Verification and Validation (V&V) processes defined in WP4 have been applied within the project's use cases to assess the translators developed in WP4 and deployed in WP6, WP7, and WP9. This section focuses on the V&V processes' practical adoption, demonstrating how each use case has followed the relevant phases to verify functional correctness, semantic accuracy, performance, and interoperability of the translation solutions.

WP6 and WP7 have already started the V&V activities, which are still ongoing, while WP9 will start shortly.

5.4.1 UC 1.6

This section describes the V&V process for AI-Based Data Translation for Battery Test Labs (UC1_6 in WP6). This use case focuses on the full life cycle of batteries for electric vehicles, and how data access can improve selected phases thereof. During battery cell design and production, for example, experimentation and test labs create huge amounts of data, unfortunately, often from different systems in incompatible data formats.

Making that trove of information available allows us to better understand the behaviour of prototype and production cells. This is expected to significantly improve the processes and workflows of running the test labs. That directly translates to clear cost reductions.

The core problem lies in the heterogeneity of data formats produced by different test systems and vendors. Traditionally, engineers have relied on manual mapping of external signals to internal models, a process that is time-consuming, error-prone, and difficult to scale. To overcome these limitations, an AI-based translator has been developed in UC1_6 together with WP4, which leverages semantic similarity techniques to automate data mapping. This document explains how the seven validation and verification steps defined have been applied to this solution.

5.4.1.1 Data Preparation and Quality Assessment

For UC 1_6, data was collected from multiple battery test systems, each using different vendor-specific formats such as XML, JSON, and proprietary structures. The preparation process involved cleaning and normalising metadata, resolving inconsistencies in parameter naming, and filling missing values where possible. Historical datasets were analysed statistically to ensure they capture at least a representative range of variations in naming conventions, measurement units, and test conditions.

In UC1_6, we did not train a model from the ground up but used data to provide context for an existing semantic mapping model, as well as for the test cases.

In a future step, augmentation techniques could be considered to simulate edge cases, such as uncommon parameter names or incomplete metadata, thereby improving the resilience of the translator.

5.4.1.2 Model Training and Fine-tuning

UC1_6 relies on Sentence-BERT embeddings to compute semantic similarity between vendor-specific parameters and the internal data model. By using this pre-trained model, only fine-tuning was required. Domain-specific terminology provided the appropriate context for EV battery testing. Another step was the iterative optimisation of similarity thresholds to minimise false matches while preserving flexibility for approximate mappings.

Domain-specific augmentation was applied to improve recognition of critical parameters such as state of charge (SOC), C-rate, and temperature profiles. The model was evaluated after each iteration to ensure that semantic integrity was maintained, and adjustments were made to embedding dimensions and similarity scoring strategies to achieve optimal performance.

5.4.1.3 Component-level Testing

Before integrating the system, each functional component of the UC1_6 prototype (written in Python) was validated independently.

The metadata extraction module was tested to confirm that it correctly identified all relevant attributes from diverse file formats.

The similarity computation engine was evaluated against a curated set of known mappings to verify that cosine similarity scores aligned with expected semantic relationships.

The mapping proposal generator was examined for logical consistency, ensuring that suggested mappings were coherent and aligned with domain standards.

5.4.1.4 End-to-end Translation Testing

Once the components were integrated into the Python script representing our prototype, the complete workflow was tested using real-world datasets from multiple vendors. Files representing typical battery test scenarios were fed as inputs, and generated mapping proposals were reviewed by a domain expert. The accuracy of these mappings was compared against manually created reference mappings to assess performance. Quantitative metrics such as semantic match scores were recorded, and qualitative evaluations were conducted to identify subtle errors in interpretation.

5.4.1.5 Performance and Scalability Testing

Performance and scalability tests were left for a later phase.

5.4.1.6 Validation Against Domain-specific Requirements

In UC 1_6, a domain expert reviewed mappings for critical parameters to confirm that essential meanings were preserved. For example, the system had to correctly interpret variations such as "Voltage_Max" and "U_max" as equivalent concepts.

5.4.1.7 Feedback, Documentation and Iterative Improvement

As the prototype is still being evaluated, feedback and iterative improvements are expected to follow later.

5.4.2 UC 1.7

Use-case 1_7, "Interoperable intelligent management of production lines", relies on an ontology-based translator (DITAG) and follows the process illustrated in Section 4.4.1 for the validation and verification of the translator.

SENER Aeroespacial also adopted an AI-based translator and a model-based one, but the V&V activities will be reported in a later stage.

5.4.2.1 Requirements Verification

DITAG translator must ensure full interoperability between the standardised data model used as input and the internal data structures required by SENER Aeroespacial for the definition of electronic circuit projects.

During this phase, the verification activity focuses on confirming that the translator implementation satisfies all functional and technical requirements defined for data ingestion, semantic interpretation, and output generation.

SENER Aeroespacial established two primary requirements:

1. Input data conformance:

The translator must correctly ingest and interpret input files fully compliant with the IPC-2581 standard. This implies verifying that the translator can parse the structure, elements, and annotations defined in IPC-2581 and correctly map them to the corresponding ontology concepts. Compliance with IPC-2581 ensures that information related to electronic components, PCB stack-ups, layout details, and manufacturing specifications can be semantically aligned within the ontology.

2. Output data conformance:

The output generated by the translator must fully comply with the SENER Aeroespacial Data Model for electronic circuit projects. This includes validating that the ontology-driven translation produces structured data representing electronic components and PCB descriptions in the exact format expected by SENER's internal engineering tools. Structural consistency, completeness of semantic elements, and preservation of domain-specific meaning must be ensured during the verification.

In this phase, the review activities therefore include:

- checking that all functional requirements related to semantic alignment and data structure compatibility are met,
- validating that the ontology mappings cover all necessary IPC-2581 concepts relevant to the use case,
- confirming that the translator supports the annotations and metadata needed for downstream processing in SENER's environment.

5.4.2.2 Unit Testing for Ontology Mapping

In this phase, unit tests are executed to validate the correctness of the individual ontology mappings and translation rules implemented in the DITAG translator. SENER Aeroespacial will prepare a set of synthetic IPC-2581 test files specifically designed to verify that each required element, attribute, and semantic relation is correctly interpreted and transformed according to the SENER-specific Data Model.

These synthetic test files will contain the minimum set of information needed to exercise the core mapping rules, while also enabling the evaluation of edge cases such as incomplete structures, optional elements, and mismatched data types. By testing each mapping in isolation, this phase ensures that the foundational translation mechanisms operate accurately before progressing to more integrated validation steps.

This verification confirms that the DITAG ontology-based translator is correctly specified to operate within Use Case 1_7 and that its implemented functionality corresponds to the interoperability requirements defined by SENER Aeroespacial.

5.4.2.3 Integration Testing with Real-world Data

In this phase, the DITAG translator is evaluated in a controlled environment to verify that the complete translation workflow, from IPC-2581 input to the SENER-specific Data Model output, operates correctly when integrating the individual mapping components tested in Phase 2.

SENER Aeroespacial will use the outputs generated during the unit testing phase and validate them through end-to-end translation in a laboratory setting, employing simple yet representative examples. The objective is to confirm that the translator produces output files that fully comply with the requirements of the SENER Data Model, ensuring structural correctness, semantic consistency, and appropriate handling of annotations.

This integration testing step demonstrates that the translator performs as expected when processing coherent datasets that resemble real operational conditions, thereby establishing readiness for more complex interoperability and performance evaluations in subsequent phases.

5.4.2.4 Compatibility and Interoperability Validation

This phase typically aims to verify that translated data is interoperable with multiple consuming systems and that the output can be correctly processed without additional adjustments. However, in the context of Use Case 1_7, the DITAG translator is designed to generate output exclusively for a single consumer, the SENER Aeroespacial engineering environment, which adheres to a well-defined and proprietary Data Model.

Since all translation outputs are specifically tailored to this unique target system, interoperability testing across multiple consumers is not required. Compatibility with the SENER Data Model is already assessed in earlier phases, particularly through requirements verification (Phase 1) and integration testing (Phase 3).

As a result, this phase is considered not applicable for Use Case 1_7.

5.4.2.5 Performance Testing and Validation of Semantic Consistency

The fifth phase of the process is composed of two steps, which are carried out in parallel, as they evaluate complementary dimensions of the DITAG translator: its operational performance and its ability to preserve semantic meaning during translation. Both dimensions must be validated before progressing to end-to-end testing.

Performance testing: In this step, the efficiency and scalability of the DITAG translator are assessed.

SENER Aeroespacial will generate multiple IPC-2581 input files representing different electronic project definitions, varying in size and complexity. These datasets will be used to measure key performance indicators such as response time, memory consumption, and CPU utilisation during translation.

The objective is to verify that the translator operates within acceptable thresholds under the expected operational load and that its performance scales appropriately with increasing data volume.

Validation of semantic consistency: This sub-phase evaluates whether the translator preserves the semantic meaning and domain intent of the input data throughout the transformation process.

SENER Aeroespacial will reuse the same IPC-2581 datasets created for the performance tests and examine whether the resulting outputs consistently reflect the expected SENER Data Model. The validation will confirm that all semantic elements are correctly represented, relationships are preserved, and the resulting files adhere to the required format.

This ensures that the translator not only performs efficiently but also maintains full semantic integrity across different input scenarios.

5.4.2.6 End-to-end Testing and Use Case Evaluation

In this phase, the DITAG translator is validated in a complete end-to-end workflow that mirrors the operational conditions of Use Case 1_7, ensuring correct integration within the Arrowhead ecosystem and proper execution of all translation steps, from input ingestion to final output delivery.

For the demonstration scenario, SENER Aeroespacial will select a representative electronic circuit design, such as a subsystem of an electromechanical actuator for launcher control surfaces. The corresponding IPC-2581 file describing the hardware will be used as the input to the translator. The expected output is the complete and correctly structured representation of the same hardware according to the SENER-specific Data Model.

The translator will be deployed as a *Producer service* within the Arrowhead platform, responsible for receiving IPC-2581 input files and returning translated outputs. A dedicated Consumer service will be implemented to send compliant hardware descriptions to the translator and retrieve the generated SENER Data Model output.

This end-to-end evaluation will confirm that:

- the translator operates correctly as part of an Arrowhead service-oriented architecture,
- the translation workflow functions seamlessly under realistic conditions,
- all system components, input provisioning, translation engine, and output consumption, interact as intended.

Successful completion of this phase demonstrates the readiness of the translator for use within the full system context of Use Case 1_7.

5.4.2.7 User Feedback and Expert Review

This phase focuses on gathering qualitative feedback from end users and domain experts to refine and validate the DITAG translator beyond technical correctness. The final translated outputs, generated during the earlier V&V phases, will be shared with the electronic design engineers responsible for the corresponding hardware development activities.

The designers will review the translated information to confirm that it is accurate, complete, and consistent with their expectations and workflow requirements. Their assessment will help identify any issues related to semantic interpretation, structural representation, or usability of the generated SENER Data Model files.

Feedback gathered during this review will be used to guide iterative refinements, if necessary.

5.4.2.8 Documentation and Reporting

The final phase ensures that the entire V&V process for the DITAG translator is fully documented, traceable, and auditable. For Use Case 1_7, SENER Aeroespacial will record the inputs, outputs, and procedures followed in each phase of the validation workflow. This includes documenting all test cases, the corresponding results, any issues encountered during the process, and the corrective actions implemented to resolve them.

5.4.3 UC 2.7

Use case 2_7, “AHMS for Trend Monitoring, Predictive Maintenance and Fleet Operations & Maintenance Simulation”, addresses the data flow between stakeholders in the Aerospace Industry, which is very often not standardised and subject to continuous changes over time, leading to slowdowns and possible errors during data transcription on the local repository.

To solve this issue, the ASD S-SERIES Specifications provides guidelines about how to establish an Integrated Product Support program and provide the means to exchange data between the different specifications, different partners and between contractors and customers. This results in a messaging specification used to define the structure and format of messages exchanged, where the structure might be represented using UML and then data converted into an XML message.

In accordance with the S-Series, it has been chosen to adopt the UML as a reference. The UML consists of a set of different modelling techniques, of which the S-Series IPS specifications use only one, the class model. A class model defines a static view of the information (data) that is needed to support the business processes.

In detail, S5000F, one of the S-SERIES Specifications, specifies information and data elements required to provide in-service data feedback. Though originating mainly from the aerospace domain, S5000F can be

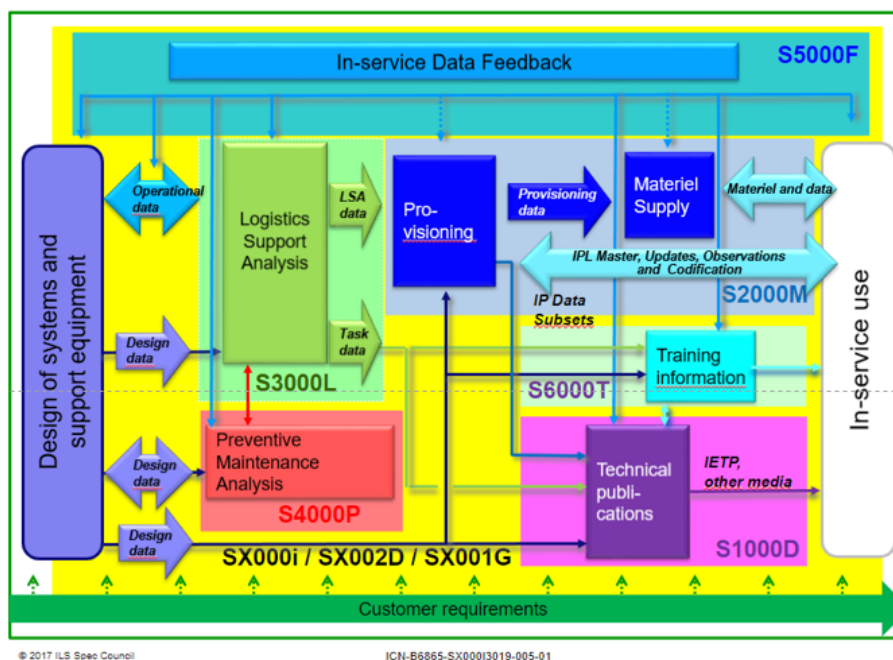


Figure 63: S-Series Suite of Integrated Product Support Specifications

applied to any products, whether mobile or not, in the air, land, sea and space domains, both for civilian and military programs.

5.4.3.1 Why use S5000F?

The advantages of using S-Series, as a common transfer data protocol, are listed below:

- Scalable: can be implemented on selected use cases and then progressively extended
- Integrable: since usage is limited to data exchange, it can be easily integrated with existing legacy systems through import/export functions
- Interoperable: fully interoperable with other S-SERIES standards (S1000D, S2000M, S3000L, S4000P, etc.)
- Relational: the data model provides the information about how the data received is related to each other, supporting warehousing and analytics activities.
- Agnostic: the implementation of S5000F does not require changing the existing legacy IT systems. It may require some adjusting of the internal processes for data validation. S5000F does not mandate how to exchange the data information, only the format, so it can be used with web services, an ad-hoc protocol, FTP, e-mail, etc.

The relevant translation system for UC 2.7 is presented in Section 4.3.12.

5.4.3.2 Use Cases Developed

At this stage, in Use Case 2.7, they have been identified 3 Use Cases to transmit the In-Service feedback regarding the aircraft activities. The following Use Cases (UC) have been identified, tailored, and tested:

- Fleet availability message (UC 50000) – this is for testing purposes only
- Removals data (UC 50403)
- Fault Code data (UC 50902)

Anonymised values of real in-service data are reported in the following sections.

5.4.3.3 Fleet Availability

This UC aims to transmit the availability of each aircraft at a specific time and the reason for its condition.

The UML is reported in Figure 64, and it is translated into an XML Schema Definition (XSD), which defines the structure of S5000F messages in XML format. Each message must be represented by an XML schema following the structure defined in the UML model.

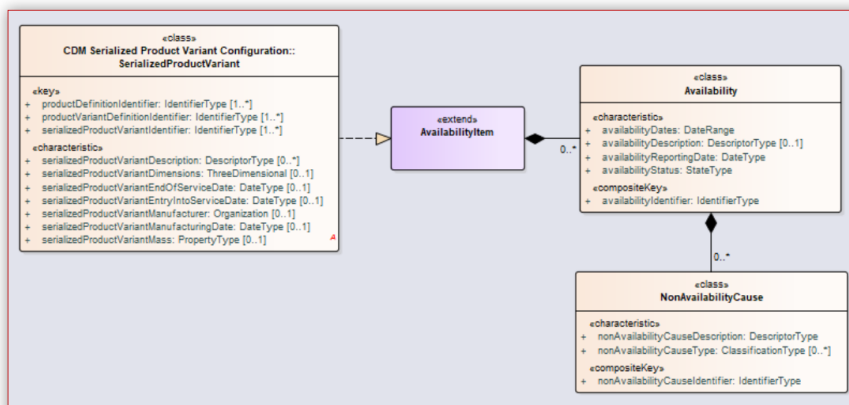


Figure 64: Example of a UML, Fleet availability

A mapping between the raw data and S5000F elements has been performed to produce the XML message, as shown in Table 15.

5.4.3.4 Data Removal

This UC aims to transmit the information about the equipment removals and their supplier feedback after the repair.

The UML is reported in Figure 65.

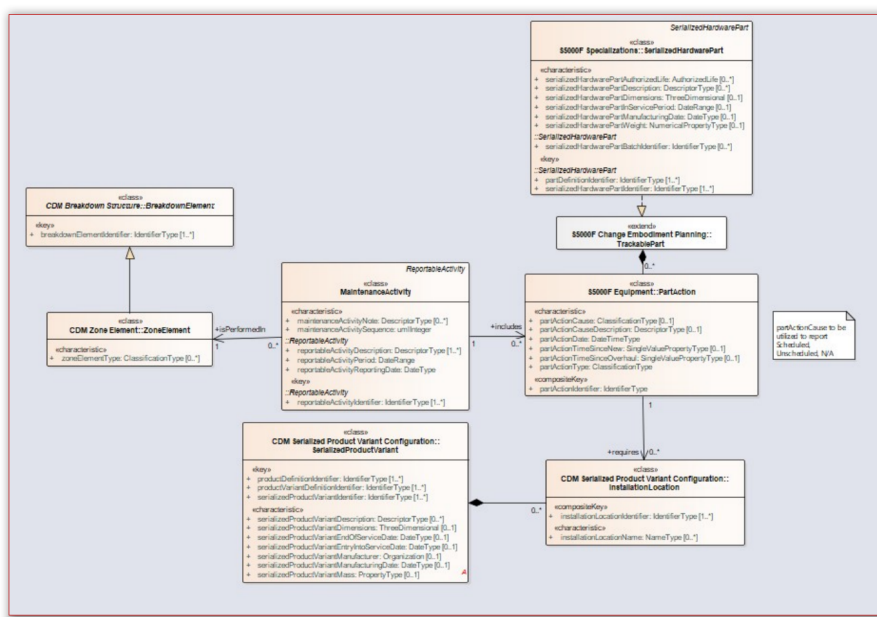


Figure 65: Example of a Use Case, Supplier Work Report

A mapping between the raw data and S5000F elements has been performed to produce the XML message (See Table 16).

Table 15: Attributes mapping example, Fleet availability

Type	S5000F Path	Field
UC	uc50000	-
Group	uc50000/facilityNonAbstractClasses	-
Table	uc50000/facilityNonAbstractClasses/opBase	-
Attribute	uc50000/facilityNonAbstractClasses/opBase/@uid	-
Field	uc50000/facilityNonAbstractClasses/opBase/fcltyId/id	Field 1
Field	uc50000/facilityNonAbstractClasses/opBase/fcltyId/setBy/orgId/id	Field 2
Field	uc50000/facilityNonAbstractClasses/opBase/fcltyName/name	Field 3
Table	uc50000/serialPV	-
Attribute	uc50000/serialPV/@uid	-
Field	uc50000/serialPV/prodId/id	Field 4
Field	uc50000/serialPV/prodVarId/id	Field 5
Field	uc50000/serialPV/serPVId[1]	-
Field	uc50000/serialPV/serPVId[1]/id	Field 6
Field	uc50000/serialPV/serPVId[1]/setBy/orgId/id	Field 7
Field	uc50000/serialPV/serPVId[2]	-
Field	uc50000/serialPV/serPVId[2]/id	Field 8
Field	uc50000/serialPV/serPVId[2]/setBy/orgId/id	Field 9
Table	uc50000/serialPV/spvob/opBaseRef	-
Attribute	uc50000/serialPV/spvob/opBaseRef/@uidRef	-
Subgroup	uc50000/serialPV/flta	-
Table	uc50000/serialPV/flta/avail	-
Field	uc50000/serialPV/flta/avail/avId/id	Field 10
Field	uc50000/serialPV/flta/avail/avDate/startDate	Field 11
Field	uc50000/serialPV/flta/avail/avDate/endDate	Field 12
Field	uc50000/serialPV/flta/avail/status/state	Field 13
Table	uc50000/serialPV/flta/avail/navcau	-
Field	uc50000/serialPV/flta/avail/navcau/causeId/id	Field 14
Field	uc50000/serialPV/flta/avail/navcau/causeType/code	Field 15
Field	uc50000/serialPV/flta/avail/navcau/descr/descr	Field 16
Field	uc50000/serialPV/flta/avail/navcau/navCause/servBuRef/docId/id	Field 17

5.4.3.5 Fault Code Data

The aim of this UC is to transmit the information about the fault that occurred during each flight. The UML is reported in Figure 66.

A mapping between the raw data and S5000F elements has been performed to produce the XML message (See Table 17 and 18).

The remaining part of this Section describes the application of the Verify and Validate (V&V) process for model-based translators defined in D4.2 to the Case Study 2.7 translator. It reports a set of concrete steps to verify and validate the translator's functionalities and the results obtained from its adoption in the project use case.

Figure 55 illustrates the V&V from D4.2, applied to the model-based translator AeroBridge. It is composed of

Table 16: Attributes mapping example, Supplier Work Reports-Report Shop Findings

Type	S5000F Path	Field
UC	uc50403	-
Group	uc50403/breakdownElementNonAbstractClasses	-
Table	uc50403/breakdownElementNonAbstractClasses/beZone	-
Attribute	uc50403/breakdownElementNonAbstractClasses/beZone/@uid	-
Field	uc50403/breakdownElementNonAbstractClasses/beZone/beld/id	Field 1
Field	uc50403/breakdownElementNonAbstractClasses/beZone/beName/name	Field 2
Table	uc50403/mact	-
Attribute	uc50403/mact/@uid	Field 3
Field	uc50403/mact/actId/id	Field 4
Field	uc50403/mact/actRepDate/date	Field 5
Table	uc50403/mact/pactRef	-
Attribute	uc50403/mact/pactRef/@uidRef	-
Table	uc50403/mact/beZoneRef	-
Attribute	uc50403/mact/beZoneRef/@uidRef	-
Table	uc50403/pact	-
Attribute	uc50403/pact/@uid	-
Table	uc50403/mact/maintItem/serialPartRef	-
Attribute	uc50403/mact/maintItem/serialPartRef/@uidRef	-
Field	uc50403/mact/party/orgRef/orgId/id	Field 6
Field	uc50403/pact/actId/id	Field 7
Field	uc50403/pact/instLocRef/prodId/id	Field 8
Field	uc50403/pact/instLocRef/prodVarId/id	Field 9
Field	uc50403/pact/instLocRef/serPVId/id	Field 10
Field	uc50403/pact/instLocRef/instLocId/id	Field 11
Group	uc50403/serializedHardwarePartNonAbstractClasses	-
Table	uc50403/serializedHardwarePartNonAbstractClasses/serialPart	-
Attribute	uc50403/serializedHardwarePartNonAbstractClasses/serialPart/@uid	-
Field	uc50403/serializedHardwarePartNonAbstractClasses/serialPart/partId/id	Field 12
Field	uc50403/serializedHardwarePartNonAbstractClasses/serialPart/serialId/id	Field 13
Field	uc50403/serializedHardwarePartNonAbstractClasses/serialPart/descr/descr	Field 14
Field	uc50403/serializedHardwarePartNonAbstractClasses/serialPart/flta/avail/avId/id	Field 15

8 phases, from the requirements V&V to the documentation of the entire V&V process and of its results. This is further described in Table 19.

5.4.3.6 Requirement Verification

A set of high-level requirements has been defined to develop the AeroBridge translator, mainly functional and relevant to compliance with S-SERIES S5000F process and Data Model, like the possibility to import raw .csv files, exporting them in S5000F XML format, or to perform a comparison within the translator between the original and the transformed set of data.

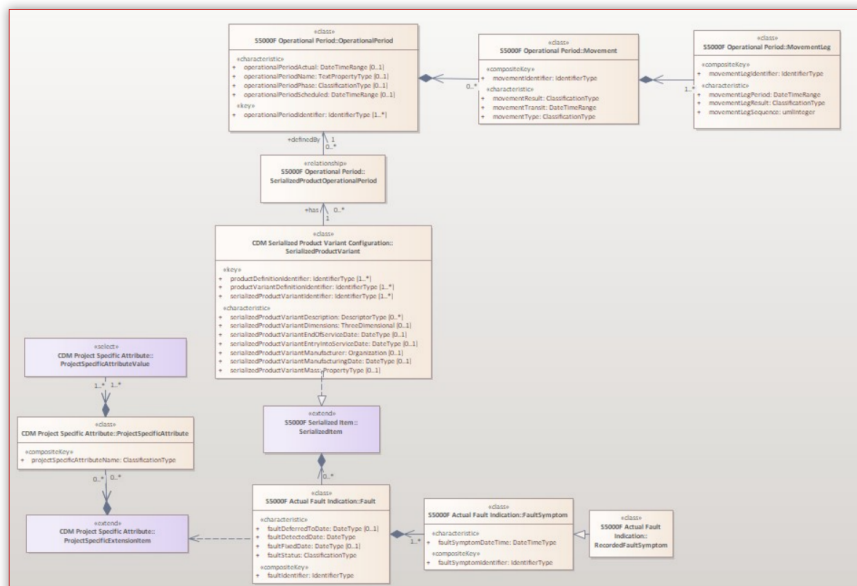


Figure 66: Example of a Use Case, Fault Code

Detailed requirements were provided with respect to the most relevant data exchange use cases identified for Use Case 2.7 (mainly Item removals, Fault Codes and Flight parameters, with an additional testing case relevant to Fleet availability).

At the moment, all the functional and S5000F-relevant requirements have been achieved.

With respect to the data exchange domain-specific requirements, Fleet Availability and Item Removals were fully achieved, while work is ongoing for Fault Codes and Flight Parameters.

Meta-model mapping verification For the most relevant data exchange use cases identified for Use Case 2.7, a mapping was performed with respect to the S5000F Data Model. See Section 4.3.12 for data mapping.

Unit tests for key attributes and elements in the mapping schema have been performed to verify the mapping against the Specification. Moreover, domain experts' reviews are leveraged to confirm semantic consistency.

In detail, we achieved an accuracy for each data exchange model as reported in the following bullets:

- Availability: 100%
- Removals: 100%
- Fault Code: 90%
- Flight Parameters: To be performed

Regarding Fault Codes, work is in progress to reach full accuracy. The issues are due to missing attributes in S5000F, which correspond to legacy data. Such as we need to transmit a timestamp including day and time, but the most appropriate identified S5000F attribute allows us to transmit only the day.

5.4.3.7 Model Consistency Testing

The consistency testing has been included from the design phase of the translator and is performed throughout its development: in order to rigorously assess the semantic preservation of the transformation process, a controlled set of input models characterised by specific structural features, including multiple inheritance and cyclic dependencies, was developed. These models were deliberately selected to challenge the transformation rules and expose potential weaknesses in the handling of complex relationships.

In the S5000F context, it means to adopt attributes "uid" and "uidRef": for example, the relationship between "Field 1" and "Field 6" of UC 50000 (see Table 15) is obtained using the attribute "uid" of element "uc50000/facilityNonAbstractClasses/opBase" and "uidRef" of element "uc50000/serialPV/spvob/opBaseRef";

Table 17: Attributes mapping example, Supplier Work Reports-Report Shop Findings

Type	S5000F Path	Field
UC	uc50902	-
Table	uc50902/opp	-
Attribute	uc50902/opp/@uid	-
Field	uc50902/opp/periodId/id	Field 1
Field	uc50902/opp/mv/movId/id	Field 2
Field	uc50902/opp/mv/movleg/legid/id	Field 3
Table	uc50902/serialPV	-
Field	uc50902/serialPV/prodId/id	Field 4
Field	uc50902/serialPV/sproop/oppRef	-
Attribute	uc50902/serialPV/sproop/oppRef/@uidRef	-
Field	uc50902/serialPV/prodVarId/id	Field 5
Field	uc50902/serialPV/serPVId/id	Field 6
Subgroup	uc50902/serialPV/serlTm	-
Table	uc50902/serialPV/serlTm/eqflt	-
Attribute	uc50902/serialPV/serlTm/eqflt/@uid	-
Field	uc50902/serialPV/serlTm/eqflt/faultId/id	Field 7
Field	uc50902/serialPV/serlTm/eqflt/detDate/date	Field 8
Field	uc50902/serialPV/serlTm/eqflt/status/code	Field 9
Group	uc50902/serialPV/serlTm/eqflt/faultSymptomNonAbstractClasses	-
Table	uc50902/serialPV/serlTm/eqflt/faultSymptomNonAbstractClasses/fltsym	-
Field	uc50902/serialPV/serlTm/eqflt/faultSymptomNonAbstractClasses/fltsym/sympId	-
Field	uc50902/serialPV/serlTm/eqflt/faultSymptomNonAbstractClasses/fltsym/sympId/id	Field 10
Field	uc50902/serialPV/serlTm/eqflt/faultSymptomNonAbstractClasses/fltsym/sympTime	-

this link is generated automatically in Aerobridge. Following the transformation, the XML obtained is systematically verified to confirm that constraints, relationships, and inheritance rules are respected across the transformed models.

This approach allows us to demonstrate that the translation process not only produced valid outputs but also preserved the intended semantics of the original design.

5.4.3.8 Semantic Integrity and Domain Validation

This step has been performed starting from the design phase, since the domain experts contributed to provide requirements for the different models, comparing the S5000F attributes used with legacy fields' vocabularies. Several times were spent by collaborative S5000F and domain experts to identify the correct attribute to map, since S5000F is not only applicable to the Aerospace domain, but potentially to all Product domains.

Terms like *"Aircraft"*, *"Flight number"*, etc. were consistently mapped to their corresponding entities in the target model without semantic drift. For example, the object *"Aircraft"* is mapped using the class *"SerializedProductVariant"*. This class is defined as: an actual product variant which is identified as an individual.

However, a mitigation to this is represented by the Guidance Document that allows for clearly states the needs and mapping of data exchange between all the stakeholders involved in the process. No Guidance Document was produced for this project, since out of scope, but expertise and knowledge were acquired while developing the translator and studying the Specification.

Table 18: Attributes mapping example, Supplier Work Reports-Report Shop Findings (Cont.)

Type	S5000F Path	Field
Field	uc50902/serialPV/serltn/eqflt/faultSymptomNonAbstractClasses/fltsym/sympTime/date	Field 11
Field	uc50902/serialPV/serltn/eqflt/faultSymptomNonAbstractClasses/fltsym/sympTime/time	Field 12
Table	uc50902/serialPV/serltn/eqflt/fltcau	-
Table	uc50902/serialPV/serltn/eqflt/projAttrs	-
Table	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[1]	-
Field	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[1]/attrName	Field 13
Group	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[1]/projectSpecificAttributeValue	-
Field	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[1]/projectSpecificAttributeValue/umlInt/value	Field 14
Table	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[2]	-
Field	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[2]/attrName	Field 15
Group	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[2]/projectSpecificAttributeValue	-
Field	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[2]/projectSpecificAttributeValue/umlInt/value	Field 16
Table	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[3]	-
Field	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[3]/attrName	Field 17
Group	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[3]/projectSpecificAttributeValue	-
Field	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[3]/projectSpecificAttributeValue/umlInt/value	Field 18
Table	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[4]	-
Field	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[4]/attrName	Field 19
Group	uc50902/serialPV/serltn/eqflt/projAttrs/projAttr[4]/projectSpecificAttributeValue	-

Table 19: Short description of the phases of the V&V process for the model-based translator

#	Phase	Description of the Phase applied to AeroBridge
1	Requirements Verification	Ensure alignment of the AeroBridge translator with all functional, technical and domain-specific requirements.
2	Meta-Model Mapping Verification	Confirm that the model accurately maps elements between source data and target S-SERIES S5000F data model
3	Model Consistency Testing	Ensure that the model is consistent with respect to S5000F and in general S-SERIES modelling rules.
4	Semantic Integrity and Domain Validation	Validate semantic correctness and adherence to the aerospace domain
5	Performance and Scalability Testing	Assess system performance and scalability
6	End-to-End Testing with Real Use Cases	Test the solution using actual data exchange needs from Use Case 7.2
7	User Feedback and Expert Review	Collect feedback from users and aerospace domain experts
8	Documentation and Reporting	Document results and produce final reports

5.4.3.9 Performance, Scalability and End-to-end Testing

Performance and Scalability Testing

At the moment, no specific requirements in terms of performance of translation are set for Use Case 2_7, since the project represents a test bench for future Company Solutions and to identify the thresholds to set.

However, to provide an indication for future scalability:

- the time (T1) to translate data from legacy origin to XML file has been identified as a Key Performance Indicator (KPI)
- the time (T2) to validate the translation, through an external Java XSD validator, has been identified as a KPI. It is defined as the time from the start to the end of the validation process.

During this phase, the data exchange use cases have been tested considering very large legacy data (text files .csv), the results are reported in Table 20.

Table 20: Scalability test. T1: time to read CSV data and translate it in accordance with S5000F. T2: T1 + time to validate the XML file.

UC	Size legacy dataset	Size xml file	Time T1	Time T2	Validate
UC50000	0.26MB (2000 rows x 17 columns)	1.8 MB	00:00:04	00:00:30	Yes
UC50403	1.8MB (10000 rows x 20 columns)	11.5 MB	00:00:35	00:00:75	Yes
UC50902 (one FC per flight)	1.0MB (10000 rows x 11 columns)	12.5 MB	00:00:80	00:02:10	Yes
UC50902 (one flight)	1.0MB (10000 rows x 11 columns)	11.5 MB	00:00:73	00:01:40	Yes

End-to-end Testing with Real Use Cases

This phase was devoted to confirming that the translator performs correctly in scenarios inspired by actual data exchange use cases. The main target is to ensure compatibility with downstream applications, so that the Aerobridge algorithm can be integrated into the system architecture and follow Digital Services.

To achieve this, it was simulated translation scenarios were used using actual aerospace in-service data and the developed S5000F-compliant data models. The output was validated against expected results to confirm semantic fidelity and structural accuracy.

As of today, the following three use cases were tested:

- Fleet Availability
- Item Removals
- Fault Codes

Flight Parameters are under development; therefore, they have not been tested yet.

The tested use cases were designed to cover not only standard scenarios but also edge cases that mirrored actual in-service contexts, such as incomplete data or conflicting constraints. In this case, the algorithm correctly returns an error message, and xml file generation process is stopped, as designed.

Compatibility with downstream applications is also to be tested, since the Digital Services are under development; however, no major issues are foreseen since the translation results are in line with the expectations.

5.4.3.10 User Feedback and Expert Review

At the moment, two main users and experts' feedback have been and are currently collected and incorporated at the moment:

- Internal Leonardo feedback
- S5000F Steering Committee feedback

As the internal ones, user feedback was gathered both to improve the usability of the translator and its accuracy, by allowing the domain expert to use the tool and test the XML against the original legacy data.

From the S5000F Steering Committee's standpoint, the process began with e-mails to S5000F expert users. These experts evaluated translation quality and semantic accuracy. Moreover, useful feedback about mapping or suggestions on which S5000F attribute to use were provided. Finally, since Leonardo is part of the S5000F Steering Committee, it was possible to have face-to-face discussions with the S5000F experts.

The collected feedback was analysed and used to adjust mapping rules, refine transformation logic, and use a shared semantics with all major companies. By embedding this iterative cycle into the methodology, the translator continuously improved in both accuracy and usability, ensuring that the transformation process remained aligned with domain standards and responsive to real-world requirements.

Useful feedback was also provided to the Steering Committee to improve the S5000F data model. Finally, since Leonardo is co-owner of some S5000F chapters, some changes to the proposed use cases of the Specification are also under development.

5.4.3.11 Documenting and Reporting

At the end of the project, in D7.4, outcomes of the V&V process will be provided to have an overall view of the results, which for the specific Use Case 2.7 will span from the raw data mapping, to the translation, to the downstream to the Digital Services of the Aircraft Health Management System.

6 Applying Local LLMs

For an overview of hardware requirements for running machine learning models locally, see: [Appendix/3_local_ml_models_20251107.pdf](#).

For an exploration of hardware alternatives for running machine learning models locally using two NVIDIA RTX PRO 6000 Blackwell (96GB) GPUs, see: [Appendix/4_local_ml_hardware_alternatives_20251107.pdf](#).

7 Cross WP Collaboration

WP4 has its efforts on strengthening how system interfaces and integrations are represented and managed. For this, WP4 applies widely used standards such as OpenAPI and AsyncAPI to ensure that service definitions and data exchanges within the Arrowhead framework are clear and consistent. This activity complements the developments from WP2, where the focus has been on microservice architectures and communication mechanisms.

By aligning the modelling efforts in WP4 with the implementation and design work in WP2, WP4 has contributed to a coherent approach to interoperability across system components. This integration enables the automatic discovery and integration of service APIs, forming the basis for autonomous translation and adoption between heterogeneous services. The resulting framework improves the ability of Arrowhead-compliant solutions to interconnect and exchange information more efficiently, reducing the need for manual configuration in future deployments.

WP4 has, throughout this period, continued to maintain a close collaboration with WP3 and the use-case providers through weekly meetings. The main goal of these meetings is to discuss and analyse standard data model structures that are utilised within the use-cases and to examine how to best apply the developed translation services in said use-cases.

Lastly, WP4 has strengthened its collaboration with WP10 and will continue to do so throughout the remainder of the project. Given that WP4 focuses on developing automated data model translation capabilities, this close cooperation is essential in order to ensure exploitation of said translation systems.

By working closely with WP10, which leads the exploitation and digital transformation management activities, we aim to ensure that the technological outcomes of WP4, as well as the broader Arrowhead fPVN results, are

effectively leveraged. Strengthening this collaboration will help to guarantee that the developed solutions are not only technically sound but also aligned with real-world exploitation needs.

7.1 Integration of BEIA's Environmental Monitoring Datasets

WP4 in D4.2 highlights dataset provision for training, testing, and evaluating translation models. In D4.3, the dataset provision is extended using BEIA's comprehensive environmental monitoring datasets, which include:

- real-time sensor streams,
- water quality descriptors,
- air quality and weather data,
- metadata from Libelium and Adcon Telemetry stations.

7.1.1 Dataset Flow (as described in D4.2)

1. **Sensors (Libelium Plug&Sense)**
2. **Meshlium IoT gateway**
3. **InfluxDB storage**
4. **MQTT broker**
5. **Grafana visualization dashboard**

This dataset is used as training and validation material for translation tasks in Tasks 4.1–4.3.

```
1 def evaluate_translation(predicted, ground_truth):  
2     """  
3     predicted: dict mapping variable -> translated attribute  
4     ground_truth: dict mapping variable -> expected attribute  
5     """  
6     total = len(ground_truth)  
7     correct = sum(1 for k in ground_truth if predicted.get(k) == ground_truth[k])  
8     return correct / total if total > 0 else 0
```

This produces quantitative metrics for translation accuracy on environmental datasets.

8 Appendices

The list of appendices for this deliverable is:

appendices/1.Model.Context.Protocol.pdf
appendices/2.AH5.Data.Model.Translation.Integration.20250613.pdf
appendices/3.local_ml_models.20251107.pdf
appendices/4.local_ml.hardware.alternatives.20251110.pdf
appendices/5.NOMS2025-AI.Final.pdf
appendices/6.Planner.paper.FINAL.pdf
appendices/appendix-hasco.pdf
appendices/EC.input1.xml
appendices/EC.IPC.MappingTable.csv
appendices/IPC2581.translated.output.xml
appendices/ISO10303.MappingTable.csv
appendices/TagDefinitions.csv

9 Conclusions

This deliverable (D4.3) presents an updated and comprehensive report on the development, integration, and deployment of **Autonomous Translation Technologies**, capturing the progress achieved from the start of the project to month 30. D4.3 details how prototypes and proposals have been developed, how the key objectives of the project have been further advanced using said solutions, and how support activities continue to support the development of state-of-the-art while further assisting integration into designated use cases.

Over this period, Ontology-based, AI-based, and Model-based translation approaches have continued to mature, where each of these approaches has progressed from initial exploration toward more refined designs and implementation. In addition, this report further presents validation and verification within quality assurance, now incorporating more concrete procedures and expanded examples of collaborative work with other work packages.

The report reflects continued progress toward WP4's core objectives:

- Examining the feasibility and implementation of a super-ontology approach.
- Advancing the investigation and prototyping of ML/AI-based translation methods.
- Further assessing and refining model-based translation approaches.

Additionally, during this period WP4 has:

- Further collaborate with the UCs to ensure that translation solutions respond to operational and technical requirements.
- More defined validation and verification processes supporting translation quality assessment.

In summary, WP4's contributions throughout the 30 months of the project demonstrate further advancement, increasing the maturity of translation solutions, and strong alignment with the AfPVN project's strategic objectives.

10 References

- [1] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.
- [2] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "SawSDL: Semantic annotations for WSDL and XML schema," *IEEE Internet Computing*, vol. 11, no. 6, pp. 60–67, 2007.
- [3] J. Köpke and J. Eder, "Semantic annotation of XML-schema for document transformations," in *On the Move to Meaningful Internet Systems: OTM 2010 Workshops*, R. Meersman, T. Dillon, and P. Herrero, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–228.
- [4] F. Moutinho, L. Paiva, J. Köpke, and P. Maló, "Extended semantic annotations for generating translators in the Arrowhead framework," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2760–2769, 2017.
- [5] F. Moutinho, L. Paiva, P. Maló, and L. Gomes, "Semantic annotation of data in schemas to support data translations," in *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2016, pp. 5283–5288.
- [6] G. Amaro, F. Moutinho, R. Campos-Rebelo, J. Köpke, and P. Maló, "JSON schemas with semantic annotations supporting data translation," *Applied Sciences*, vol. 11, no. 24, 2021. [Online]. Available: <https://www.mdpi.com/2076-3417/11/24/11978>
- [7] LangChain Inc., "Langgraph workflows and agents." [Online]. Available: <https://docs.langchain.com/oss/python/langgraph/workflows-agents>
- [8] Model Context Protocol, "Architecture overview — model context protocol," 2025. [Online]. Available: <https://modelcontextprotocol.io/docs/learn/architecture>

- [9] d893a, “data-mapper,” <https://github.com/d893a/data-mapper>, n.d., accessed: 2025-11-17.
- [10] C. Hegedűs, Á. Tóth, M. Bancsics, and P. Varga, “Multi-agentic plan-and-solve engine for managing system of systems via natural language,” in *NOMS 2025 IEEE Network Operations and Management Symposium*, 2025, pp. 1–6.
- [11] T. Tóthfalusi, Z. Csiszár, and P. Varga, “Taco - a generative ai copilot for intent-based telecommunication core network analysis,” in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, 2025, pp. 1–7.
- [12] J. Delsing, “Smart city solution engineering,” *Smart Cities*, vol. 4, no. 2, pp. 643–661, 2021. [Online]. Available: <https://www.mdpi.com/2624-6511/4/2/33>
- [13] K. Mohsenzadegan, V. Tavakkoli, W. V. Kambale, and K. Kyamakya, “Towards seamless data translation based on data models: A hybrid ai framework for smart transportation and manufacturing,” in *2024 19th International Workshop on Semantic and Social Media Adaptation & Personalization (SMAP)*, 2024, pp. 68–73.
- [14] —, “A hybrid ai framework integrating ontology learning, knowledge graphs, and large language models for improved data model translation in smart manufacturing and transportation,” in *2024 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, 2024, pp. 1–6.
- [15] K. Mohsenzadegan, V. Tavakkoli, and K. Kyamakya, “Enhanced hybrid framework for industrial standards translation: Integrating advanced ontology learning, knowledge graphs, and large language models,” 2025, to be submitted.
- [16] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” 2019. [Online]. Available: <https://arxiv.org/abs/1908.10084>

11 Revision History

11.1 Contributing and Reviewing Partners

Contributions	Reviews	Participants	Representing Partner
X	X	Carl Borngrund	LTU
X	X	Jerker Delsing	LTU
X		Oskar Wintercorn	LTU
X		Filipe Moutinho	UNINOVA
X		João Rosas	UNINOVA
X		Afonso Oliveira	UNINOVA
X		Carolina Oliveira	UNINOVA
X		Pedro Maló	UNINOVA
X		Asma Smaoui	CEA
X		Paolo Azzoni	ETH
X		Tamas Tothfalusi	AITIA
X		Peter Olaszi	AITIA
X		Yael Alejandro Chiquito Garcia	SIN
X		Géza Kulcsár	IQL
X		Kabeh Mohsenzadegan	KLU
X		Vahid Tavakkoli	KLU
X		Kyandoghene Kyamakya	KLU
X		Eduard Popovici	UPB
X		Octavian Fratu	UPB
X		Alexandru Vulpe	UPB
X		Șerban Obreja	UPB
X		Salar Sefati	UPB
X		Luis Lino Ferreira	ISEP
X		Clarisse Sousa	ISEP
X		Ricardo Venâncio	ISEP
X		Luis Ribeiro	ISEP
X		Ricardo Severino	ISEP
X		David Hästbacka	TAU
X		Paulo Pinheiro	Piaget
X		George Suciu	BEIA
X		Svetlana Segarceanu	BEIA
X		Theodor Pintilie	BEIA
X		Inge Gavat	BEIA
X		Marius Enache	BEIA
X		Cosmina Stalidi	BEIA
X		Joseba Ibarbia	SAE
X		Miren Illarramendi	SAE

Contributions	Reviews	Participants	Representing Partner
X		Alessandro Ortis	UNICT
X		Dario Allegra	UNICT
X		Filippo Stanco	UNICT
X		Sebastiano Battiato	UNICT
X		Peter Priller	AVL

11.2 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2025-09-25	0.1	Creation of the draft and structure of the document	Carl Borngrund
2	2025-10-01	0.2	Draft for the Introduction and Summary of WP4 Aims and Task sections	Carl Borngrund
2	2025-10-02	0.3	Draft of introductory sections for the sub sections	Carl Borngrund
3	2025-11-12	0.4	Generation of translation solutions	WP4 Partners
4	2025-11-20	0.5	Further description of generation of translation solutions	WP4 Partners
5	2025-11-25	0.6	Finalised the introductory chapters and conclusion	Carl Borngrund
6	2025-11-25	0.7	Further descriptions of translation solutions	WP4 Partners
7	2025-12-11	0.8	Added Task 4.4	WP4 Partners
8	2025-12-11	0.9	Updated Task 4.1	Carl Borngrund
8	2025-12-29	0.10	Added Technological summary	Carl Borngrund
9	2026-01-05	1.0	Final version	Carl Borngrund

11.3 Quality Assurance

No.	Date	Version	Approved by
1	2026-01-07	1.0	Jerker Delsing