Document title
Deliverable D 4.2
Version
1.0
Author
Cristina Paniagua

Date
2024/11/20
Status
Final

Deliverable D 4.2

Second Generation of Translation Technology

Work package leaders:

Cristina Paniagua
cristina.paniagua@ltu.se

Filipe Moutinho
fcm@fct.unl.pt

Abstract
This document constitutes deliverable D4.2 of the Arrowhead fPVN project. This document outlines the second version of translation technologies, built to meet the specific needs detailed in the provided use cases.

AChips.JU project 101111977 - Arrowhead FPVN
THEME [SP1-JTI-ARTEMIS-2012-AIPP4 SP1-JTI-ARTEMIS-2012-AIPP6]
[Project Coordinator: Professor Jerker Delsing — Luleå University of Technology]

Grant agreement no.	101111977
Project acronym	Arrowhead fPVN
Project full title	Arrowhead flexible Production Value Network
Dissemination level	PU
Date of Delivery	November 2024
Deliverable Number	4.2
Deliverable Name	Second generation of translation technology
AL / Task related	Tasks 4.1, 4.2, 4.3 and 4.4
Author/s	Cristina Paniagua and Filipe Moutinho
Contributors	Arrowhead fPVN WP4 partners
Reviewer	Jerker Delsing
Keywords	Translation, Interoperability, Ontology, AI, Modeling
Abstract	This document constitutes deliverable D4.2 of the Arrowhead fPVN project. This document outlines the second version of translation technology, built to meet the specific needs detailed in the provided use cases.

Contents

Abbreviations

Abbreviation	
AFPVN	Arrowhead flexible Production Value Network
AI	Artificial Intelligence
CPS	Cyber-Physical Systems
DITAG	Data Interoperability Translators Automatic Generator
DSL	Domain Specific Language
JSON	JavaScript Object Notation
LLM	Large Language Models
ML	Machine Learning
NLP	Natural Language Processing
NMT	Neural Machine Translation
OWL	Web Ontology Language
QoS	Quality of Service
SAWSDL	Semantic Annotations for WSDL and XML Schema
SysML	System Modeling Language
UC	Use Case
UML	Unified Modeling Language
WP	Work Package
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema

1 Introduction

Work Package (WP) 4 focuses on researching, designing, and deploying translation solutions among designated data models. The collaboration among partners showcases their work across three distinct translation methodologies: ontology-driven, AI-powered, and modeling-based approaches. The work done during the first 18 months of the project is described in this comprehensive report, D4.2, which outlines the second generation of translation solutions and provides insights into their current status. The document is organized as follows.

Section 2 describes the aims of WP4 and the individual purposes of each task. Section 3 summarizes the objectives of WP4 and relates the efforts described in this deliverable to the WP and project objectives. Section 4 is the core section of this document; it is divided into three subsections, each describing the proposed translation solutions classified by approach. Section 5 highlights the collaboration between WP4 and the UCs. Section 6 reflects the work initiated by Task 4.4 to validate the solutions and perform the quality assessment. Section 7 describes the collaboration with other WPs to achieve the common project objectives. A list of appendices is included at the end of the document.

2 Summary of WP4 Aims and Tasks

WP4 aims to design and develop translation service solutions, facilitating seamless data model translation across prominent data models and standards. It consists of four tasks, namely:

- 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.

Their main goals are the following:

Task 4.1 Task 4.1 will investigate the use of ontologies as a bridge between significant data modeling languages to resolve inherent property mismatches. The focus will be on ontology-based translation tools to facilitate microservices translations relevant to workflow management. Key actions include identifying relevant ontologies, applying expertise in workflow languages for seamless translation, and exploring machine learning techniques to address mapping challenges.

Task 4.2 Task 4.2 will assess the effectiveness of Machine Learning/Artificial Intelligence (ML/AI) approaches for translating between major data modeling languages. By testing translation accuracy with real-world data from diverse use cases, the task aims to validate these methods across different language structures. It will also develop translation microservices based on existing WP2 architecture, employing techniques such as dictionary learning, translation algorithms, and Natural Language Processing (NLP) to analyze feasibility, precision, and applicability for seamless data model translation.

Task 4.3 Task 4.3 will explore model-based approaches like Unified Modeling Language (UML) and System Modeling Language (SysML) to evaluate their effectiveness in translating between major data modeling languages. The task will test these translations using data from various use cases and develop microservices based on WP2 architecture. Activities include examining UML metamodels, identifying similarities, and applying formal semantic reasoning or learning methods. Additionally, it will evaluate industrial production requirements and usability through specific use cases, while investigating guidelines for model-based translation in safety-critical domains.

Task 4.4 Task 4.4 will provide essential datasets for the development, training, and validation of translation quality throughout the project lifecycle. It will collaborate closely with use cases and other work packages to ensure comprehensive dataset procurement, supporting rigorous testing and evaluation of translation outcomes to ensure alignment with project objectives and quality benchmarks.

3 WP4 Objectives

The primary aim of WP4 is to advance translation service technology, facilitating seamless data model translation across prominent data modeling languages such as ISO 10303, ISO 15926, IEC 81346, and S5000F. This work has been divided into the following set of 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 data set 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

This table shows which project objectives are addressed by this part of the deliverable and how these 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 second generation of translators and details the efforts made during the first 18 months of the project to achieve the targeted percentage. The work involves collaborating with various use cases to integrate translation seamlessly into production value networks.

3.2 WP Objectives

This table shows which WP objectives are addressed by this part of the deliverable and how these 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 ontologies for translation, as detailed in Section 4.1. This work includes three prototypes (DITAG and DITAG Manager, HASCO/Rep, Polygllot) and a new proposal, the business process ontology-based adaptor. They also have been used in UC (WP6 and WP7), see Section 5.
Investigating the capabilities and feasibility of an ML/AI-based approach.	This document outlines the work conducted on the utilization of AI and ML techniques for translation, as detailed in Section 4.2. It includes an development of translation solutions using RAG and langChain frameworks, as well as the collaboration and integration of solutions with the UC (WP6 and WP7), see Section 5.
Investigating the capabilities and feasibility of a model-based approach.	This document provides a comprehensive overview of the work conducted on the utilization of models for translation, detailed in Section 4.3. It encompasses translation solutions based on Papyrus and translation between SysMLv1 and SysMLv2. A new collaboration with Task 4.2 also has been initiated to obtain further benefits from the usage of AI together with Models, including the generation of UML diagrams and Object detection in engineering diagrams. Finally, Rely-SysML translation has been tested in WP7, Section 5.
Provision of data set for translation development and early assessment of translation quality.	This document includes in Section 6 a proposed methodology to validate and verify the translation solutions. An individual process has been defined for each approach.
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. Section 7 describes the efforts and results from this collaboration.

4 Second Generation Translation Technologies

Effective communication between systems relies on compatible interfaces and a shared data understanding, including factors like protocols, encoding, encryption, compression, message structure, payload key values, and semantics. Once data is received, accurate interpretation is critical but mismatches in interfaces or data formats can obstruct communication. Translation techniques, serving either as interim or permanent solutions, can bridge these gaps—particularly during system failures when an alternative system lacks compatibility, or when integrating systems from various vendors or versions to save engineering time. Updates to legacy systems may also disrupt compatibility, making translation necessary. However, autonomous translation presents challenges, especially in achieving deep semantic understanding across diverse data models and managing multimodal data. Ensuring interoperability often requires standardized exchange formats and well-defined APIs to maintain seamless communication across systems.

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

- **Ontology-based translation:** This approach leverages ontologies, which are formal representations of knowledge, to map concepts and relationships between different data models. By utilizing ontologies, semantic similarities and differences can be identified and translated, facilitating interoperability between heterogeneous systems.
- **AI-based translation:** AI-based translation utilizes machine learning algorithms and techniques to automatically learn and translate data between different formats and structures. This approach often involves training models on large datasets to develop accurate translation mechanisms that can handle complex data semantics and structures.
- **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.

The second generation of translation technologies is described in the following subsections. The work has been classified into three categories to facilitate its review:

- Proposal: This category encompasses the initial conceptualization and proposition of translation solutions, outlining the theoretical framework and methodology.
- Prototype: In this category, tangible prototypes of the proposed translation solutions are developed and tested, providing practical insights into their feasibility and functionality.
- Support activity: The support activities encompass pre-studies, partial investigations, and analyses aimed at enhancing the domain's state-of-the-art knowledge. These activities offer valuable insights to partners implementing prototypes, supporting and complementing the design and development of the translators.

4.1 Ontology–Based Translation

The following section summarizes the efforts and progress accomplished during the second year of the project concerning ontology-based translation. These efforts encompassed the analysis of requirements, as well as the design and development of translation solutions.

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

During this period, the development of the tool Data Interoperability Translators Automatic Generator (DITAG) continued. DITAG automatically generates translators (Figure 1) based on ontologies and Schemas (XML Schemas or JSON schemas) with semantic annotations and also performs message/document translation (Figure 2). 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.

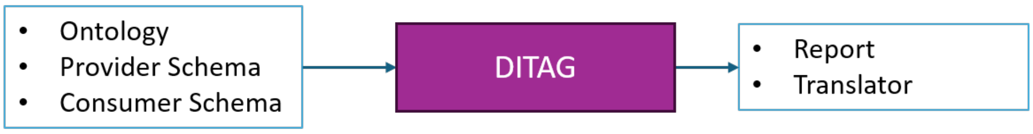


Figure 1: Translator creation (stage 1).

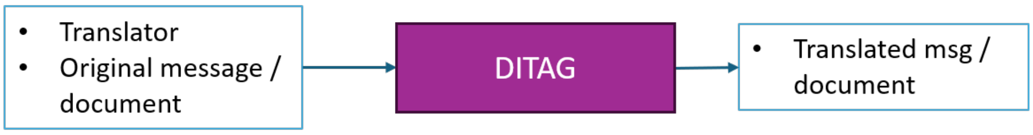


Figure 2: Translator Application (stage 2).

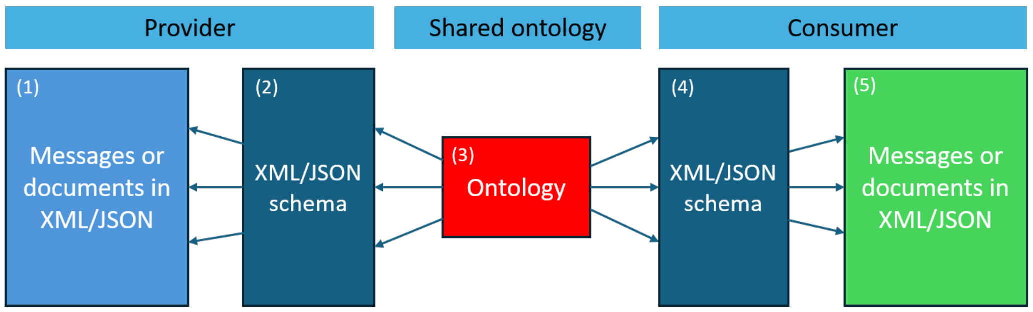


Figure 3: Elements needed/used (blocks 1, 2, 3, and 4) and created (block 5) by DITAG.

Figure 3 illustrates which elements are needed/used (blocks 1, 2, 3, and 4) and also created (block 5) by DITAG.

The elements presented in the Figure 3 have the following role:

- Block (1): represents the set of messages or documents that the provider sends/generates. The messages/documents' content format can be XML or JSON.
- Block (2): represents the Schema that describes the structure of the messages or documents in block 1. This Schema must have semantic annotations for a shared/reference ontology.
- Block (3): represents the ontology that is used in both provider and consumer Schemas.
- Block (4): represents the Schema of the messages/documents the consumer can read/understand. This Schema must also have semantic annotations for the shared/reference ontology (block 3).
- Block (5): represents the set of messages or documents (XML or JSON) that the consumer can read/understand (in compliance with block 4). These messages/documents are generated by DITAG using the translator it automatically creates.

As described in deliverable 4.1, 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 it, DITAG needs *Providers* and *Consumers* metadata, namely XML Schemas (XSDs) or JSON Schemas with semantic annotations. The semantic annotations are added to the XSDs using the SAWSDL (Semantic Annotations for WSDL and XML Schema) [1] extension attribute “modelReference” with annotation paths [2] and group identifiers [3]. These annotations provide additional information about the meaning of the elements and attributes defined in XSD documents, using Ontologies specified in Web Ontology Language (OWL) [4]. Moreover, the Annotating Schemas to Support Semantic Translations (A3ST) extension [5] is also employed to supplement SAWSDL annotations. This enables the provision of additional information about an element without disturbing the original structure of the XSD. DITAG uses JSON Schemas with semantic annotations to support JSON data, as proposed in [6].

The updated version of DITAG, developed during the second year, includes several improvements, such as bug fixes, enhanced support for data type conversions, and support for an improved version of the A3ST extension. The improved A3ST additionally includes: (1) the element ”a3st:model” with an attribute ”a3st:ontology”, which is used by DITAG to check if both provider and consumer schema annotations are from the same ontology; and (2) an improved method for specifying mappings between data values and ontology individuals (”a3st:map-data-ind”). With this improved A3ST, DITAG now supports data values with the same syntax but different semantics. Figure 4 illustrates the use of both SAWSDL and A3ST to add more context to XSD. In the presented XSD, the element ”a3st:map-data-ind” was used to specify the values (”F” and ”C”) used by the system to specify temperature units (”Fahrenheit” and ”Celsius”). A3ST also supports the specification of complement property values (”a3st:comp-property-value”), which were used in the presented XSD to specify the location of the temperature sensors.

```xml
<?xml version="1.0"?>
<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">
    <xs:element name="datavalues">
        <xs:complexType><xs:sequence>
            <xs:element type="xs:float" name="sensor1temp" sawsdl:modelReference ="/TemperatureSensor{1}/hasDecValue" />
            <xs:element type="xs:float" name="sensor2temp" sawsdl:modelReference ="/TemperatureSensor{2}/hasDecValue" />
            <xs:element type="xs:string" name="units" sawsdl:modelReference="/TemperatureSensor{1;2}/hasTempUnits/TemperatureUnits{1}"
       a3st:mdi-ref="1"/>
        </xs:sequence></xs:complexType>
    </xs:element>
    <xs:annotation>
        <xs:appinfo>
            <a3st:model a3st:ontology="http://gres.uninova.pt/tag/validation/json/tempsensorv2b.owl" />
            <a3st:map-data-ind a3st:mdi-id="1">
                <a3st:map-value-ind a3st:individual="Fahrenheit" a3st:value="F" />
                <a3st:map-value-ind a3st:individual="Celsius" a3st:value="C" />
            </a3st:map-data-ind>
            <a3st:comp-property-value a3st:property="/TemperatureSensor{1}/hasLocation/Latitude{1}[hasLocUnits/LocationUnits{1}/
      hasIndividual/DD]/hasDecValue" a3st:value="38.659963"/>
            <a3st:comp-property-value a3st:property="/TemperatureSensor{1}/hasLocation/Longitude{1}[hasLocUnits/LocationUnits{1}/
      hasIndividual/DD]/hasDecValue" a3st:value="-9.203966"/>
            <a3st:comp-property-value a3st:property="/TemperatureSensor{2}/hasLocation/Latitude{2}[hasLocUnits/LocationUnits{1}/
      hasIndividual/DD]/hasDecValue" a3st:value="38.659571"/>
            <a3st:comp-property-value a3st:property="/TemperatureSensor{2}/hasLocation/Longitude{2}[hasLocUnits/LocationUnits{1}/
      hasIndividual/DD]/hasDecValue" a3st:value="-9.203929"/>
        </xs:appinfo>
    </xs:annotation>
</xs:schema>
```

Figure 4: XML Schema Example. Notice the added context from SAWSDL for the elements `units`, `sensor1temp`, and `sensor2temp`. Additionally, the a3st extension complements SAWSDL by allowing further annotations at the end of the schema.

Additionally, the updated version of DITAG 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 analyzed 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 5). 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.

4.1.2 DITAG-Manager (Prototype)

In deliverable 4.1, an approach to integrate the DITAG into the Arrowhead Framework was presented, as illustrated in Figure 6. This section describes the preliminary work for this integration. Integrating DITAG into the Arrowhead framework may allow service consumers to translate the messages/documents resulting from the provision of services into messages/documents whose structure and semantics consumers can understand.

This integration requires the creation of a tool with the necessary functionalities for the automatic and real-time generation of translators as they become needed. This tool is named DITAG-Manager. Its operation is divided into two stages (Figures 1 and 2). In the first stage, a specific translator is created. After its creation, the translator needs to be stored and ready for future use. This stage only needs to be performed once for each

```
{
  "combinations": [ ... ],
  "response": {
    "message": "Operation Completed Successfully",
    "code": 0
  },
  "match": {
    "obligatory-matches": 2,
    "optional-matches": 1,
    "pairs": [ ... ]
  }
}
```

Figure 5: Excerpt of Report.json generated by the DITAG Tool.

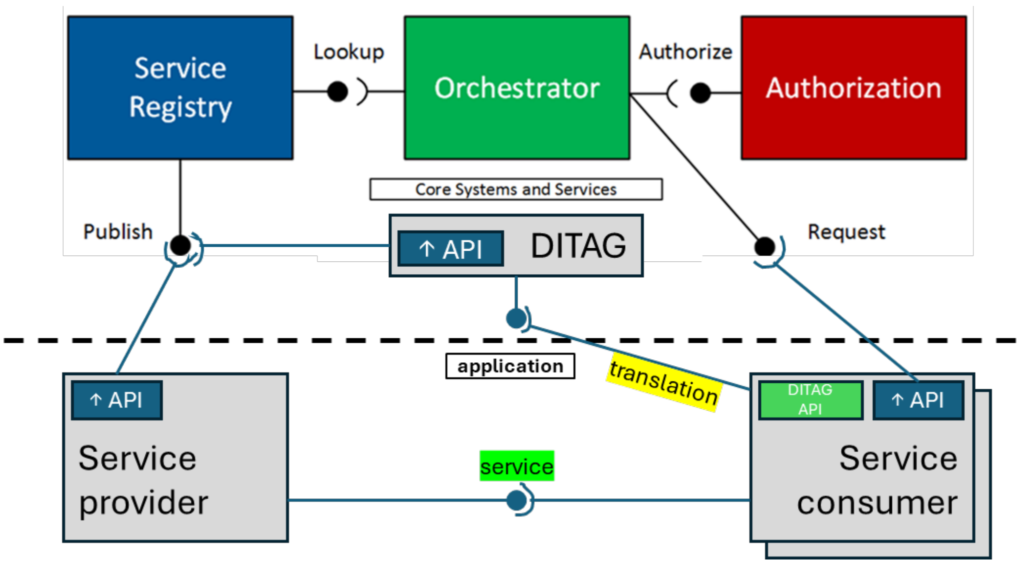


Figure 6: Integration of the DITAG tool in the Arrowhead cloud (adapted from https://arrowhead.eu/technology/architecture/).

translator. In the second stage, the translator generated in the previous stage is used to translate documents or messages. The second stage is to be used whenever a translation is needed. This functionality will then be supported and managed by the DITAG-Manager tool. The architecture of this tool is illustrated in Figure 7. As shown, the tool is composed of various components, namely the Rest controller, the Swagger API, the ontology manager, and the translation engine. At the base of the architecture are the Spring Boot framework and DITAG itself.

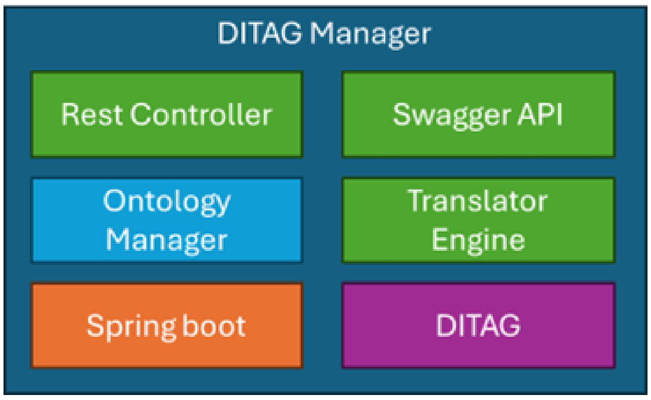


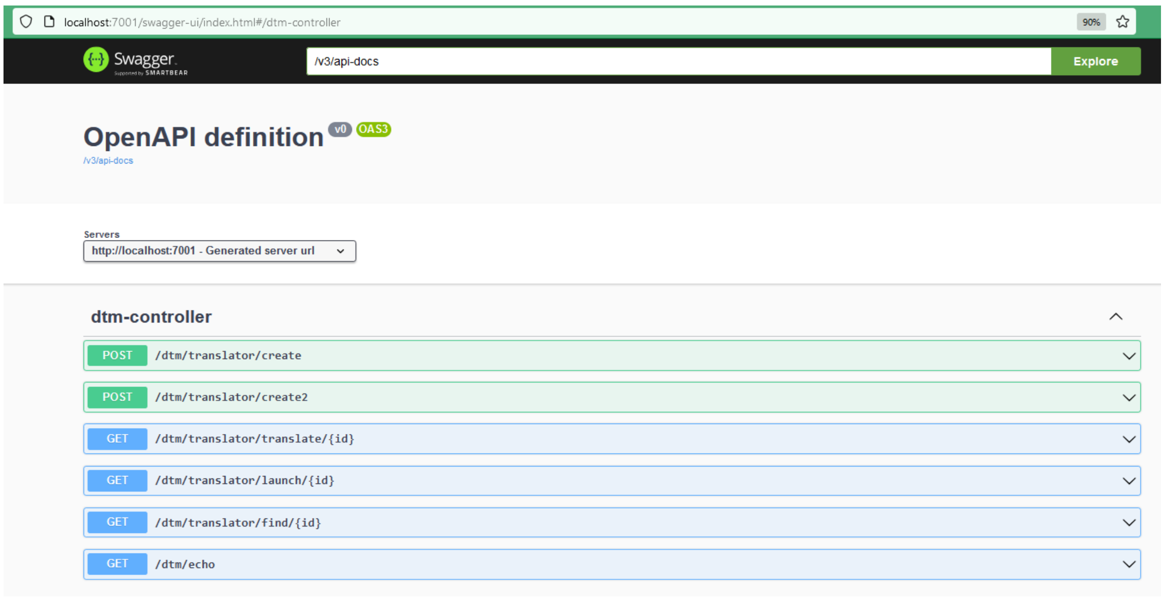
Figure 7: DITAG Manager.

The internal structure of the most relevant blocks of the DITAG-Manager is described in Table 3, namely the Rest controller and the DITAG. The Translator Engine provides the runtime for executing DITAG and its translators. This enables the effective generation and translation of messages/documents in a production environment without the need for human intervention.

Table 3: DITAG Manager Services.

DITAG Manager		Description
RestController	createTranslate()	Receives the schemas and the ontology. Invokes DITAG to create a translator. The resulting translator is identified by an unique ID. A codebase for the translator is created and accessed through the ID.
	translateMessage()	Receives a message and ID of the translator.
	findTranslator()	Query for the existence of a translator using metadata information, like schemas and ontologies.
DITAG	createTranslator()	Creates the translator.
	TranslateMessage()	Translates a message or document.
	...	
Created translators	09247b79-24db-4b43-84da-b6dfbd936590	Translator between provider 1 and consumer 1
	80b84bf3-c35e-49a1-9ea7-54e478272179	Translator between provider *i* and consumer *j*
	...	...

Figure 8 presents the services implemented for the DITAG-Manager through its respective Swagger API interface. It is important to mention that the manager is in an intermediate stage of development, with some services partially developed.


localhost:7001/swagger-ui/index.html#/dtm-controller

Swagger /v3/api-docs Explore

OpenAPI definition v0 OAS3
/v3/api-docs

Servers
http://localhost:7001 - Generated server url

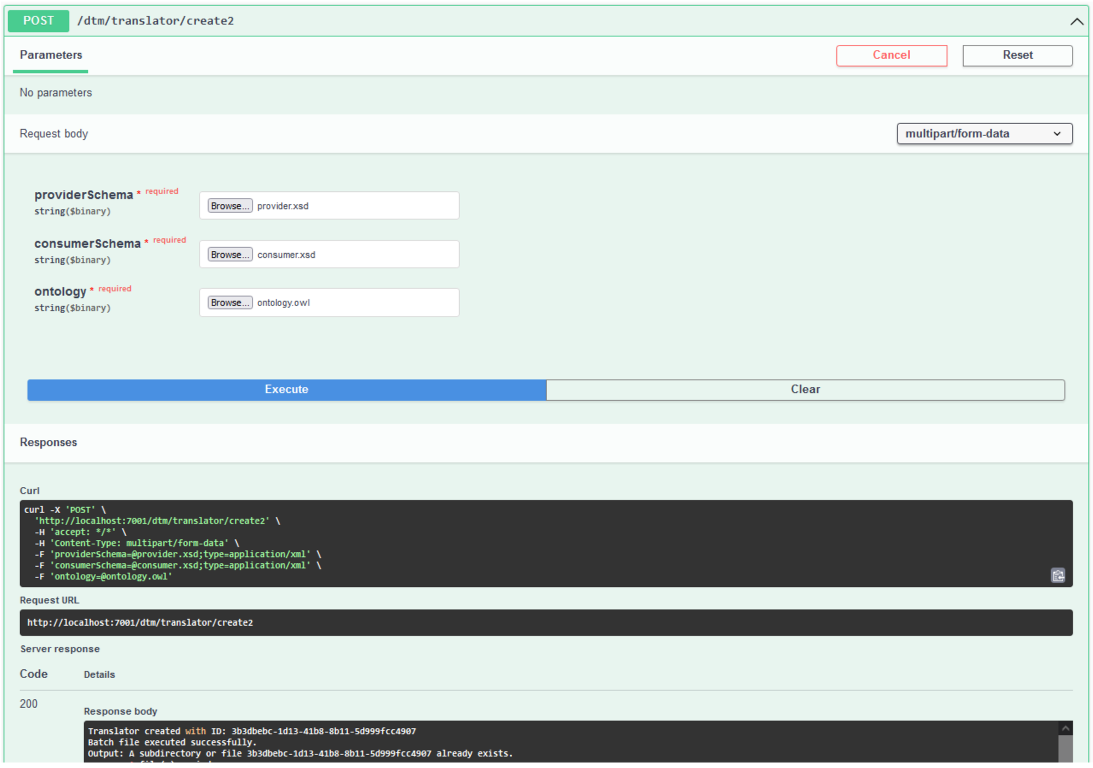
dtm-controller

```
POST /dtm/translator/create
POST /dtm/translator/create2
GET  /dtm/translator/translate/{id}
GET  /dtm/translator/launch/{id}
GET  /dtm/translator/find/{id}
GET  /dtm/echo
```

Figure 8: Swagger API interface.

Figure 9 illustrates a call to the DITAG-Manager through the Swagger API to create a translator. The result of this call was the generation of an ID (3b3dbebc-1d13-41b8-8b11-5d999fcc4907) that unequivocally represents the generated translator. This translator can subsequently be used for translating messages between the provider and the consumer involved in the service.

Figure 10 illustrates the invocation of a service to proceed with the translation of a message using the translator with id=" 3b3dbebc-1d13-41b8-8b11-5d999fcc4907", which is input in the field "id" shown in the figure. The file with the message or document to be translated is uploaded in the field named "providerMessage".


POST /dtm/translator/create2

Parameters — Cancel — Reset

No parameters

Request body: multipart/form-data

providerSchema * required — string($binary) — Browse... provider.xsd

consumerSchema * required — string($binary) — Browse... consumer.xsd

ontology * required — string($binary) — Browse... ontology.owl

Execute — Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:7001/dtm/translator/create2' \
  -H 'accept: */*' \
  -H 'Content-Type: multipart/form-data' \
  -F 'providerSchema=@provider.xsd;type=application/xml' \
  -F 'consumerSchema=@consumer.xsd;type=application/xml' \
  -F 'ontology=@ontology.owl'
```

Request URL

```
http://localhost:7001/dtm/translator/create2
```

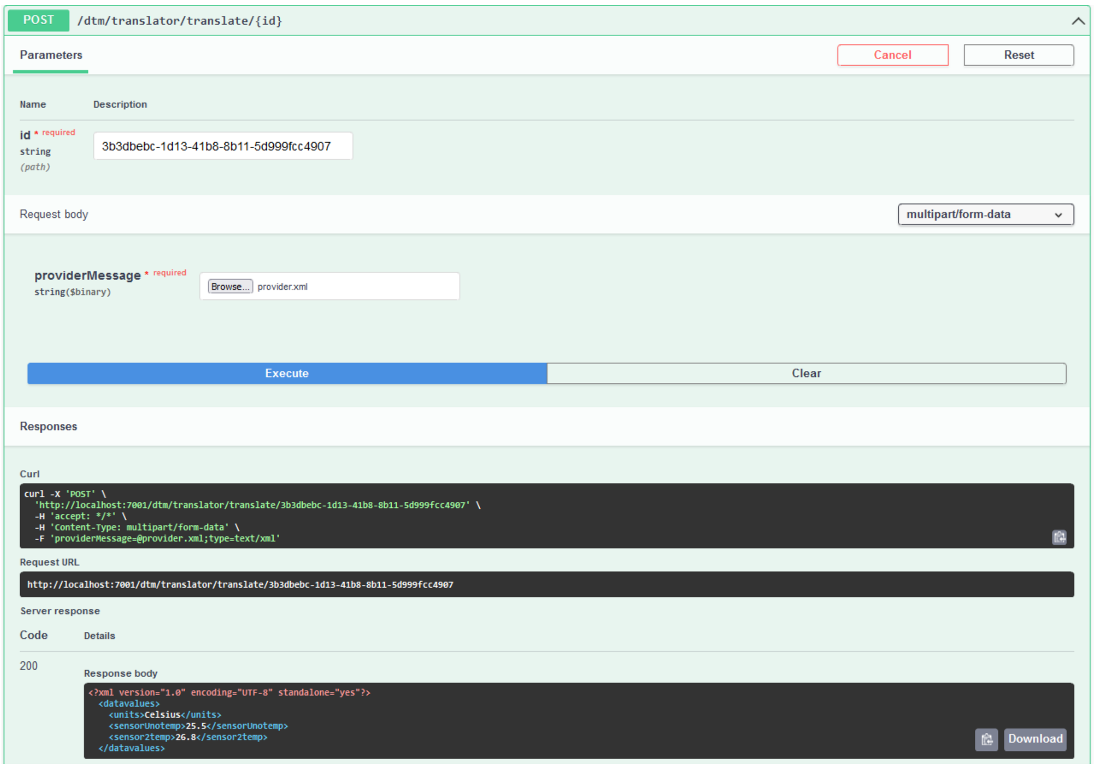
Server response

Code — Details

200 — Response body

```
Translator created with ID: 3b3dbebc-1d13-41b8-8b11-5d999fcc4907
Batch file executed successfully.
Output: A subdirectory or file 3b3dbebc-1d13-41b8-8b11-5d999fcc4907 already exists.
```

Figure 9: Swagger API to create a translator.


POST /dtm/translator/translate/{id}

Parameters — Cancel — Reset

Name — Description

id * required — string (path) — 3b3dbebc-1d13-41b8-8b11-5d999fcc4907

Request body: multipart/form-data

providerMessage * required — string($binary) — Browse... provider.xml

Execute — Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:7001/dtm/translator/translate/3b3dbebc-1d13-41b8-8b11-5d999fcc4907' \
  -H 'accept: */*' \
  -H 'Content-Type: multipart/form-data' \
  -F 'providerMessage=@provider.xml;type=text/xml'
```

Request URL

```
http://localhost:7001/dtm/translator/translate/3b3dbebc-1d13-41b8-8b11-5d999fcc4907
```

Server response

Code — Details

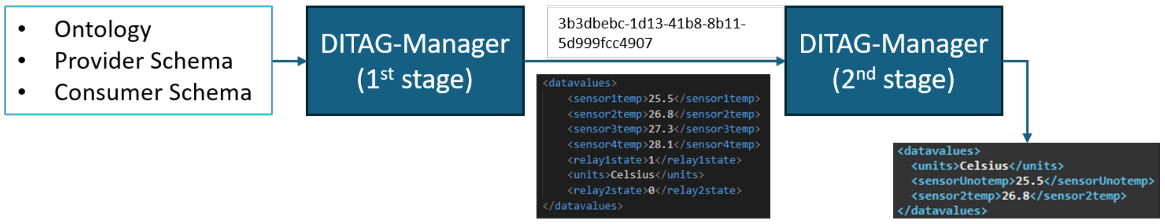
200 — Response body

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <datavalues>
    <units>Celsius</units>
    <sensorUnotemp>25.5</sensorUnotemp>
    <sensor2temp>26.8</sensor2temp>
  </datavalues>
```

Download

Figure 10: Swagger API to execute a translator.

Both the stages for the translation process performed by the DITAG-Manager are illustrated in Figure 11. As shown, in the first state, the DITAG-Manager is provided with the Schemas and the ontology and generates a translator identified by a unique ”ID”. In the second stage, a message from the provider and the ID of the translator is provided, and as a result, the translator yields the translated message. As WP4 can see, the translated document has a structure that the consumer can now read.


• Ontology
• Provider Schema
• Consumer Schema
DITAG-Manager (1st stage)
3b3dbebc-1d13-41b8-8b11-5d999fcc4907
DITAG-Manager (2nd stage)

```
<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>
```

```
<datavalues>
  <units>Celsius</units>
  <sensorUnotemp>25.5</sensorUnotemp>
  <sensor2temp>26.8</sensor2temp>
</datavalues>
```

Figure 11: DITAG-Manager stages illustration.

Finally, it is important to note that the following phases, proposed in deliverable 4.1, were tested successfully:

- Publish phase:
  1. The service provider publishes its services in the Arrowhead registry. Beyond the Arrowhead-specific elements, the provider publishes the service schemas as metadata.
- Request:
  2. The consumer makes a request to the Orchestrator for a service of a given name. It receives a list of providers that can provide the requested service.
- Translation:
  3. For the required service, the consumer requests DITAG-Manager to select suitable providers among the ones received from orchestration. The request carries both the consumer schema and provider schemas.
  4. DITAG-Manager replies with a compatible provider and corresponding translator.
- Service provision:
  5. The service request is made to the chosen provider, receiving a corresponding message/response.
  6. The response is translated by the received translator.
  7. The consumer uses the translated message.

4.1.3 HASCO/Repo: Intelligent Consumers and Producers (Prototype)

Most scientific data repositories have limited capacity to integrate data within studies and even less to support harmonization across multiple studies. Data collected to support scientific research is often gathered in diverse contexts (e.g., using different instruments), making it difficult to integrate datasets from different studies. In this report, WP4 describes how a semantic repository can capture, preserve, and leverage the data generated and consumed in the Arrowhead fPVN Project. WP4 also explains how this infrastructure supports DITAG harmonization operations.

On the Vagueness of “Data Kinds”

Arrowhead framework supports the interaction between application services, which is facilitated by a set of core services included in the framework. These core services enable the exchange of information between application services. The framework helps a service consumer (or simply ”consumer”) find a service provider (or ”provider”) that offers the service the consumer requires. By offering a service, Arrowhead implies that it can inform the consumer about available providers capable of supplying the ”kind of data” that the consumer needs. The term “variable” would be a more formal way to describe the kinds of data provided by a service. However, the specification of variables within Arrowhead is somewhat vague, assuming that variables underlying a producer’s data can be annotated with metadata using terms from a reference ontology.

There are several issues with the Arrowhead baseline approach for describing variables, as outlined above. Below, WP4 identifies three main issues regarding the description of variables in Arrowhead services:

- **Lack of Semantic Annotation Automation.** The semantic annotation of services (i.e., generating the annotation metadata) is currently done manually, and most services in real-world Arrowhead scenarios, whether consumers or producers are unable to provide such annotation metadata.
- **Reliance on a Single Reference Ontology.** Even if semantic annotation were available for every service, it is unlikely that these annotations could all be provided by a single reference ontology. For example, in the existing use case scenarios within the Arrowhead fPVN Project, the domains vary widely, spanning industries such as automotive, aviation, and oil. Each of these domains would likely have its own vocabulary, defined in one or more ontologies. Consequently, there would likely be minimal overlap between the content of these ontologies.
- **Reliance on a Variable’s Attributes to Fully Describe Variables.** If the vocabulary needed to describe the variables for every service were provided by a single reference ontology, many essential properties of variables—such as units, code books, temporal and spatial restrictions, and role restrictions—might still be overlooked.

Our proposed solution to the issues outlined above regarding “data kinds” is to leverage a semantic infrastructure that provides metadata annotations based on a vocabulary derived from a collection of authoritative ontologies. Furthermore, WP4 aims for the vocabulary to be precise (based on W3C’s semantic languages) and unique (using W3C’s Uniform Resource Identifier (URI)).

On the Use of Semantic Infrastructure

W3C’s semantic languages enable machines to understand and process data on the web by providing structured, machine-readable formats. These languages are already leveraged by various industry standards for data integration, sharing, and intelligent applications.

- **RDF (Resource Description Framework)**: A framework for representing data as subject-predicate-object triples, enabling the description of resources and their relationships. It’s used in standards like SPARQL [7] for querying and Linked Data for data interlinking.
- **RDFS (RDF Schema)**: Extends RDF by defining classes and properties, allowing for simple hierarchies and relationships between resources. RDFS is used in many ontologies and metadata standards, like Dublin Core.
- **OWL (Web Ontology Language)**: Used to define complex ontologies, including logical reasoning and constraints for rich, detailed knowledge representation. OWL is integral to standards like Health Level 7 (HL7) FHIR for healthcare data and Schema.org for web semantics.

These languages are foundational to the Semantic Web and are widely used in fields like healthcare, finance, e-commerce, and AI for improving data interoperability, search, and automated reasoning.

Knowledge About Scientific Studies

For encoding knowledge about scientific studies, HASCO/Repo’s semantic data ingestion leverages science ontologies including the Human-Aware Science Ontology (HAScO) [8] and the Semanticscience Integrated Ontology (SIO) [9]. HAScO is used for encoding knowledge about studies, study types, and data elicitation from human subjects. SIO is used for encoding knowledge about entities and their properties.

Scientific ontologies, like SIO (and others), tend to be domain-agnostic and used with the exclusive purpose of describing the data. They don’t intend to support the process of the data throughout the scientific life cycle but are dependent on some data organizational needs and circumstances specified during the study design phase. HAScO supports scientific data organization by defining the notion of collections (SampleCollection and TimeCollection) and groups (SubjectGroup) of objects of interest and their relations.

Other ontologies generally used are W3C PROV Ontology [10] for encoding provenance knowledge and the Virtual Solar-Terrestrial Observatory (VSTO) [11] for encoding knowledge about instruments and platforms.

Knowledge Graph Infrastructure

HASCO/Repo (Human-Aware Science Ontology's Repository) is a knowledge graph infrastructure based on the Human-Aware Science Ontology (HAScO) [8] that has been conceptualized to support the storage of large volumes of scientific data and the comprehensive description of entities that compose a scientific study. The infrastructure provides core features in support of scientific data repositories including an RDF triple-store (Apache Jena Fuseki [12]) for storage and a collection of metadata templates to support the capture of scientific metadata.

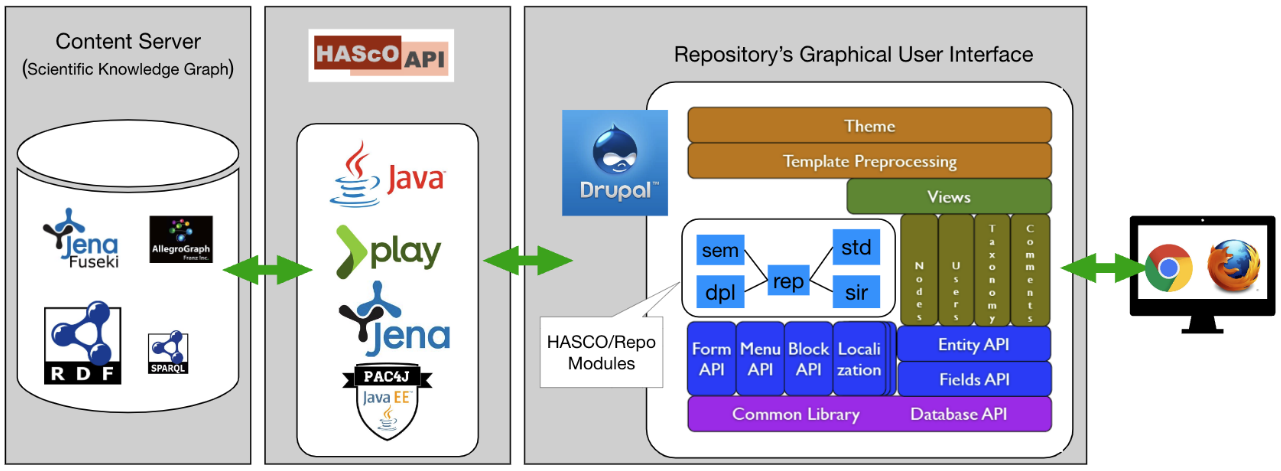


Figure 12: HASCO/Repo Architecture.

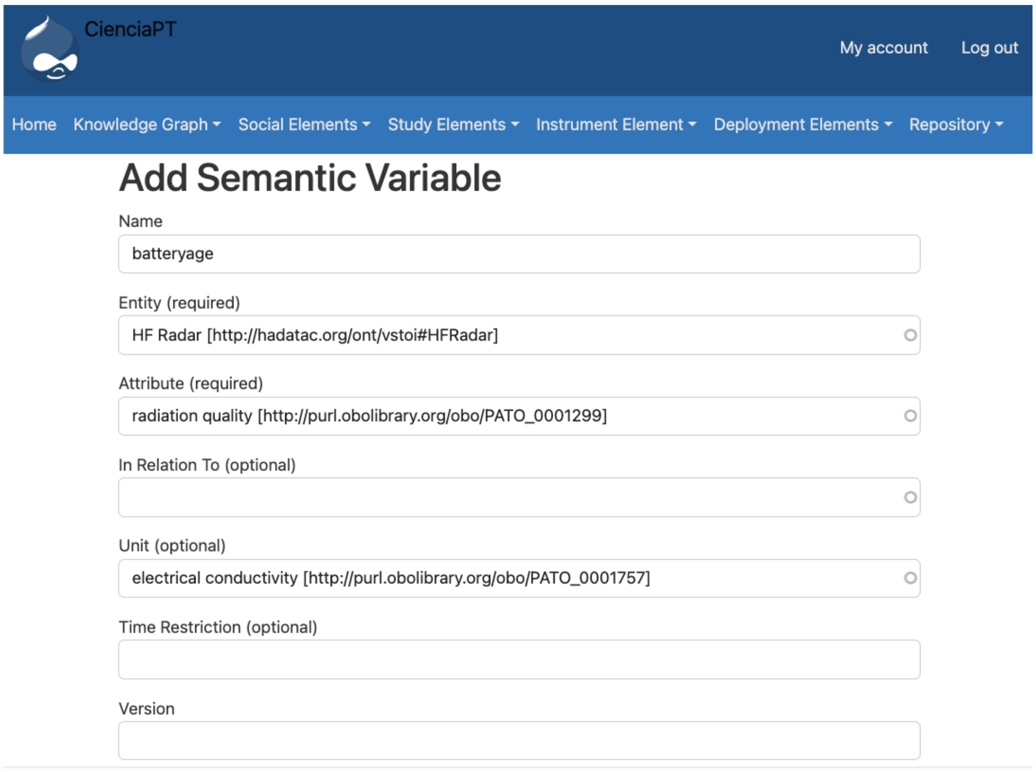
Externally, the HASCO/Repo is presented as a web application that provides FAIR access to the repository's content. HASCO/Repo relies on HASCO/API to manage its content, and HASCO/API is available to other infrastructure's subsystems as a Java API that encodes HAScO's entities as Plain Old Java Object (POJO) classes. The HAScO API classes are used to build and maintain HASCO/Repo's SKG, persisting the graph's elements into the repositories. They have the important role of encapsulating the use of SPARQL queries against the content repository.

HASCO/Repo's core component is composed of the SKG as well as their associated ingestion processes, which are responsible for extracting, annotating, and storing study content from data and metadata files into the repository. Content is ingested into HASCO/Repo either through parsing uploaded files, or directly using the HAScO API.

Variables and Semantic Variables

A *Variable*, from the point of view of a tabular data file, is a column in the file. Each variable value, i.e., a value in a column of the table corresponding to our variable of concern, is the measured, elicited, or simulated value of an entity's attribute. For example, for one of our battery datafile `UseCase_1_6` there is a variable named `BATTERYAGE` that corresponds to the attribute "age" of an entity of type "electricity storage", and it is stated in "years". WP4 considers all information about the variable (such as attribute, entity, and unit, in this example) as properties of the variable. For example, "age" is the property *Attribute* of the variable, "electricity storage" is the property *Entity* associated with the variable, and "years" is the property *Unit* of the variable. It is important to mention that our dataset contains data about 200 batteries, which is the battery population of our `UseCase_1_6` study.

A *Variable Specification* is the description of the properties of a variable. The population "Battery Population



Add Semantic Variable

Name: batteryage

Entity (required): HF Radar [http://hadatac.org/ont/vstoi#HFRadar]

Attribute (required): radiation quality [http://purl.obolibrary.org/obo/PATO_0001299]

In Relation To (optional):

Unit (optional): electrical conductivity [http://purl.obolibrary.org/obo/PATO_0001757]

Time Restriction (optional):

Version:

Figure 13: Editting a semantic variable in HASCO/Repo.

in `UseCase_1_6`", the attribute "age", the entity "electricity storage", and the unit "years" are all part of the specification of the `BATTERYAGE` variable for the `UseCase_1_6` study. WP4 observes that some properties are present in some variable specifications while others may not. For instance, not all variables require a property "Unit", especially when these variables are categorical like "Is Functional". A comprehensive discussion about variable specification formalization is beyond the scope of this deliverable. However, WP4 would like to particularly stress that variable specifications may be missing essential content if their properties *Entity*, *Attribute*, and *Population* are not provided, or are provided as empty definitions.

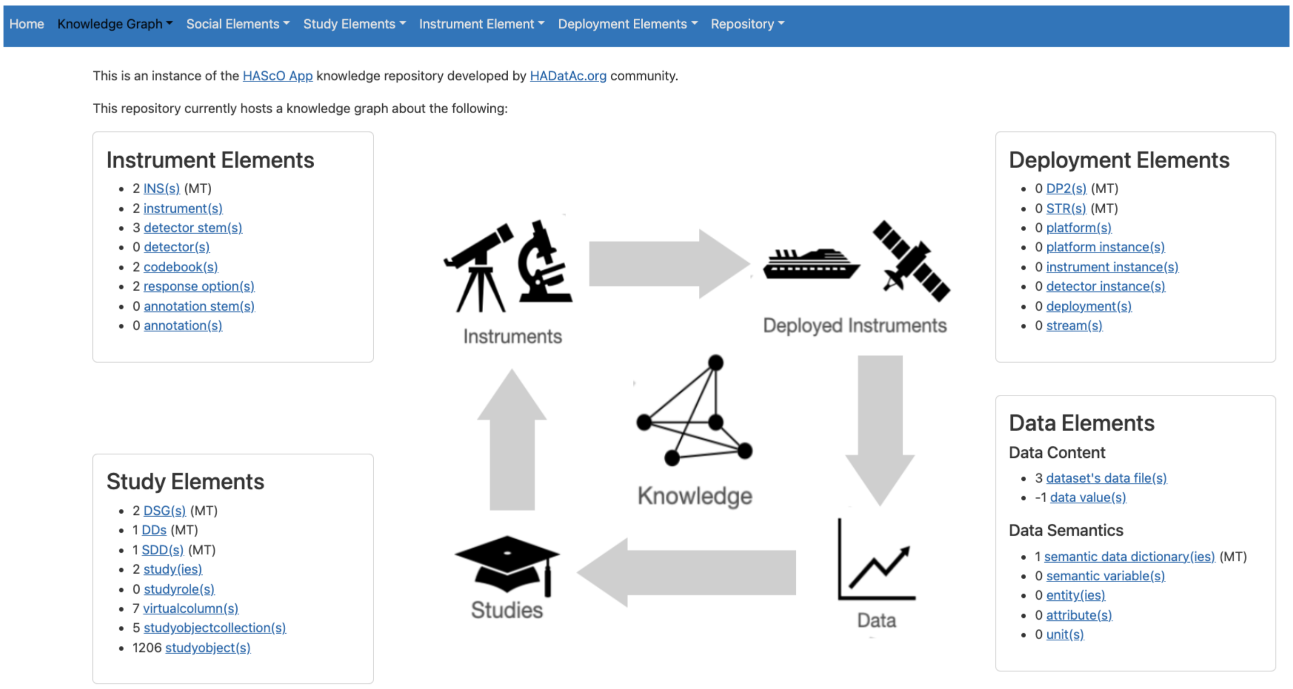
From our definition of Variable Specification, WP4 defines a *Semantic Variable* as a variable specification that does not include a population property. From the variable definition above, each variable is bound to a given population. When the only distinction between the set of properties of any two variables is their populations, WP4 would say that the two variables have the same semantic variable, i.e., the two variables share a common semantic variable. In this case, WP4 can say that `BATTERYAGE` for the `UseCase_1_6` study and `BTYEARS` for the `UseCase_2_3` study are two variables with the same semantic variable ("age of the electricity storage in years") reference. In fact, the only distinction between these two variables is their populations.

Capture of Variable's Semantic Context

Variables need to be described at the semantic level. Similarly, contexts, where variable data are acquired, are also required to be described at the semantic level. HASCO/Repo captures scientific knowledge including knowledge about contexts where variable data are acquired through the use of metadata templates (MTs).

MTs provide a framework for domain experts to identify and define the semantics of the study elements, including study metadata, study object collections (e.g., cohorts), roles that object collections play in studies (e.g., subjects, samples), object properties, and relationships among object collections. MTs encode knowledge at both the study and dataset levels.

HASCO/Repo's MTs are encoded as tabular data (CSV and XLSX) and each type of MT has a specific set of required tables and content (columns and rows). When interpreted by HADatAc, each row in a MT table will be translated to RDF resources in the scientific knowledge graph, according to the purpose of the MT and the



This is an instance of the HAScO App knowledge repository developed by HADatAc.org community.

This repository currently hosts a knowledge graph about the following:

Instrument Elements

- 2 INS(s) (MT)
- 2 instrument(s)
- 3 detector stem(s)
- 0 detector(s)
- 2 codebook(s)
- 2 response option(s)
- 0 annotation stem(s)
- 0 annotation(s)

Deployment Elements

- 0 DP2(s) (MT)
- 0 STR(s) (MT)
- 0 platform(s)
- 0 platform instance(s)
- 0 instrument instance(s)
- 0 detector instance(s)
- 0 deployment(s)
- 0 stream(s)

Study Elements

- 2 DSG(s) (MT)
- 1 DDs (MT)
- 1 SDD(s) (MT)
- 2 study(ies)
- 0 studyrole(s)
- 7 virtualcolumn(s)
- 5 studyobjectcollection(s)
- 1206 studyobject(s)

Data Elements

Data Content

- 3 dataset's data file(s)
- -1 data value(s)

Data Semantics

- 1 semantic data dictionary(ies) (MT)
- 0 semantic variable(s)
- 0 entity(ies)
- 0 attribute(s)
- 0 unit(s)

Figure 14: HASCO/Repo dashboard showing HASCO elements currently stored in the scientific knowledge graph.

values in such row. Within each MT table, the column *hasUri* is used to inform the URI a row will be mapped to in the knowledge graph. A column *type* or *subClassOf* is usually present to characterize a row. If the column is *type*, HASCO/Repo assumes the row is a class instance, and the value of the column identifies the type of the row. If the column is *subClassOf*, then the row is of type *rdf:Class*, and is a subclass of the column's value. Below WP4 further describes each type of MT currently used to capture project and study knowledge.

Instrument (INS) specifications are derived from the VSTO ontology [13], and is a key building block of sensor networks. During the development of the HAScO ontology, it was decided to derive the VSTO-I ontology from VSTO, with the goal of expanding VSTO's description of instruments without interfering with the development of VSTO itself. Within VSTO-I, instruments rely on detectors to feed them with signals captured from target entities. Instruments are responsible for translating detectors' signals into values. Signals can be very diverse ranging from physical properties of environmental objects to the mindset of a human subject. Values captured by a detector are values of a given variable. These values can be either literal, continuous, or categorical.

Study Design (DSG) specifications are where data acquisition activities designed by humans, i.e., scientists and engineers, are planned and executed. These activities acquire data that, once analyzed, should be able to answer scientific questions. DSG specifications are used to capture and preserve knowledge from humans regarding their aims, scientific questions, and data acquisition activities. One important property of an DSG is the nature (or type) of the study, which can be an observation, an empirical experiment, a computational experiment, or a combination of those.

Deployment (DPL) specifications are used to comprehensively describe the measurement infrastructure of a study. A DPL has several tables to capture metadata about the data acquisition infrastructure of the study, including instruments, detectors attached to instruments, and platforms where instruments are deployed. In addition, the DPL allows scientists to define deployments to state the combinations of the aforementioned elements in which data has been acquired.

Data Stream (STR) specifications are used to comprehensively describe the variables associated with each variable of each data file (or data stream). STR specifications are eventually used to bound data files (or data acquisitions (DAs)) to instruments and SDDs, allowing HACO/Repo to ingest data into the SKG, and to semantically annotate each data value added into the SKG with its corresponding Semantic Variable.

4.1.4 PolyglIoT: Middleware Translator for Communication Protocols (Prototype)

Also known as the Multiprotocol Middleware Translator, PolyglIoT provides seamless communication between systems using different communication protocols. For instance, consider a scenario with multiple sensors measuring temperature and communicating via MQTT, while a service processes this data using Kafka. PolyglIoT acts as an intermediary service, enabling communication between these systems with distinct protocols.

PolyglIoT also manages the Quality of Service (QoS) for message delivery, ensuring reliable communication as required. Here, two key terms are introduced: *internal producer/consumer* refers to components within PolyglIoT, and *external producer/consumer* refers to systems interfacing with PolyglIoT.

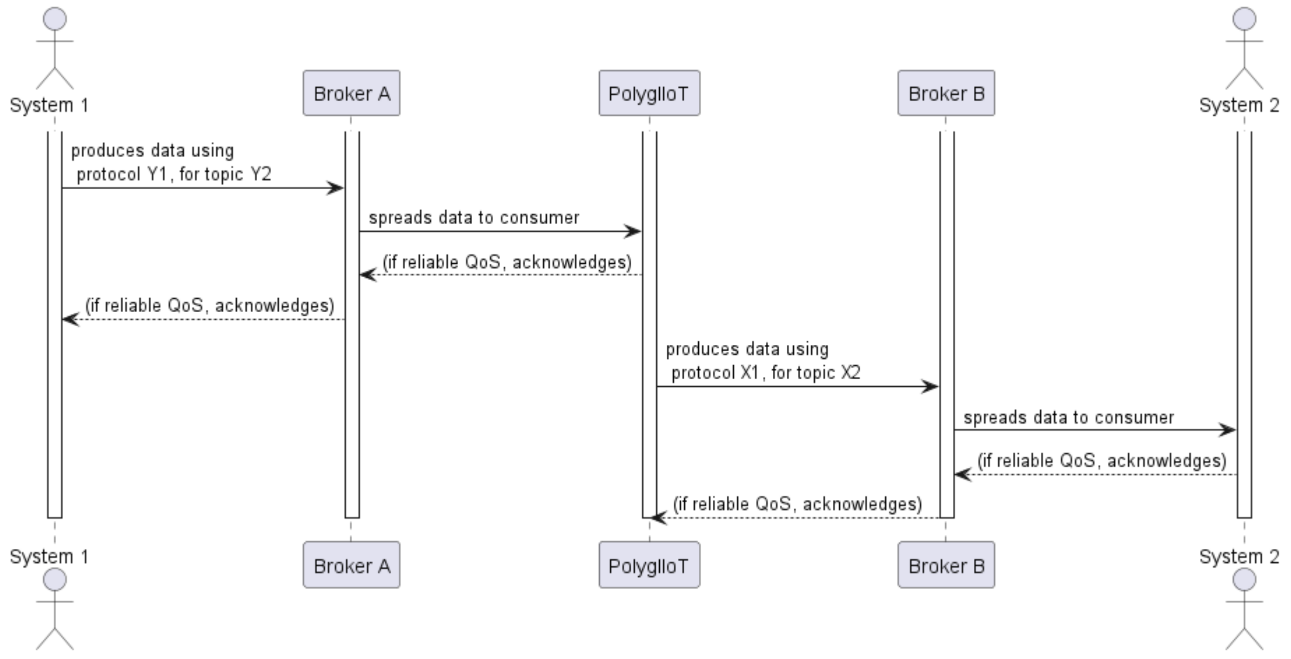

System 1
Broker A
PolyglIoT
Broker B
System 2
produces data using protocol Y1, for topic Y2
spreads data to consumer
(if reliable QoS, acknowledges)
(if reliable QoS, acknowledges)
produces data using protocol X1, for topic X2
spreads data to consumer
(if reliable QoS, acknowledges)
(if reliable QoS, acknowledges)

Figure 15: Runtime Operation of PolyglIoT.

Figure 15 represents, the workflow of PolyglIoT. There are two brokers that interconnect the external producer/consumer and the internal consumer/producer. The external producer (System 1, in this case) publishes messages to the common broker. This broker spreads the message to the subscribers who are interested (subscribed to the same topic). If the delivery QoS is reliable, there is an acknowledgment response from the broker to the external producer, otherwise, it won't know if the message reached the broker. For each message that the internal consumer processes, it produces for the other broker, regarding the different communication protocols. Two design alternatives are proposed: modifying Arrowhead versus keeping Arrowhead unchanged.

Changes on Arrowhead This approach requires additional development in the Orchestrator and Service Registry.

Modifications to Service Registry. The Service Registry (SR) allows systems to define services, systems, and instances. When registering a producer, relevant metadata (QoS, topic/queue, broker address/port, and protocol) is added. However, the SR automatically associates the producer with a compatible broker, as the Orchestrator does not have permissions for system/service creation.

The SR includes a new endpoint, `registerPublisher`, which:

1 - Finds a compatible broker for the producer.

2 - Registers the producer as a system and service instance with a "publisher" definition.

Modifications to Orchestrator. The orchestration endpoint `/orchestrator/orchestration` enables a system to specify the type of service it seeks. In this case, the System is an external producer that wants to find a compatible producer. The orchestration request is going to:

1 - Finds a compatible broker for the external producer.

2 - Finds a compatible internal producer.

3 - Requests PolyglIoT to create the necessary entities (producer and consumer).

4 - Returns the broker's details.

PolyglIoT has a Rest API that allows to create producers/consumers by sending an HTTP request. This approach ensures transparency for systems utilizing protocol translation, offloading responsibilities to the Orchestrator. The only thing that the System should concern is:

- As a producer – request Orchestrator to create a consumer with topic X, QoS Y, and protocol Z.
- As a consumer – request Orchestrator to create a producer with topic A, QoS B, and protocol C.

Meaning this, that Orchestrator is responsible for:

- Find a compatible broker (if needed) for a producer.
- Find a compatible producer for the consumer request.
- Find PolyglIoT service.
- Make the proper requests to PolyglIoT.

No Changes on Arrowhead Contrary to the previous approach, in this approach, the system handles additional responsibilities. For instance:

- As a producer, it must request SR for PolyglIoT's service address/port, find a compatible broker, and properly interact with PolyglIoT.
- As a consumer, it must similarly locate and interface with PolyglIoT and compatible brokers.

Comparison In considering design alternatives for PolyglIoT, each approach presents unique advantages and disadvantages. For the "Changes on Arrowhead" approach, the primary advantage is that it provides full transparency to the system, allowing it to operate without needing awareness of the underlying protocol translations. However, this approach does require code modifications in both the Orchestrator and the Service Registry (SR), which could add complexity to the implementation.

In contrast, the "No Changes on Arrowhead" approach has the advantage of not requiring any code modifications, making it simpler to implement at the cost of added responsibilities. In this setup, the system must directly manage multiple service requests and handle interactions with PolyglIoT and compatible brokers independently, which could increase the workload for systems using the protocol translation feature.

Structure and State of PolyglIoT PolyglIoT is implemented as a middleware translator supporting different QoS levels. PolyglIoT's architecture (Figure 16) supports multiple communication protocols and QoS levels.

- **Blue:** External producers.
- **Orange:** Internal consumers and producers.
- **Green:** Broker/exchange.
- **Red:** External consumers.
- **Black:** Thread management.

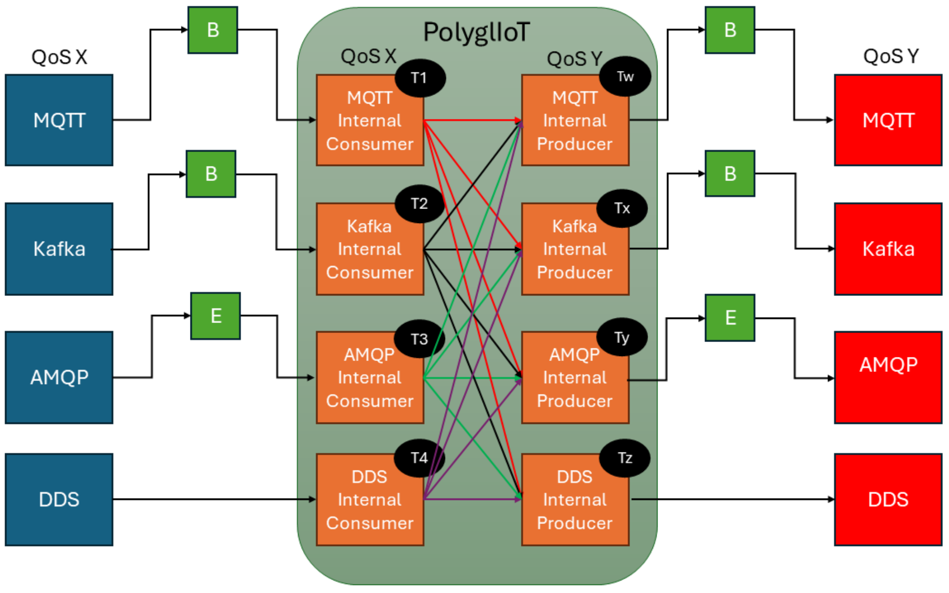


Figure 16: PolyglIoT Architecture.

Several associations between internal consumers and producers are depicted, indicating the potential for translations. It should be noted that each protocol can translate for any other protocol. Each internal consumer has a separate thread waiting for data and every time it consumes something, a thread is created for each linked internal producer. For example, a MQTT external producer publishes a message to the shared broker. After the internal MQTT consumer gathers that data, it creates a thread to run the linked internal producer(es) on the required output protocol. It is noteworthy to remember that DDS doesn't have either a broker nor an exchange due its distributed nature, producer and consumer communicate directly.

Currently, PolyglIoT supports: Four communication protocols:

- Data Distribution Data (DDS) – mostly used on critical embedded applications.
- Kafka – used mainly at IT level.
- MQTT – one of the most used protocols to interconnect with IoT devices.
- AMQP – used mainly at IT level.

Three levels of QoS:

- At Most Once – Sends packets without guarantees of delivery (also known as fire and forget).
- At Least Once – Sends packets and ensures that are delivered, but it might send duplicates.
- Exactly Once – Sends exactly one packet, ensuring that there is no duplication.

Each one of the supported communication protocols has its own specifications and configurations related to QoS delivery. PolyglIoT establishes delivery QoS levels to ensure end-to-end seamless data protocol translation in this regard. PolyglIoT's strategy was to take MQTT and Kafka delivery QoS levels as inspiration, choosing for each protocol a set of properties that could guarantee a similar QoS level.

To guarantee each one of the previously listed QoS's, there are specific protocol properties that need to be activated and configured. Table 4 depicts the mapping for each protocol and QoS.

Table 4: QoS Mapping.

Producer	At Most Once	At Least Once	Exactly Once
Kafka	acks=0, enable_idempotence=false	acks='all', enable_idempotence=false	acks='all', enable_idempotence=true, max_in_flight_requests_per_connection=1
MQTT	qos=0, retained=false, clean_session=true	qos=1, retained=true, clean_session=false	qos=2, retained=true, clean_session=false
RabbitMQ	confirmSelect=false	confirmSelect=true, prefetchCount=10	confirmSelect=false, prefetchCount=1
OpenDDS	reliability.kind=best_effort	reliability.kind=reliable, deadline=5, liveliness.kind=automatic_liveliness_qos	

PolyglIoT Operation PolyglIoT has two main modes of operating: Configuration file and REST API. The mode of operating must be specified in a configuration file that PolyglIoT reads at startup. In `general_properties.json` there is a field designated by `flexible_api` that is a Boolean. If it is assigned as true, dynamic mode is set up (and all the brokers' information below is ignored), otherwise, the static mode is set up. See Figure 4.

- Configuration File Mode. This operation mode is defined as static because it does not allow any kind of modifications in runtime. For instance, let us assume the configuration file is defined to have two producers and one consumer. If a system is interested in producing the same topic as these, a new (internal) consumer is needed. However, using this static mode demands modifications to the configuration file and a restart.

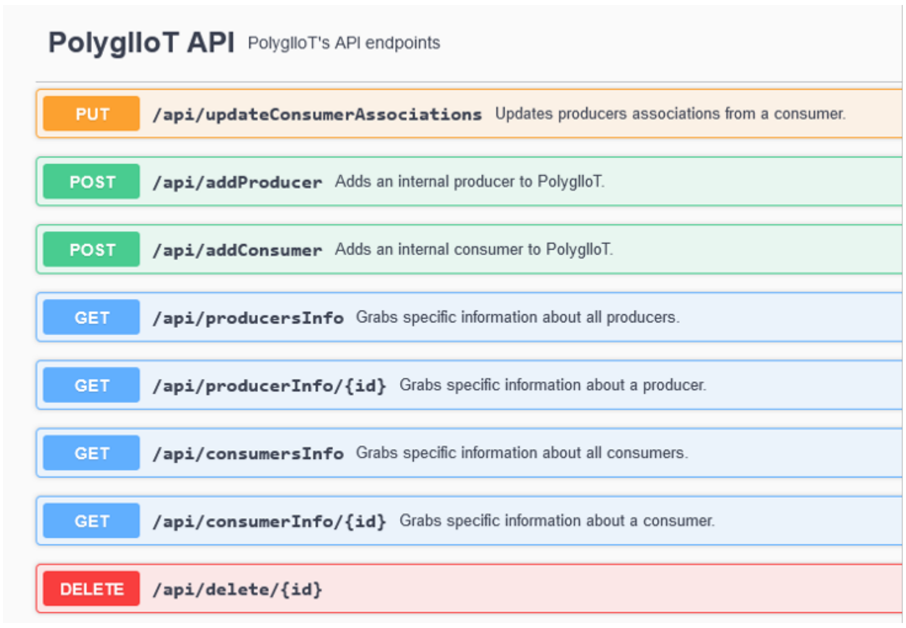
- REST API Mode. This operation mode is defined as dynamic because it allows modifications in runtime. Let us take the same example as the configuration file, it is possible to add a new (internal) consumer. This avoids unnecessary restarts. When this operation mode is set up, PolyglIoT starts listening to HTTP/S requests on port 8080. It is allowed to create producers/consumers, alter producers associated with a consumer, get information about a specific (or all) producer(s)/consumer(s), and delete a specific producer/consumer. It is also prepared to handle input errors with suggestive exceptions being thrown. This API is prepared for multiple requests accessing shared resources using fined-grained synchronization. See Figure 17 for available endpoints. Documentation can be found at `<IP_Address>:8080/swagger-ui/index.html#/`.

Integration with Arrowhead Core Services Integration is possible both with and without modifications to Arrowhead's Orchestrator and SR.

- With Modifications. Requests are managed exclusively by the Orchestrator. External consumers register new producers via SR, while producers request the Orchestrator for compatible brokers and addresses. External consumers need to register new internal producers in the Service Registry, urging to request the Service Registry to "registerPublisher" endpoint. Service Registry is going to check if there is any compatible (that satisfies the system protocol in use) broker and if so, it associates and creates the respective system and service. See Figure 18. External producers need to request Orchestrator to find a compatible broker, a producer that is interested in the same subject/topic, and to grab PolyglIoT's address and port (to send future requests). Then, it sends a producer creation request followed by a consumer creation request, see Figure 19.

- Without Modifications. Without the modifications to the Orchestrator and Service Registry, the System is forced to communicate directly with PolyglIoT. Before communicating with it, one needs to know the address and port. It requests the Orchestrator for this service. Also, it needs to request Orchestrator for a broker, in other words, a service that has a definition of broker and is available for the respective protocol.

PolyglIoT has a Chat Bot that helps working with it. For instance, generating configuration files for static initialization. This Chat Bot can give clear instructions on how to configure OpenDDS with JNI, after installing



PolyglloT API PolyglloT's API endpoints

PUT /api/updateConsumerAssociations Updates producers associations from a consumer.

POST /api/addProducer Adds an internal producer to PolyglloT.

POST /api/addConsumer Adds an internal consumer to PolyglloT.

GET /api/producersInfo Grabs specific information about all producers.

GET /api/producerInfo/{id} Grabs specific information about a producer.

GET /api/consumersInfo Grabs specific information about all consumers.

GET /api/consumerInfo/{id} Grabs specific information about a consumer.

DELETE /api/delete/{id}

Figure 17: PolyglloT Endpoints.

the compiler for C++. In addition, it is also capable of, for example, enumerating the available endpoints of the REST API or even explaining the advantages of using this API.

Several kinds of tests were made to evaluate the system's performance, efficiency, and functionality. System performance tests consist of a stand-alone testbed that evaluates the production speed for each protocol as producer/consumer. Multi-translation tests evaluate the system's capacity to perform simultaneous translations. Unit tests, assure that the system is running as expected and imposed by the diverse requirements.

Several tests were conducted, spanning different protocol combinations and delivery QoS levels. Data delivery rate results confirmed the intended behavior, and it was verified the increase in data delivery reliability for higher QoS, which happened at the expense of a decrease in data delivery rate. This was particularly visible for Kafka external producers. Our results partially confirm that the superior performance in message dispatching of Kafka when compared to MQTT and RabbitMQ clients. Moreover, it was found that this holds true independently of serving as an external consumer or producer. However, it is also the protocol most affected by changes in QoS.

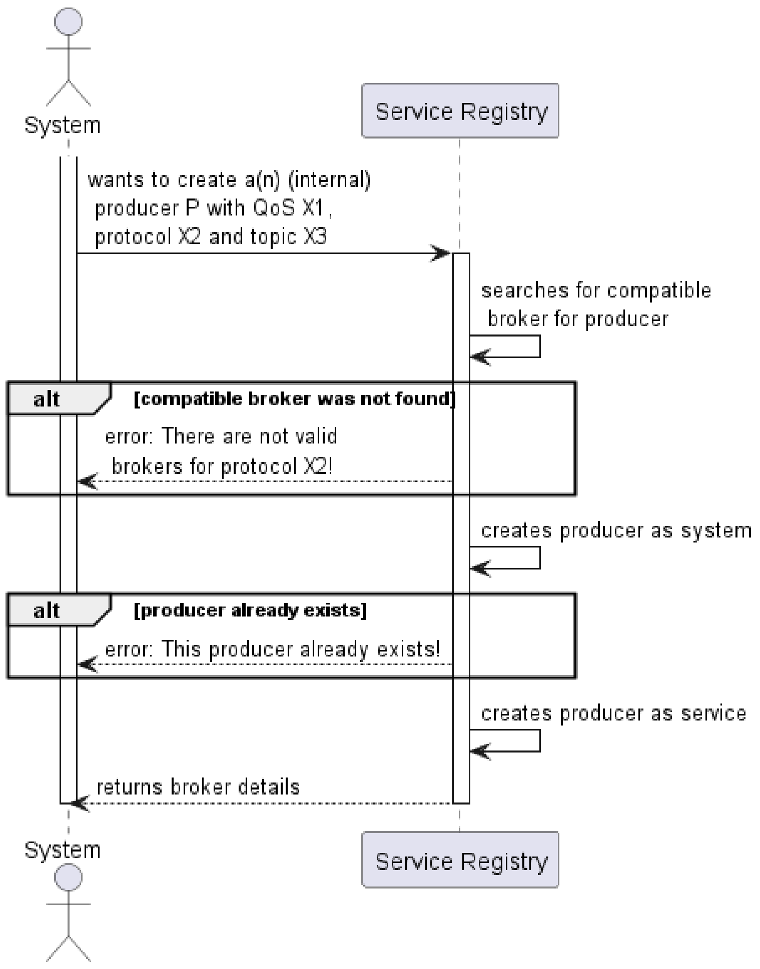

System
Service Registry
wants to create a(n) (internal) producer P with QoS X1, protocol X2 and topic X3
searches for compatible broker for producer
alt
[compatible broker was not found]
error: There are not valid brokers for protocol X2!
creates producer as system
alt
[producer already exists]
error: This producer already exists!
creates producer as service
returns broker details
System
Service Registry

Figure 18: Sequence diagram for the use with modifications- Service Registry.

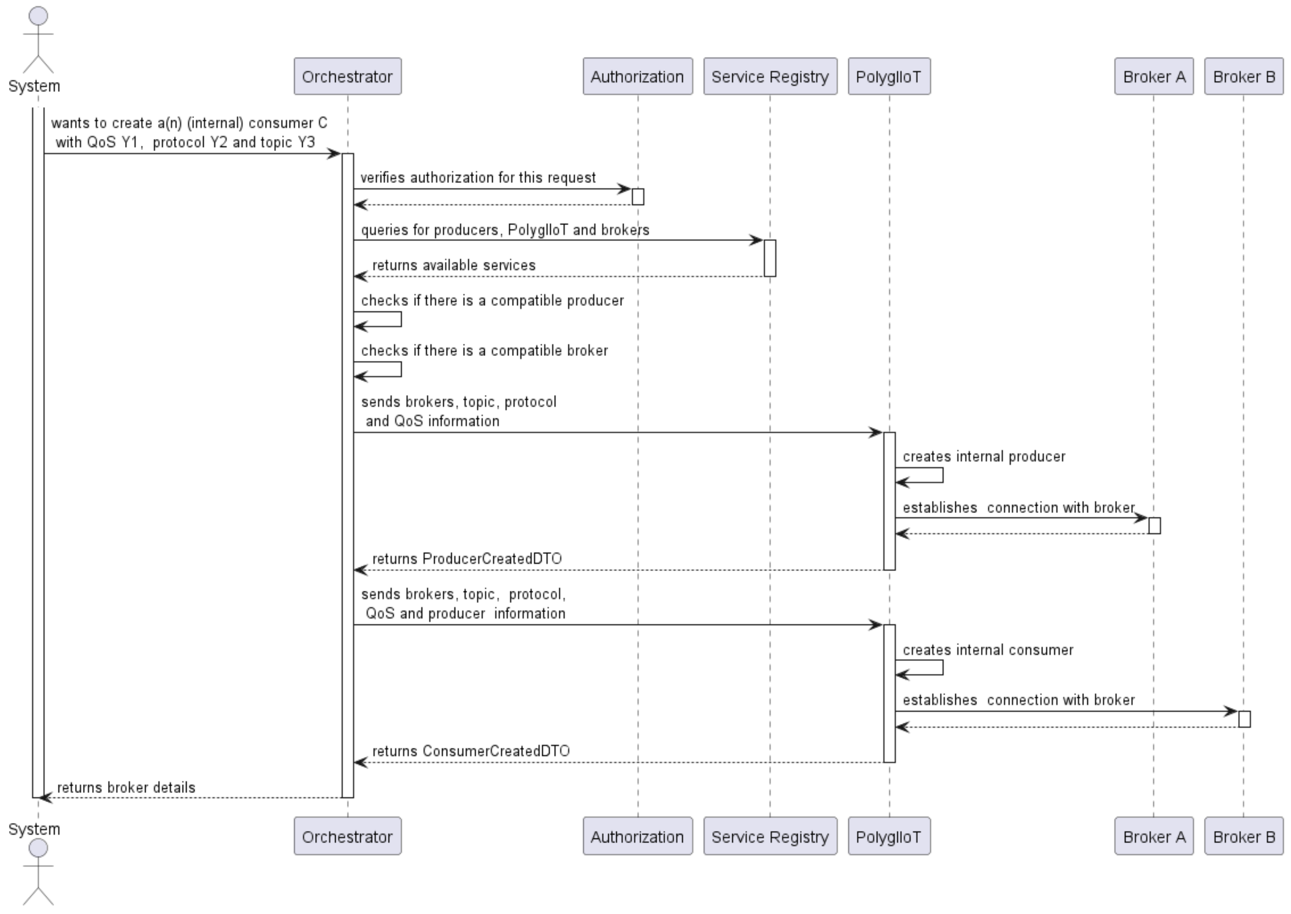

System
Orchestrator
Authorization
Service Registry
PolyglIoT
Broker A
Broker B
wants to create a(n) (internal) consumer C with QoS Y1, protocol Y2 and topic Y3
verifies authorization for this request
queries for producers, PolyglIoT and brokers
returns available services
checks if there is a compatible producer
checks if there is a compatible broker
sends brokers, topic, protocol and QoS information
creates internal producer
establishes connection with broker
returns ProducerCreatedDTO
sends brokers, topic, protocol, QoS and producer information
creates internal consumer
establishes connection with broker
returns ConsumerCreatedDTO
returns broker details

Figure 19: Sequence diagram for the use with modifications - Orchestration.

4.1.5 Bussines Process Ontology-based Adaptor (Proposal)

A focus is placed on ontology-based data model translation tailored for process industry needs. The goal is to develop a method to identify mapping requirements using ontology concepts to enable translations between various data modeling languages.

An investigation was conducted into how ontology-based translation can be integrated within executable business processes for technical information exchange among partners in a complex value network. This conceptual approach addresses use case requirements from UC 3_9, which involve technical information exchange in the process industry, where information is exchanged between business entities over days or weeks, often involving human tasks. For such cases, process digitalization is necessary, meaning that tasks must be standardized and ordered. Business process modeling is proposed as the method to define these interaction sequences. Additionally, dynamic system integrations are needed as the value network evolves because business partners and information systems may change.

To enhance interoperability, an adapter architecture has been designed to embed translations within business processes. At its core, this design uses the Eclipse Arrowhead Framework to identify translation needs during the orchestration process as systems are interconnected. This vision, as illustrated in the following Figure 20, outlines technical information exchange requirements where ontology-based translations are essential. Further work is ongoing to implement this approach in demonstrators.

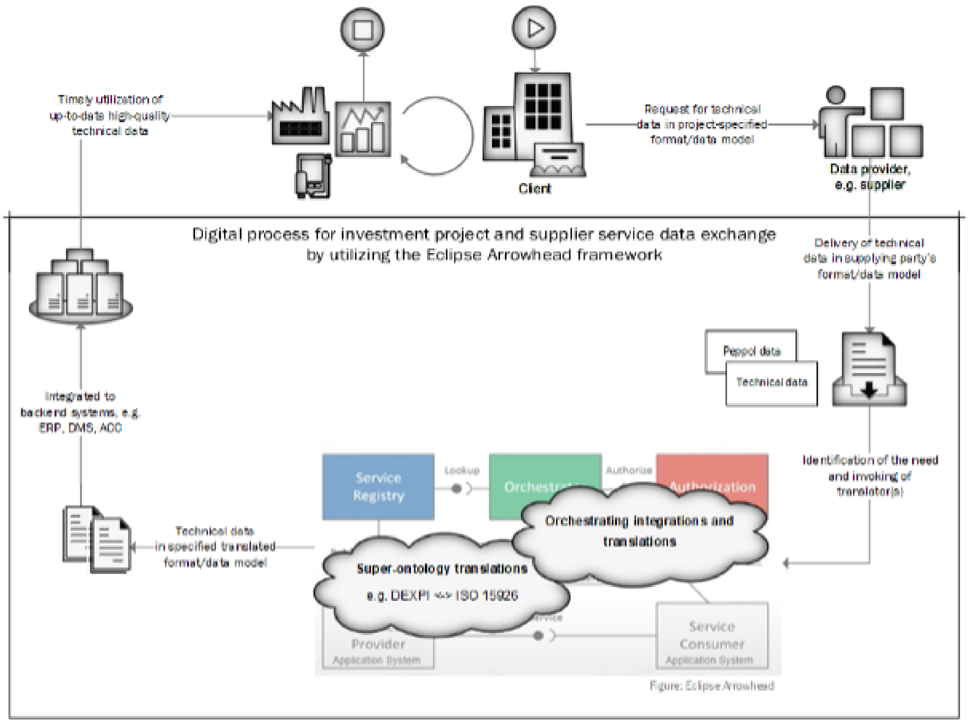

Timely utilization of up-to-date high-quality technical data
Client
Request for technical data in project-specified format/data model
Data provider, e.g. supplier
Digital process for investment project and supplier service data exchange by utilizing the Eclipse Arrowhead framework
Delivery of technical data in supplying party's format/data model
Peppol data
Technical data
Integrated to backend systems, e.g. ERP, DMS, ACC
Identification of the need and invoking of translator(s)
Service Registry
Lookup
Orchestr
Authorize
Authorization
Orchestrating integrations and translations
Super-ontology translations
e.g. DEXPI <-> ISO 15926
Technical data in specified translated format/data model
Provider Application System
Service Consumer Application System
Figure: Eclipse Arrowhead

Figure 20: Adaptor overview scenario.

4.2 AI–Based Translation

The following section summarizes the efforts and progress accomplished during the second year of the project concerning AI-based translation. These efforts encompassed the analysis of requirements and translation scenarios, AI methods, and the design and development of translation solutions.

4.2.1 Selected AI tools and Frameworks to work with standardized data models (Prototype)

This subsection details the techniques and approaches used for AI translation based on standardized data models.

RAG (Retrieval Augmented Generation)

Retrieval-Augmented Generation (RAG) is a powerful technique that combines the strengths of information retrieval and generative models. It involves retrieving relevant information from a large corpus of text and then using that information to generate more informative and accurate text responses. RAG converts the text information into numerical representations (vectors) by applying embedding models. Embeddings capture semantic meaning and relationships between words and phrases. The calculated vectors will be stored in a specialized database optimized for similarity search. After searching a keyword, RAG finds the most similar vectors to a given query, allowing for efficient retrieval of relevant information. It can provide more comprehensive and accurate answers to questions that require specialized knowledge or understanding of multiple concepts. However, this method also has some limitations. One of the main challenges is ensuring that the retrieved information is accurate. If the corpus of text contains inaccurate or outdated information as well, the generated responses may also be incorrect. Furthermore, building the vector store can be computationally expensive, especially for large corpora of text.

In the project, partners work with standardized data models, whose standards often contain or cross-reference more than a thousand pages of documents. By creating a searchable database from the content of these documents, WP4 could solve relevant tasks like:

- Retrieving context information to answer a question regarding a standardized data set.
- Retrieving standardized pattern for a value.
- Retrieve the schema to automatically map between data.

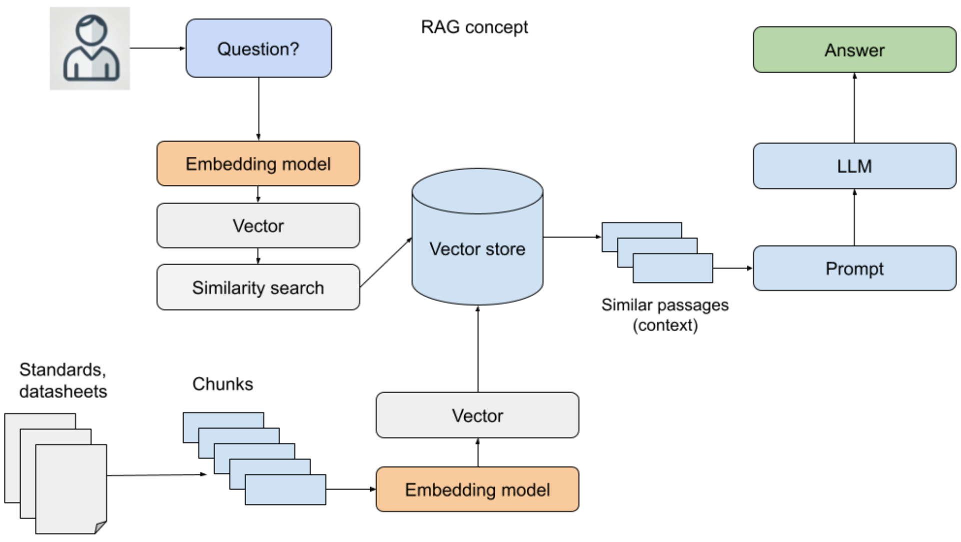


Figure 21: RAG concept.

As a first step, WP4 should create a database (vector store) from the numerical representation of the document set, which was previously split into small, manageable information pieces. To retrieve the relevant passages, RAG operates with a similarity search method, which is based on the distance of the numerical representations. To calculate the results, the human questions should be also mapped to this representation.

After retrieving the best matches, the information pieces will be added to the original question as an extra context, and an LLM will be used to generate an answer. Figure 21 represents the concept of the RAG method.

To create a vector store, WP4 should split the standards, datasheets, or human-readable documents into small chunks. These chunks could represent e.g., one page, or just some lines from the document. This splitting step is based on two parameters:

- The number of characters in one chunk,
- The number of overlapped characters regarding the chunks.

The size of the chunks is related to the content of the document.

The next step is to create the numerical representation (vector) from each chunk. This step could be performed by embedding models, offered by several AI frameworks (e.g., huggingface, or openai).

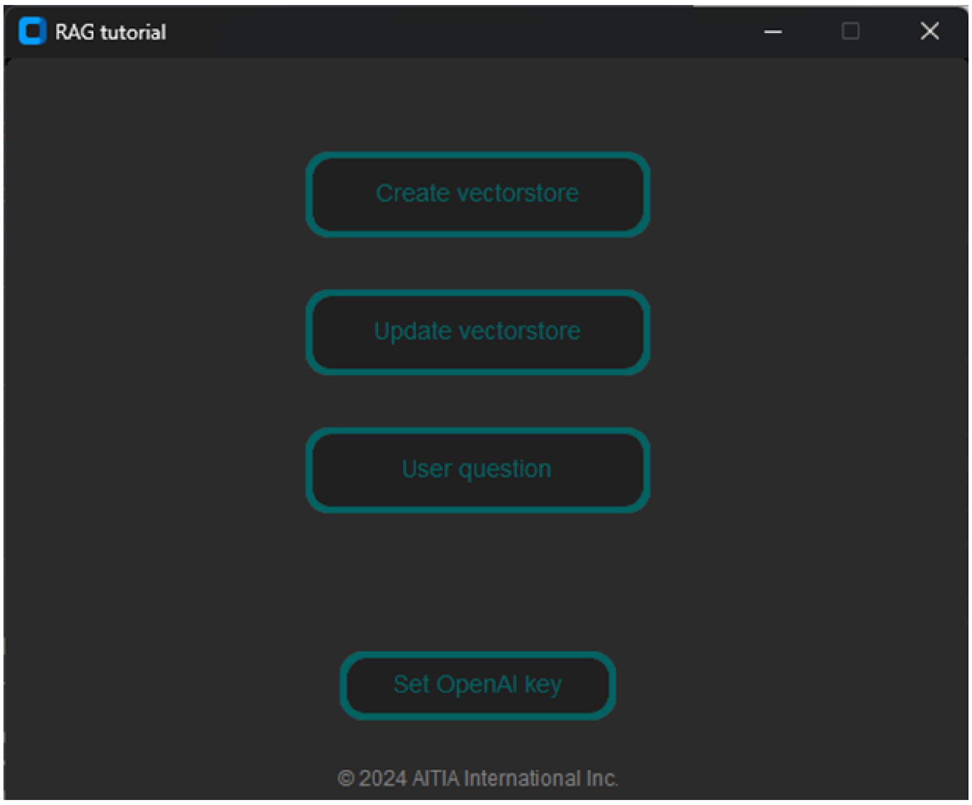


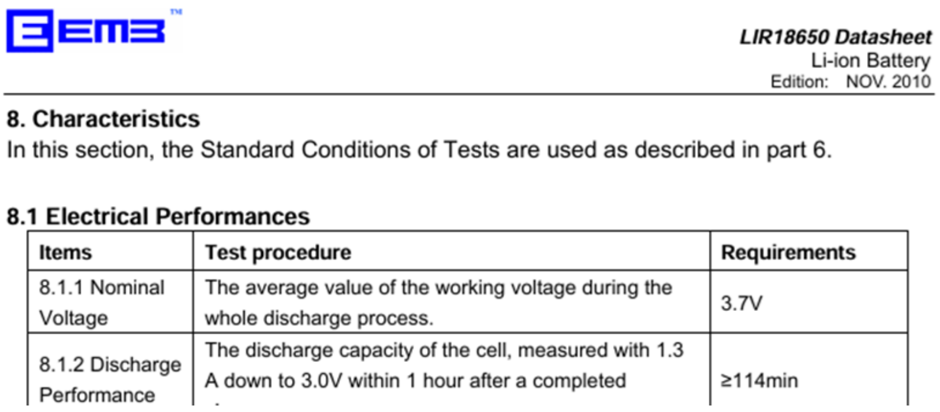
Figure 22: The user interface of the easy-to-use Python code to try the RAG method.

Task 4.2 has created an example Python code [14] with an easy-to-use user interface to create and update a vector store from different types of documents (pdf, YAML, md files, or text files). This example code supports the partners in trying RAG on their datasheets or standards easily. Figure 22 presents the menu of the example RAG project.

After choosing the Create Vector Store button, WP4 could define the chunk size and the overlap size parameters, and the source of the documents. By asking a question, the example code will retrieve the top 5 relevant chunks, and extend the original question with the retrieved information as a context. This context will be used by the LLM to answer.

To demonstrate the usage, WP4 created a vector store from the content of a Lithium-ion Battery datasheet and asked a simple question to retrieve parameters. Figure 23 represents the results, compared to the content of the document (see Figure 24). Table 5 represents the top 5 similar chunks, which were used as the context to answer the question.

To create an effective RAG vector store, the documents should be pre-processed and analyzed. Since the graphs and figures could not be embedded into a RAG vector store, the recognition of the graphs and the


LIR18650 Datasheet
Li-ion Battery
Edition: NOV. 2010

8. Characteristics
In this section, the Standard Conditions of Tests are used as described in part 6.

8.1 Electrical Performances

Items	Test procedure	Requirements
8.1.1 Nominal Voltage	The average value of the working voltage during the whole discharge process.	3.7V
8.1.2 Discharge Performance	The discharge capacity of the cell, measured with 1.3 A down to 3.0V within 1 hour after a completed	≥114min

Figure 23: Part of a Lithium-ion Battery datasheet.

information retrieval from them could be a key feature. The recognition of the pages that contain graphs is not clear. The details are summarized in `Appendix/PDFPreprocessing_AITIA.docx`

LangChain framework

LangChain [15] is a versatile framework designed to simplify the development of applications that leverage LLMs (Large Language Models). It provides a generic API to work with LLM-powered models, which are capable of understanding and answering human questions. The framework also offers various prompt templates and techniques to optimize prompt engineering. LangChain introduces the concept of Agents, which are autonomous entities that can interact with their environment and execute tasks using LLMs (see Figure 25). Agents can be configured to perform a wide range of actions, such as searching for information, executing tools, and making decisions. These tools could be Python functions or pre-trained models as well. The modular design allows easy customization and integration with other components, making it a highly flexible framework. It supports a wide range of use cases, from simple chatbots to complex decision-making systems. The framework is open-source and can be easily extended with new components and features. As the field of AI continues to evolve, LangChain is likely to play an increasingly important role in shaping the future of LLM applications.

To perform automated operations on standardized data, the Agent module could be a possible AI tool in the fPVN project. The advantages are the following:

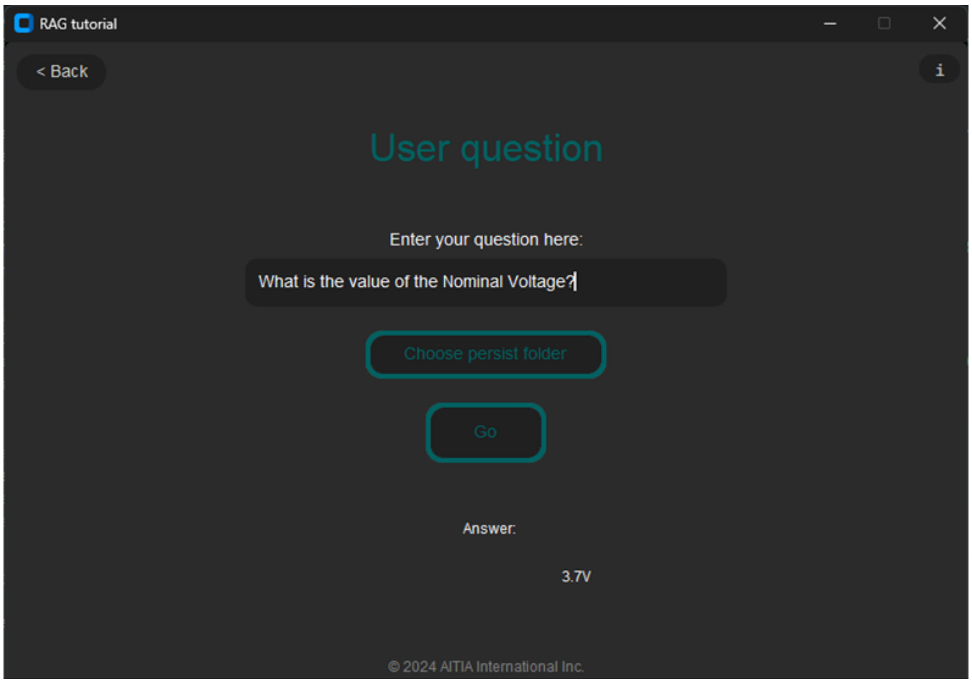
- Operates with LLMs to answer a human question or generate standardized data.
- Operates with a predefined toolset, which toolset could contain Python functions to solve custom, standard-related tasks.
- The solution and the algorithm should not be predefined to solve a problem. The Agent module automatically creates a plan and takes the tools.

LangGraph is an MIT-licensed open-source agent framework that was developed by the LangChain team for building LLM agent and multi-agent applications. The LangGraph concept is independent of the existing LangChain product, but it can be used in combination with the original LangChain framework to expand the abilities of the AI-based application. The main advantage of this framework compared to other agentic frameworks is that the developers could build complex systems using agents as graphs.

Based on the LangChain's definition, an agent is a control flow defined by an LLM. That means that instead of a predefined chain of steps (see Figure 26), the LLM can choose tools from a predefined toolset. This can give more versatile LLM chains, where the LLM can choose the tool, it needs to answer a question. The steps can provide context to the LLM, provide persistent memory, add the ability for the user to approve or edit the state of the LLM, or add the ability to the LLM to search documents.

There are different kinds of agents. Figure 27 represents a router agent, where the LLM chooses from a predefined toolset. As another use case, the LLM can choose freely any kind and number of steps to go through, or even generate its own steps, thus making it a fully autonomous agent.

The Graph system, that the LangGraph framework defines, tries to increase the reliability of the more autonomous LLM chains. It provides them tools in a graph system, but only gives them the ability to choose



What is the value of the Nominal Voltage?

Answer:

3.7V

© 2024 AITIA International Inc.

Figure 24: AI Answer based on the datasheet.

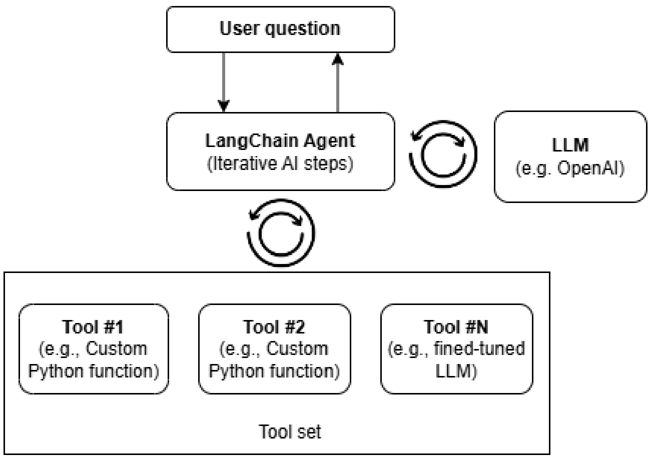


Figure 25: LangChain Agent module concept.

from a limited number of possible paths for each node. These possible state transitions are defined by the edges that connect the nodes of the graph. WP4 can define nodes that have only one edge that goes outward or only goes back to the previous node. The edges are usually defined as one-way paths, but WP4 can also define two-way edges where the system needs the ability to go backward. Every possible path needs to end in the `_end_` node and start in the `_start_` node, or the graph will be invalid.

WP4 can define all kinds of tools as a node:

- Interface for human interaction, where the user could modify the state of the system.
- Search engines.
- Persistent memory.
- Context window to the LLM.
- Functions, which could modify the input of the LLM.

Table 5: Retrieved passages from the vector store.

Nr of the chunk	Content of the chunk
1	8.1 Electrical Performances Items Test procedure Requirements 8.1.1 Nominal Voltage The average value of the working voltage during the whole discharge process. 3.7V
2	5.2 Nominal Voltage 3.7V 5.3 Internal Impedance $\leq 70m\Omega$ 5.4 Discharge Cut-off Voltage 3.0V 5.5 Max Charge Voltage 4.20V 5.6 Standard Charge Current 0.52A
3	Current = 0.52A End Voltage = 3.0V 7. Appearance All surfaces must be clean, without damages, leakage, and corrosion. Each product will have a product label identifying the model.
4	shipping voltage is 3.75-3.80v . because storage at higher voltage may cause loss of characteristics.
5	following: name, type, nominal voltage, quantity, gross weight, date, capacity, and impedance.

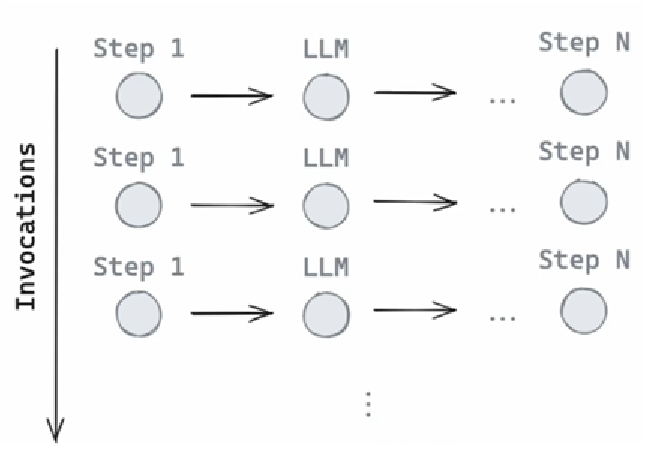


Figure 26: LLM Chain.

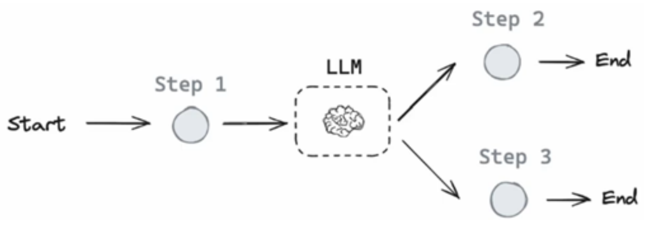


Figure 27: LLM Agent.

- Give the ability to the LLM to interact with other machine learning models.

To use LangGraph, WP4 needs to define graphs by using the Python framework. WP4 can find examples for building systems with varying complexity on the tutorial pages [16] [17]. There is also a desktop application to define LangGraph systems called LangGraph Studio [18], but it is still in beta and WP4 needs a running Docker engine. The application is currently only available on MacOS, but they are claiming that the support for Linux and Windows is coming soon. As this application becomes accessible to a broader user base, the barrier to developing complex LLM systems will be somewhat reduced, making these capabilities more widely available.

LangGraph is a versatile AI framework to work with standardized data. Task 4.2 will apply LangGraph to extend the abilities of the proposed model.

4.2.2 LangChain Agent-based model to work with standards (Proposal)

To present the possibilities of the combination of the RAG method and the LangChain Agent concept, an AI model (see Figure 28) was proposed. To test the approach telecommunication-related datasets have been used and created a vector store from 3GPP standards.

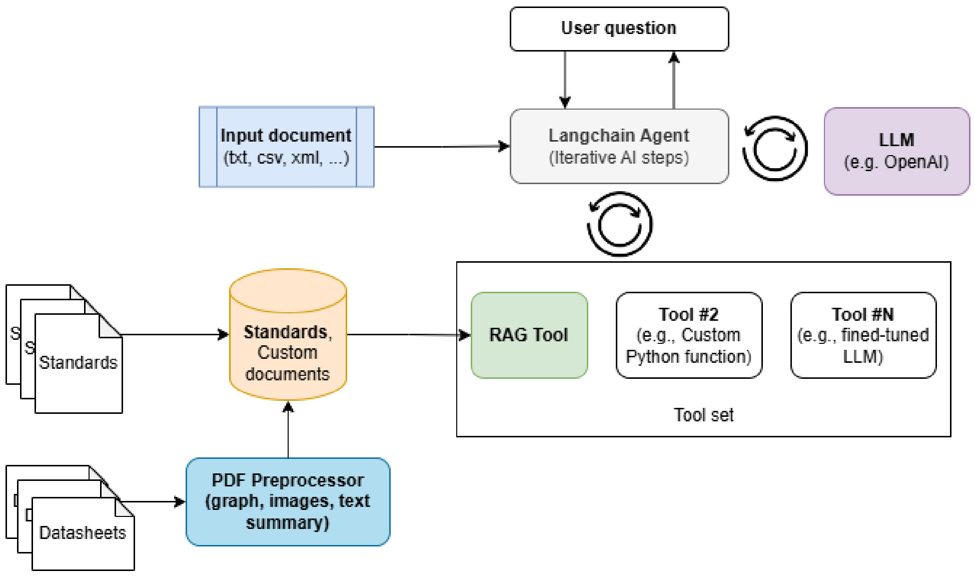


Figure 28: Proposed AI model to work with standards and standardized data.

Task 4.2 investigated two use cases regarding working with standards:

- Answering human questions about the standardized data set.
- Generate values in a standardized format and with standardized content.

The scope of this project is to combine a 3GPP standards-related vector store with LangChain's reasoning engine to determine which actions to take. Since the Agent module applies an LLM call to determine the plan, the user does not need deep telecommunications knowledge. The algorithm is not predefined; the plan is calculated before solving a task or answering a question. Some basic and domain-specific tools have been specified in Python and made available for the Agent module to use. The user can ask questions regarding 5G control plane signaling without needing to know which parameters should be investigated or which transactions should be collected and filtered.

The results were submitted as a publication to the NOMS-2025 (IEEE/IFIP Network Operations and Management Symposium) conference [19]. The publication is currently under review.

4.3 Model–Based Translation

The following section summarizes the efforts and progress accomplished during the second year of the project concerning model-based translation. These efforts encompassed the analysis of requirements, SysML versions, and the design and development of translation solutions.

4.3.1 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. 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 guarantees 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 modeling constructs and validation support, which are beneficial for defining rigorous well-formedness criteria within the 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, the *AHT* Library was devised in SysML v2, as a re-interpretation of the already existing SysML profile for SysML v1.6. This library consists of a foundational structure with three components: Library, Domain, and Example, laying the groundwork for structured, scalable validation within the Arrowhead fPVN framework. The fPVN ruleset for SysML v2, planned in the future for this library, would not only support enhanced model validation but also align with Arrowhead's goal of facilitating high levels of interoperability across diverse system architectures, reinforcing SysML v2 as a robust language for advanced model validation.

For illustration, the following listing shows an excerpt of the new library, in particular, a representation of the IDD (Interface Design Description) concept crucial for model-based translation interactions:

```
abstract port def ServiceDefinition {
}

port def InterfaceDesignDescription :> ServiceDefinition {
        attribute provided : Boolean := true;
        attribute broadcast : Boolean := false;
        attribute global : Boolean := false;
  }
```

4.3.2 SysML 1.6 to SysML 2 translation (Prototype)

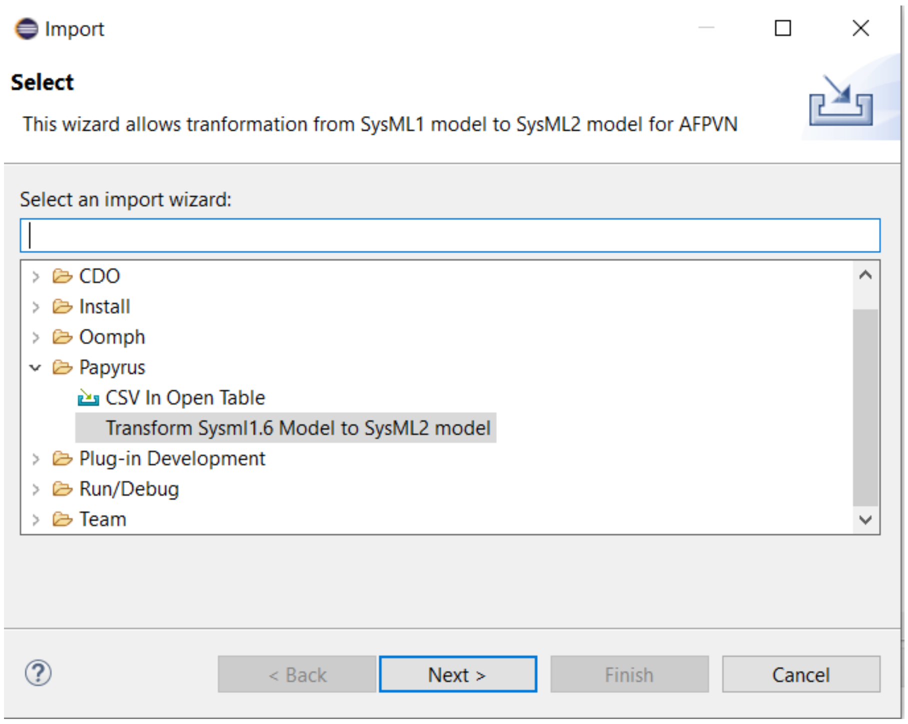
In this subsection, the Papyrus model-based translation from SysML1.6 to SysML2 is detailed. The translation is under development in an add-on plugin that will be installed in Papyrus as an update site. In order to test the translation, the user should build using Papyrus as its source model and select the menu *"Import → Papyrus"* and select the Transformation " Transform Sysml1.6 Model to SysML2 model" as shown in Figure 29.

Once the transformation is selected, the user hits Next and a popup to select his SysML1.6 file is shown (see Figure 30).

The tool will create automatically a SysML2 project named "Sysml2Project" from the Sysml1 file, import the Sysml1 file, and create the equivalent SysML2 file.

Once the user hits Finish, the SysML2 project is created as well as the SysML2 file (.sysml). An example of a Sysml2 serialization used EMF is presented in figure 31. It is worth noting here that the EMF editor of the SysMLv2 models is allowed due to the EMF API plugins (editors) generated from the ecore MM of SysML2.

This Model2Modeltransformation is still under development due to several technical problems mainly the references towards SysML libraries that we can see in Figure 32 (the implicit references) and the xtext serialization. we will test the newer version of the OMG pilot implementation to fix those issues.


Import

Select

This wizard allows tranformation from SysML1 model to SysML2 model for AFPVN

Select an import wizard:

- CDO
- Install
- Oomph
- Papyrus
  - CSV In Open Table
  - Transform Sysml1.6 Model to SysML2 model
- Plug-in Development
- Run/Debug
- Team

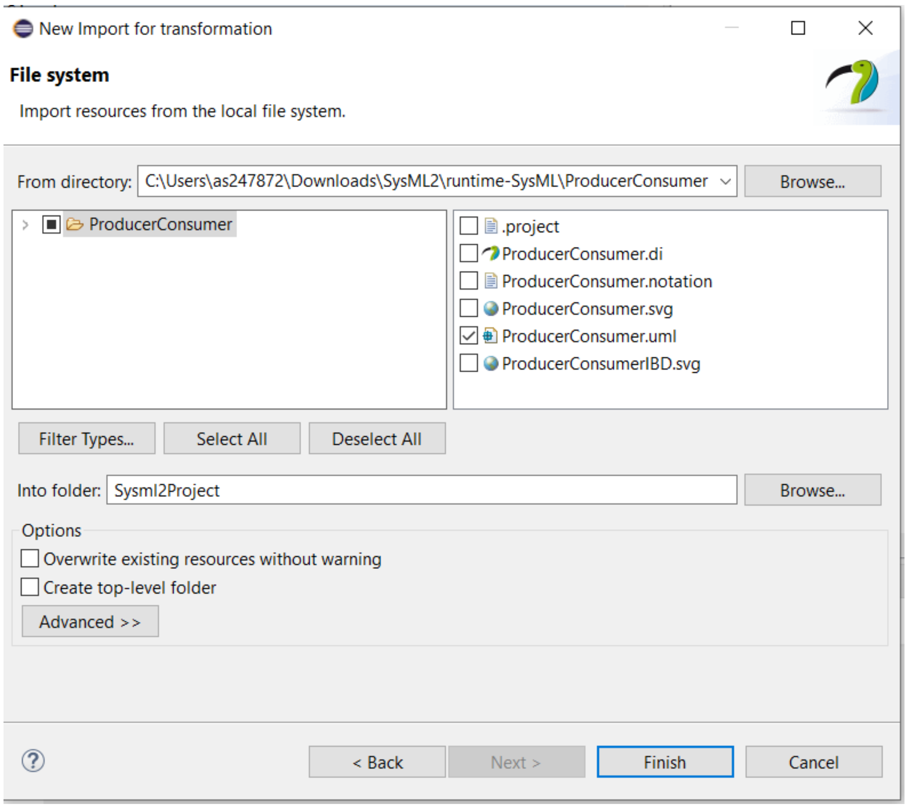
Figure 29: Trigger in Papyrus the transformation from SysML1 to SysML2 models.

4.3.3 UML model generation using LLM (Prototype)

A collaboration between Task 4.3 and Task 4.2 about using LLM to model-based translation has emerged. The result is the development of a new Papyrus UML class diagram design technique using LLMs. The LLM will help 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 a UML Class Diagram of a Library system) as shown in Figure 33 and the LLM will generate the semantic model automatically.

This POC will be presented in the upcoming technical workshop of the AFPVN project. Figure 34 shows the content of the .uml file generated from the LLM in Papyrus as a response to the prompt *”Can you please give me the UML Class Diagram of a Library system in Papyrus ?”*.

In the streaming mode, the LLM modify directly the content of the .uml file. WP4 aims to enhance this work by generating the AFPVN DSL diagram from the user prompt. This will require updating the actual context of the LLM to include the AFPVN example rather than simple class diagram examples.



New Import for transformation

File system

Import resources from the local file system.

From directory: C:\Users\as247872\Downloads\SysML2\runtime-SysML\ProducerConsumer

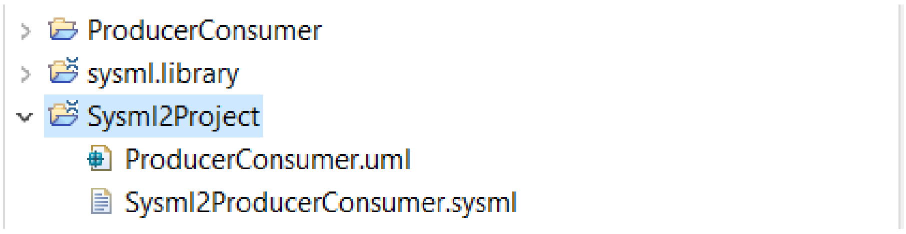
- ProducerConsumer

- [ ] .project
- [ ] ProducerConsumer.di
- [ ] ProducerConsumer.notation
- [ ] ProducerConsumer.svg
- [✓] ProducerConsumer.uml
- [ ] ProducerConsumerIBD.svg

Into folder: Sysml2Project

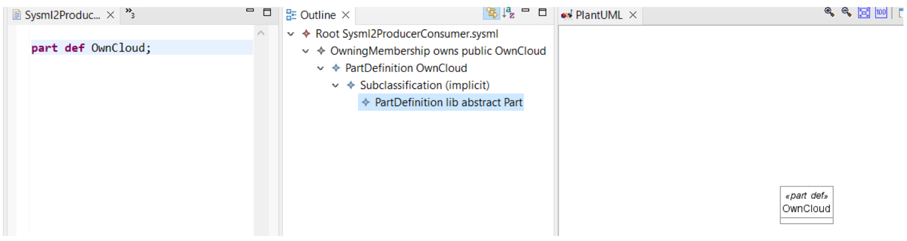
Options
- [ ] Overwrite existing resources without warning
- [ ] Create top-level folder

Figure 30: The user should select the Sysml1 file to be imported into the Sysml2Porject.



- ProducerConsumer
- sysml.library
- Sysml2Project
  - ProducerConsumer.uml
  - Sysml2ProducerConsumer.sysml

Figure 31: the automatically created Sysml2Porject.



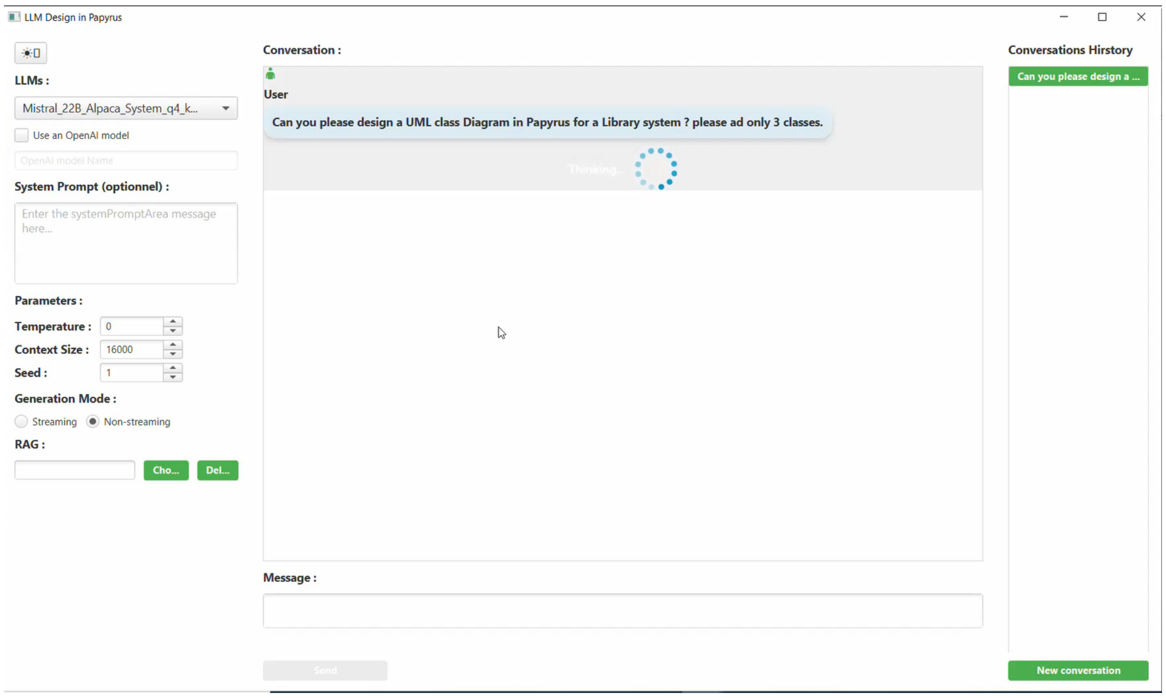
Sysml2Produc...

```
part def OwnCloud;
```

Outline

- Root Sysml2ProducerConsumer.sysml
  - OwningMembership owns public OwnCloud
    - PartDefinition OwnCloud
      - Subclassification (implicit)
        - PartDefinition lib abstract Part

Figure 32: The EMF Serialization of the SysMLV2 model (in the middle).



LLM Design in Papyrus

LLMs : Mistral_22B_Alpaca_System_q4_k...

Use an OpenAI model

OpenAI model Name

System Prompt (optionnel) :

Enter the systemPromptArea message here...

Parameters :

Temperature : 0

Context Size : 16000

Seed : 1

Generation Mode :

Streaming [✓ Non-streaming]

RAG :

Conversation :

User

Can you please design a UML class Diagram in Papyrus for a Library system ? please ad only 3 classes.

Thinking...

Message :

Conversations Hirstory

Can you please design a ...

Figure 33: The Graphical Interface Implemented in Papyrus to chat with LLMs.

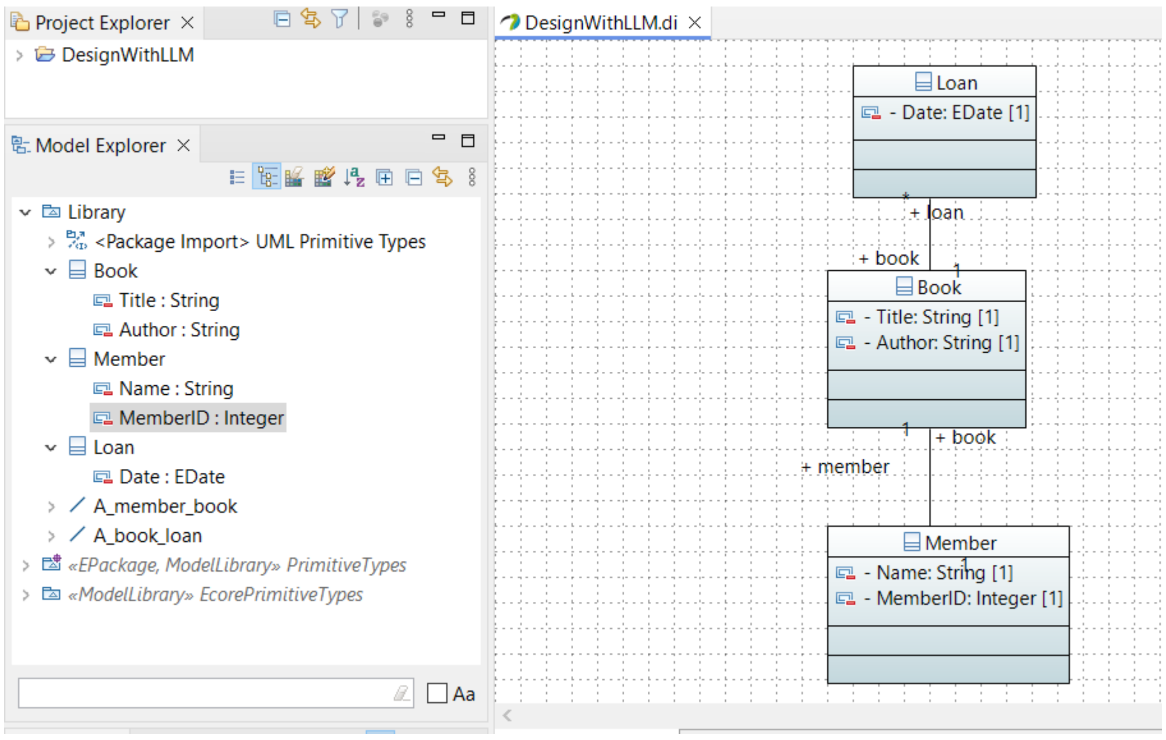


Figure 34: The UML file generated from the LLM in Papyrus.

4.3.4 Exploring Deep-Learning Based Meta-Modelling for Data Translation (Support Activity)

This section presents a comprehensive analysis of contributions to advancing data translation technologies through a hybrid AI framework that integrates Ontology Learning (OL), Knowledge Graphs (KGs), and LLMs. These efforts aim to meet the growing demand for effective data interoperability across various sectors, such as smart manufacturing, transportation, and sensor data fusion.

Recent presentations at the IEEE 19th International Workshop on Semantic and Social Media Adaptation & Personalization [20] and the 16th Symposium on Sensor Data Fusion [21] underscored the potential of hybrid AI models to improve translation accuracy, scalability, and resilience. However, these conferences only introduced preliminary findings, which covered part of the originally proposed objectives. This report extends beyond those initial presentations to outline a complete research strategy, emphasizing further analysis, empirical evaluation, and planned enhancements for publication in a high-impact, rank-1 journal.

The research is structured around four main activities, each designed to tackle different aspects of deep-learning-based data translation, with the goal of building an adaptable, high-accuracy framework. Below is a detailed outline of each activity, including its current status, findings from recent conferences, and steps for future work.

ACTIVITY 1: Comprehensive Literature Review of DL-Based Data-Model Translation Technologies This activity focuses on conducting an exhaustive literature review of state-of-the-art AI methodologies in data model translation, with an emphasis on OL, KG, and LLMs. By analyzing each methodology's strengths and limitations, WP4 aims to understand how a hybrid framework could synergize these approaches for enhanced scalability, robustness, and accuracy. The recent IEEE and SDF conference findings underscored the need for this hybrid approach to address the unique demands of complex industrial environments. The recent conference papers provided an introductory overview of DL-based data translation techniques, showcasing the potential of combining OL, KGs, and LLMs. However, this coverage was not exhaustive and did not fully explore emerging innovations or optimization techniques within these fields. To fulfill the objectives outlined for Activity 1, the upcoming journal paper will extend this review to cover the latest advancements in AI for data translation, including novel hybrid frameworks. Additionally, it will provide a deeper analysis of the scalability, efficiency, and robustness of existing techniques, positioning our work as a definitive reference on advanced data-model translation methodologies.

ACTIVITY 2: Exploration of Large Language Models and Ontology Learning for Enhanced Data Translation Building on the insights presented at the conferences, this activity aims to assess the combined utility of LLMs, OL, and KGs in improving the semantic depth and resilience of data translation models. Specifically, WP4 aims to leverage the contextual understanding capabilities of LLMs to tackle semantic ambiguities inherent in complex data standards like IEEE 1451 and ISO 15926. Initial conference findings introduced the potential of hybridizing LLMs with OL and KGs, demonstrating early-stage improvements in managing semantic nuances. However, these findings were not exhaustive and lacked a comparative analysis of various LLM architectures and their combined performance with OL methods. The hybrid approach's initial evaluation showed promising results in mitigating semantic ambiguities, particularly in scenarios where both structured and unstructured data coexist. However, the complexity of certain industrial standards demands a more nuanced exploration of LLM capabilities, which WP4 plans to address in the next phase.

ACTIVITY 3: Comprehensive Use-Case Development and Analysis for IEEE1451 and ISO15926-4 Translation

This activity involves creating a use-case analysis for translating data between IEEE 1451 and ISO 15926 standards. While the recent conference presentations were limited to a conceptual overview, the full scope of this activity includes both theoretical and empirical validation, focusing on the systematic structuring of semantic relationships in these standards. The aim is to provide a practical model for interoperability that integrates findings from Activities 1 and 2. Due to resource and time limitations, the use-case presented at the conferences remained largely theoretical, lacking real-world data validation and practical implementation.

The journal publication will expand this use-case study to include empirical testing with real IEEE 1451 and ISO 15926-4 data. This hands-on evaluation will allow us to assess the hybrid framework's effectiveness in a practical, industry-relevant context, providing a comprehensive analysis of its applicability to complex, standardized data models. This depth of analysis will be a significant addition beyond the conceptual exploration covered in the conference presentations.

The layered architecture of the hybrid framework—OL for semantic structuring, KGs for relational modeling, and LLMs for linguistic translation—demonstrated initial promise. However, further empirical testing with real-world data is essential to establish the model's practicality in a dynamic, multi-standard environment.

ACTIVITY 4: Evaluation of Existing Translation Frameworks Using the Hybrid AI Framework on the Use-Case Without New Training

The objective of this activity is to evaluate the hybrid AI framework's performance in a real-world use-case scenario without additional model training. By using existing frameworks, the goal is to determine the practical feasibility of the hybrid approach in terms of accuracy, adaptability, and computational efficiency in handling large datasets. The conference papers presented only preliminary results of the framework's initial performance tests. These tests did not fully explore translation accuracy, scalability, and efficiency, nor did they include a detailed comparative analysis with other frameworks.

The findings suggested that hybrid models offer substantial benefits for real-time translation across large datasets without needing extensive re-training. However, the conference presentations lacked a complete evaluation, which will be addressed in the forthcoming journal article.

This work will implement optimization techniques, such as parallel processing and model compression, to reduce computational demands and enhance scalability, making the hybrid framework suitable for real-time applications in complex environments. Additionally, lightweight LLMs and distributed computing methods will be explored to further improve operational efficiency.

Key Areas for Completion and Improvement

- Benchmarking Against Industry Standards - The capabilities of the hybrid framework will be validated through benchmarking against leading translation systems, including industry-standard tools. This comparison will extend beyond the initial scope, positioning the framework as a high-performance solution for data translation.
- Enhanced Metrics for Robustness and Flexibility - This work will expand on robustness and flexibility metrics by introducing user-centric and domain-specific assessments (e.g., interpretability and noise resilience), which are essential for real-world adoption and publication in high-impact venues.
- Iterative Refinement Through Industry Stakeholder Feedback - Collaborative projects with industry partners will provide practical validation, enhancing the framework's real-world relevance and impact. Feedback from pilot implementations will inform further refinements.
- Ethical and Societal Implications of Data Translation - Addressing ethical and societal considerations related to data privacy and interoperability will add depth to this research, appealing to high-impact venues that prioritize responsible AI development.

In conclusion, the proposed hybrid AI framework represents a substantial advancement in data interoperability, combining OL, KGs, and LLMs to create a high-precision, scalable model suitable for diverse, data-intensive industries. While the recent conference papers provided a foundational understanding, they only partially covered the objectives outlined in this report. The planned journal publication will bridge these gaps with a full empirical evaluation, comprehensive benchmarking, and rigorous optimization, establishing a robust model for data translation ready for industry adoption and scholarly recognition.

4.3.5 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 are still in the form of raster images. Converting raster images to more easily processed and modified vector graphics can be performed utilizing neural networks as the diagrams largely consist of repeating symbols. The drawbacks of machine learning methods include erroneous detection and large time investment of both the training data composing 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 minimize the time investment and the required training data size in individual cases. Additionally, generating synthetic training data by augmenting existing diagrams can aid with 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 labeled 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 utilizing diagrams provided by Semantum and some of its customers, notably Afry. All diagrams were vector graphics and rasterized 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 utilized by scaling the diagrams to different sizes before rasterization. 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 1 129 700 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 with 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 favoring FSCE and nine categories favoring MPSR. Comparing the results from the few-shot algorithms directly to YOLO results is somewhat misleading as base training with them utilized 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.

4.3.6 Payload Recognition and Mapping (Support Activity)

Data models are essential for data organization 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, recognizing 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 recognizing 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. 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 serialization 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 serialization 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 deserializes 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 customized 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.

5 Use Cases Collaboration and Support

The following section outlines the collaboration between the WP4 and UC to integrate the translation solutions into the industrial scenarios.

5.1 UC 1_6

There is a wide variety of automotive test and measurement systems available. Typically such systems are used in research, development, and End-of-Line (EoL) testing, for example, battery cyclers in a battery test lab. Such devices assess the health of rechargeable batteries, typically on the cell level, but there are cycles for modules and packs available as well. Batteries are characterized by analyzing series of charge and discharge cycles.

Other examples include measurement systems for electrical parameters of inverters, particle analyzers for tire wear or combustion analysis, or in-vehicle test systems (like AVL MOVE). Such systems come from a wide variety of manufacturers. Unfortunately, manufacturers, as well as operators, have not aligned on one common data structure and naming convention. While the data exchange format often uses standard formats (JSON or XML), data is individually structured in a similar, but not equivalent way. There is no generally accepted structure, naming convention, or even semantics. In a fully automated product line, there is a strong need to share and exchange data between those (currently incompatible) systems. If two systems of different suppliers have to exchange data, there is typically the need to create a translator. Such translators are currently often developed with hand-written code, as they typically lack formal descriptions or other means of automatable documentation. This is costly and error-prone.

As a first step, data translation from different measurement systems shall be automated by using AI/ML. In particular, WP4 wants to leverage the abilities of an LLM to “understand” equivalencies of certain data labels. The overview is shown in Figure 35.

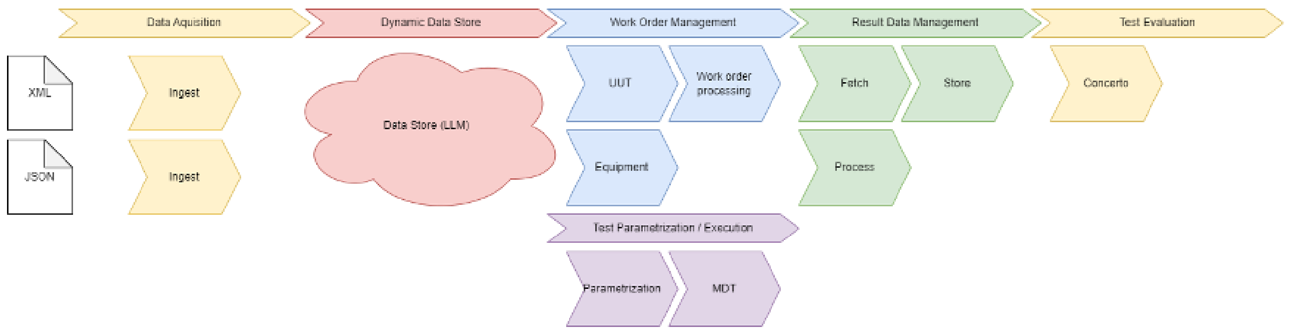


Figure 35: UC architecture overview.

WP4 focuses on the initial steps, where client data in various formats is translated into a common data model, shown in yellow in Figure 35. This approach is outlined in Figure 36, where the AI model is responsible for attribute mapping based on the attributes requested in the input documents, with continuous improvement facilitated by a human-in-the-loop process.

The objective of AI-driven attribute mapping from XML and JSON files is to enable an adaptive method for identifying and linking attributes within files where key names and structural layouts vary widely. This method aims to simplify and automate the retrieval and mapping of data by leveraging AI to interpret and reconcile differences across multiple data sources, allowing for a more cohesive data integration process.

5.1.1 Key Challenges

One of the primary challenges is handling inconsistent key names. Different data sources may use unique naming conventions or abbreviations for the same concept, making it challenging to standardize and accurately match attributes without manual intervention. An AI-based solution can utilize NLP and machine learning models to learn these variations, identifying equivalent terms and establishing semantic relationships among different keys.

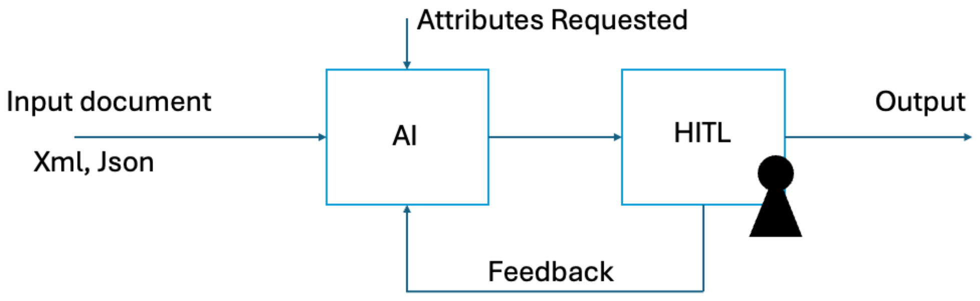


Figure 36: Block diagram of the approach.

Another significant obstacle is the varied data structures within XML and JSON files. These files often represent data in diverse hierarchical or flat structures, requiring a dynamic approach to parse and interpret each format correctly. AI can assist in understanding the intended relationships and context of each element, thereby simplifying data integration.

Additionally, the complex nesting of elements complicates data extraction, as key information may reside in deeply nested fields. AI-driven models can be trained to navigate through these nested structures, prioritizing relevant data and accurately mapping it even from complex hierarchies.

Effective AI-driven attribute mapping must also handle various types of relationships. Many-to-one mapping requires combining attributes from multiple sources into a single destination attribute, while one-to-many mapping distributes one source attribute across multiple destinations. Many-to-many relationships, the most complex of all, involve linking multiple attributes across multiple sources to multiple destinations. AI models can establish these relationships by recognizing patterns in data usage, context, and semantic similarity, significantly reducing the need for manual mapping.

By addressing these challenges, AI-driven attribute mapping provides a scalable and adaptive solution for integrating XML and JSON data into consistent formats, ensuring accurate, efficient data retrieval and mapping across heterogeneous sources.

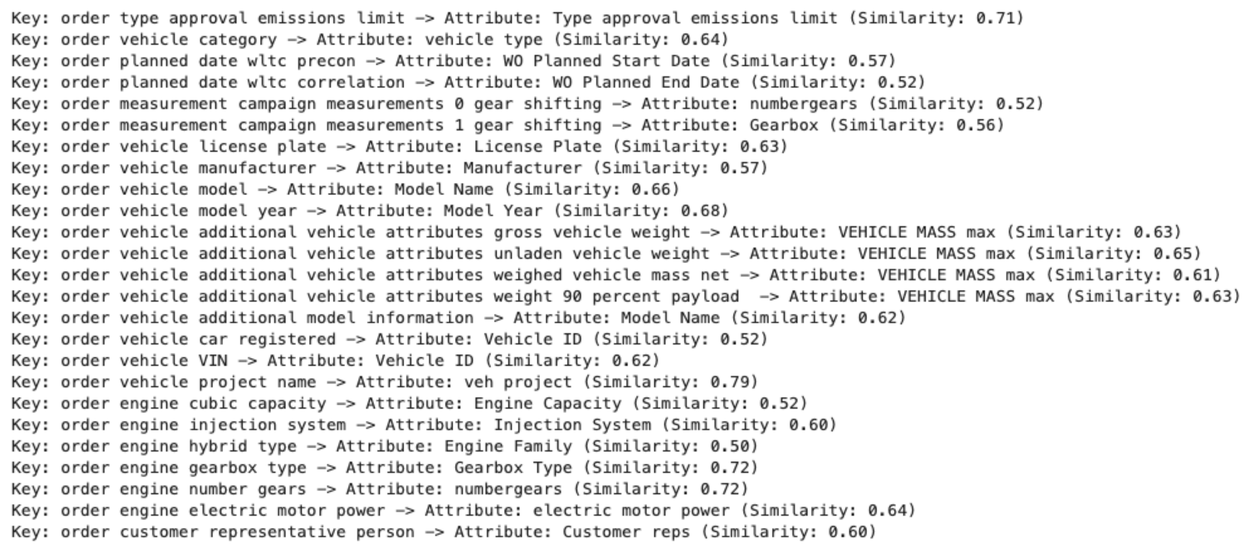
5.1.2 Approach Applied

The problem centers around achieving consistent and accurate attribute mapping and data transformations across heterogeneous data sources, particularly from XML and JSON files. The goal is to develop an intelligent solution capable of managing the inconsistencies in key names, variations in structure, and complex nesting in these data files. The envisioned solution is an AI-driven pipeline that leverages advanced NLP techniques and data-cleaning strategies to enhance attribute mapping precision and overall data integration.

Significant progress has been made in transforming the data to align attributes across diverse sources. This includes the introduction and testing of various algorithms, each tailored to handle different aspects of attribute mapping. The following test implementations were done and are currently under evaluation. Models such as *BERT*, *SpaCy*, Fuzzy Matching, and *SentenceTransformers* have been employed to understand semantic similarities between attribute names, even when they differ across sources. BERT and SpaCy focus on understanding context and semantics, Fuzzy Matching helps handle slight variations and typos, while *SentenceTransformers* effectively captures the relationship between sentence-level meanings, showing promise in accurately mapping similar but differently named attributes.

For optimal performance, specific *thresholds* and *weights* have been introduced within select models and experimental notebooks to refine mapping decisions based on similarity scores. These thresholds act as benchmarks for similarity (e.g., a 0.6 threshold for *SentenceTransformers*), ensuring that only closely related attributes are matched while minimizing mismatches. Further refinements involve *attribute improvements*, such as enhancing descriptions, adding synonyms, and cleaning up noisy data by removing symbols (e.g., underscores, hyphens) and standardizing naming conventions (e.g., converting camelCase). Additionally, overly generic terms (e.g., "type," "name") are filtered out, preventing irrelevant matches and improving the clarity of each attribute.

Among the various approaches, a combination of *SentenceTransformers*, cleaned-up attributes, and synonym mapping with a 0.6 threshold has yielded the most accurate results. This configuration has significantly improved matching precision, especially when the data was preprocessed to remove noise and standardize terminology. To further enhance this approach, weighting models based on attribute importance are planned to balance the influence of specific attributes and fine-tune the system's decision-making process. By implementing these strategies, the envisioned solution is well-positioned to deliver robust, scalable attribute mapping across complex data sources.



```
Key: order type approval emissions limit -> Attribute: Type approval emissions limit (Similarity: 0.71)
Key: order vehicle category -> Attribute: vehicle type (Similarity: 0.64)
Key: order planned date wltc precon -> Attribute: WO Planned Start Date (Similarity: 0.57)
Key: order planned date wltc correlation -> Attribute: WO Planned End Date (Similarity: 0.52)
Key: order measurement campaign measurements 0 gear shifting -> Attribute: numbergears (Similarity: 0.52)
Key: order measurement campaign measurements 1 gear shifting -> Attribute: Gearbox (Similarity: 0.56)
Key: order vehicle license plate -> Attribute: License Plate (Similarity: 0.63)
Key: order vehicle manufacturer -> Attribute: Manufacturer (Similarity: 0.57)
Key: order vehicle model -> Attribute: Model Name (Similarity: 0.66)
Key: order vehicle model year -> Attribute: Model Year (Similarity: 0.68)
Key: order vehicle additional vehicle attributes gross vehicle weight -> Attribute: VEHICLE MASS max (Similarity: 0.63)
Key: order vehicle additional vehicle attributes unladen vehicle weight -> Attribute: VEHICLE MASS max (Similarity: 0.65)
Key: order vehicle additional vehicle attributes weighed vehicle mass net -> Attribute: VEHICLE MASS max (Similarity: 0.61)
Key: order vehicle additional vehicle attributes weight 90 percent payload  -> Attribute: VEHICLE MASS max (Similarity: 0.63)
Key: order vehicle additional model information -> Attribute: Model Name (Similarity: 0.62)
Key: order vehicle car registered -> Attribute: Vehicle ID (Similarity: 0.52)
Key: order vehicle VIN -> Attribute: Vehicle ID (Similarity: 0.62)
Key: order vehicle project name -> Attribute: veh project (Similarity: 0.79)
Key: order engine cubic capacity -> Attribute: Engine Capacity (Similarity: 0.52)
Key: order engine injection system -> Attribute: Injection System (Similarity: 0.60)
Key: order engine hybrid type -> Attribute: Engine Family (Similarity: 0.50)
Key: order engine gearbox type -> Attribute: Gearbox Type (Similarity: 0.72)
Key: order engine number gears -> Attribute: numbergears (Similarity: 0.72)
Key: order engine electric motor power -> Attribute: electric motor power (Similarity: 0.64)
Key: order customer representative person -> Attribute: Customer reps (Similarity: 0.60)
```

Figure 37: Results from the *SentenceTransformers* approach.

5.1.3 Ontology-based Approach

A study design metadata template for Arrowhead's use case 1_6 was created. In this study, WP4 collected a collection of battery's related data files. Some of these datasets were acquired from the use case's documents themselves. Other datasets were found over the web from NASA's projects and Carnegie Mellow University's projects.

For each registered data file, WP4 retrieved supplementary documentation from their sources, and WP4 made an effort to build human-level data dictionaries (DDs). These DDs were registered inside of HASCO/Repo, and identified to be in the context of their corresponding data files.

WP4 is currently working towards translating these DDs into Semantic Data Dictionaries (SDDs). The process of building these SDDs involves the process of inspecting their DDs, identifying the variables related to each DD, and annotating the many properties of each variable in the SDD with terms coming from associated ontologies. This process of building the SDDs also includes the process of associating new ontologies to HASCO/Repo.

5.2 UC 1_7

UC 1_7 serves as a pilot to test various approaches for the same scenario and compare their effectiveness. The text below provides a detailed update on the current status of the effort, outlining recent developments, progress made, and any challenges encountered.

5.2.1 AI based translation

In line with previous efforts to address data model translation challenges, AI translators are being developed by Task 4.2 to explore different approaches for ensuring interoperability, with a focus on Arrowhead platform-based translators.

EC Model to IPC 2581 translation In the recent period, Task 4.2 has investigated the AI-related translation possibilities to standardize the SAE's custom EC data set based on the IPC 2581 standard's requirement. Task 4.2 has started the work based on the following plan:

1. Definition of an initial EC – IPC 2581 element pair table by hand, to understand the requirements, and to have a reference mapping table.
2. Examining and preprocessing the IPC 2581 schema definition file, to allow an LLM-based XML tag pair definition.
3. Examining the RAG method to create an XML tag pair definition (EC custom tag – IPC 2581 tag).

Column B in the file located at `Appendixes/EC_IPC2581C_AI.xlsx` contains the results generated from Step 1. This column holds the processed data or outputs relevant to the initial step of the analysis, which may include preliminary findings, extracted values, or mappings associated with the EC and IPC 2581 standards.

As a next step, the XML files provided were pre-processed to gather possible custom tags and their respective values. An analysis was conducted to determine whether the value set could facilitate the automatic identification of a Simple Type or a Complex Type and define an IPC 2581 element name for the EC tag. Various combinations were tested (see Figure 38); however, the model could not reliably select an appropriate IPC 2581 element name based on the EC tag name, value set, or tag descriptions. It was ultimately concluded that this approach did not meet the desired objectives.

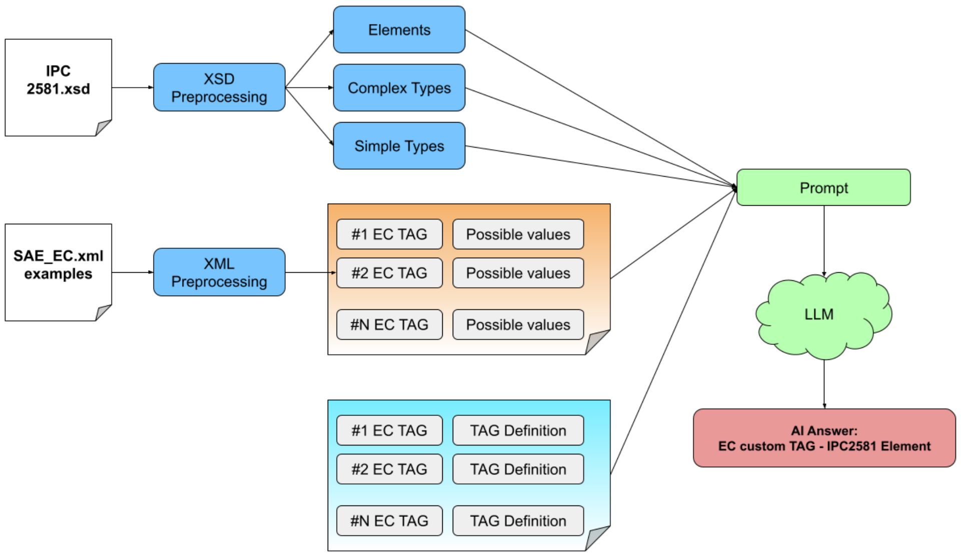


Figure 38: Performed steps based on the IPC 2581 schema definition file.

Since the schema definition alone was insufficient to create an element pair table, further examination of the IPC 2581 standard was conducted. The PDF document contains a structured layout, where elements are described within subsections that can be divided into three parts:

- Element description
- Possible subcomponents (both mandatory and optional)
- Example XML showcasing subcomponents with sample values

To leverage the benefits of the RAG method, the IPC 2581 PDF was preprocessed, splitting element-related subsections into individual text files. This resulted in 199 files, all formatted in Markdown. During preprocessing, images were identified and stored separately for potential future use, while information extraction from the images was intentionally omitted.

To build an RAG vector store, an example project from the RAG method was utilized, with chunk sizes set to 2000 and an overlap size of 100. Figure 39 illustrates these steps.

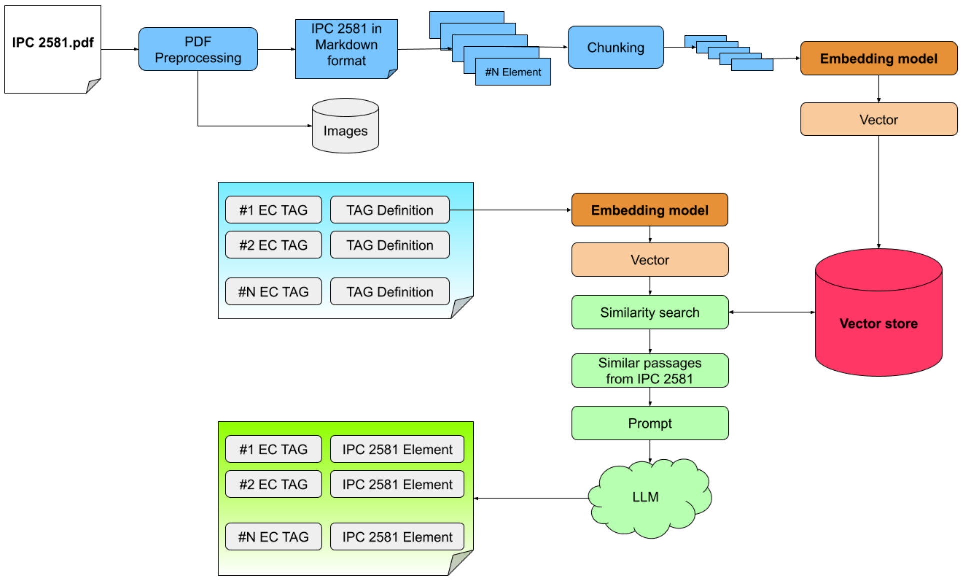
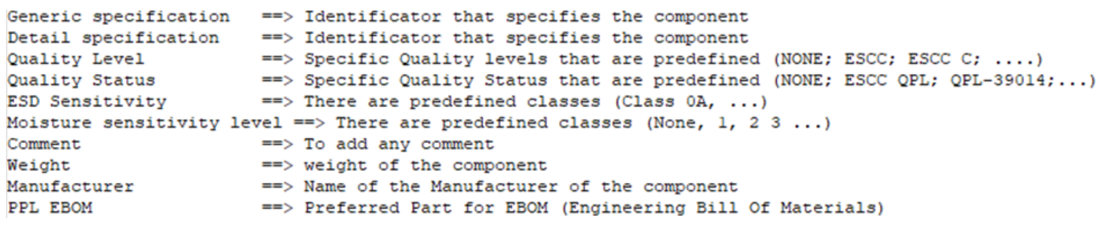


Figure 39: Applying RAG to create the EC custom tag – IPC 2581 element pair definition.

After constructing the RAG vector store, a TAG definition file was applied to retrieve relevant IPC 2581 elements. This file lists custom EC tag names, descriptions, and possible values associated with each tag. Figure 40 provides an example from the TAG definition file.



```
Generic specification     ==> Identificator that specifies the component
Detail specification      ==> Identificator that specifies the component
Quality Level             ==> Specific Quality levels that are predefined (NONE; ESCC; ESCC C; ....)
Quality Status            ==> Specific Quality Status that are predefined (NONE; ESCC QPL; QPL-39014;...)
ESD Sensitivity           ==> There are predefined classes (Class 0A, ...)
Moisture sensitivity level ==> There are predefined classes (None, 1, 2 3 ...)
Comment                   ==> To add any comment
Weight                    ==> weight of the component
Manufacturer              ==> Name of the Manufacturer of the component
PPL EBOM                  ==> Preferred Part for EBOM (Engineering Bill Of Materials)
```

Figure 40: Example content from SAE's TAG definition file.

To retrieve the IPC 2581-related element name from the RAG vector store, custom tag definitions were combined with the following prompt: *"In which element could be stored the . . . ? Give me 2 element names with subcomponents. Print only the element and the subcomponent."*

E.g., if WP4 would like to get an IPC 2581 element to store the EPPL Group content (which is a custom TAG name), the following prompt could give us possible options:

Human Question: *"In which element could be stored the type of material of the component? Give me 2 element names with subcomponents. Print only the element and the subcomponent."* AI Answer: *BomItem, matDes; Component, matDes*

Columns C and D from `Appendixes/EC_IPC2581C_AI.xlsx` present the retrieved IPC 2581 element names in element/subcomponent format. Table 6 represents an example from the pair definition table.

The content of the table could be used to map a custom XML file to the standardized format. Figure 41 presents a possible plan to perform the translation task.

The new element name related to the standard can be easily retrieved from the pair definition table. Using this new element name, an LLM can generate XML content that incorporates the original value. To illustrate this process, a series of steps were followed, as shown in Figure 41, and the subsequent prompt was applied: *"Create an XML content based on the following information. Value: 01 CAPACITORS, Element: Component, Subcomponent: refDes"*

The value of the Element and the Subcomponent was retrieved from the mapping table, and the Value was retrieved from the input XML file.

The AI answer was the following: $< Component >< refDesname = "01CAPACITORS" >< /Component >$

Table 6: Extract from `Appendixes/EC_IPC2581C_AI.xlsx`.

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

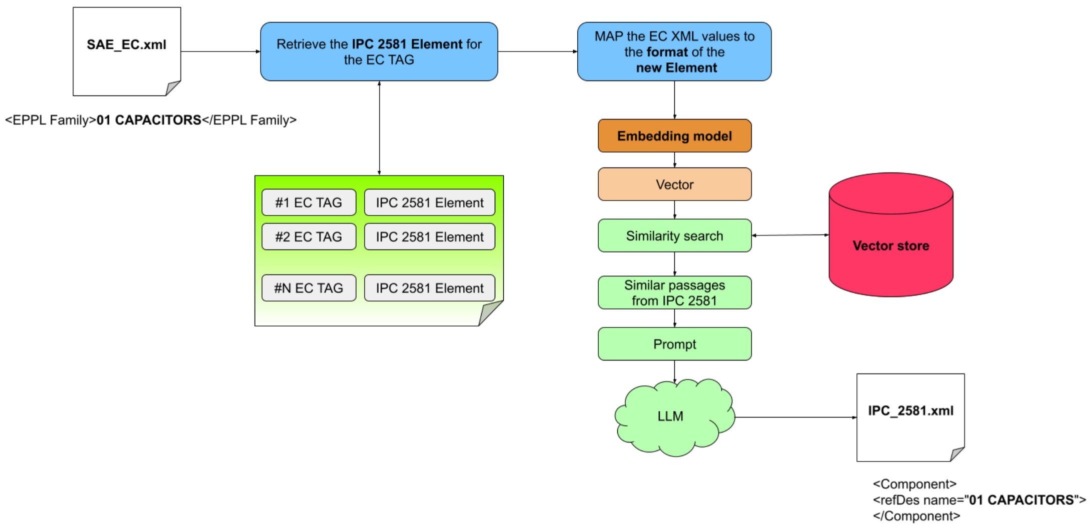

SAE_EC.xml
<EPPL Family>01 CAPACITORS</EPPL Family>
Retrieve the IPC 2581 Element for the EC TAG
MAP the EC XML values to the format of the new Element
#1 EC TAG
IPC 2581 Element
#2 EC TAG
IPC 2581 Element
#N EC TAG
IPC 2581 Element
Embedding model
Vector
Similarity search
Vector store
Similar passages from IPC 2581
Prompt
LLM
IPC_2581.xml
<Component>
<refDes name="01 CAPACITORS">
</Component>

Figure 41: Mapping custom xml to IPC 2581 standardized format.

Based on the prior results, it was determined that the RAG method could be effectively utilized for automatic translation.

Future Work and Next Steps:

- Fine-tuning the chunking method by exploring the separation of element descriptions from content definitions and example XMLs.
- Further developing the Python code to automate the generation of a CSV file from the TAG definition file and insights from the RAG vector store.
- Refining the details of the mapping steps using the generated CSV file. (This task will be undertaken by the responsible team.)
- Adjusting the TAG definitions to ensure the creation of an accurate CSV file. (This task will be undertaken by the relevant stakeholders.)
- Creating a system to facilitate automatic mapping based on the generated CSV file.

5.2.2 Ontology-based Translation

In the ongoing efforts to transition mechatronic projects from design to production, efforts are being made to enhance the management of data model translation between various tools.

WP4 and WP7 are exploring the integration of DITAG with the Arrowhead platform to enhance data handling and improve workflow efficiency. The team has explored its capabilities and evaluated its performance in translating data models for the migration of electronic board designs to the PLM system.

Initially, the IPC-2581 standard and custom data models were considered for the data model translation process. However, the decision to use the first one has been delayed for the first version of the use case demo. Instead, WP4 will proceed with custom data models for now. Despite this delay, WP4 will continue analyzing the possibilities of incorporating IPC-2581 in future phases of the project as a potential standardized data model.

A simple example is provided to illustrate the current progress. In this case, there are two distinct data models: one for the electronic design tool (the provider) and another for the data model used in the PLM tool (the consumer). Figure 42 presents an XML sample from the provider, which the consumer cannot understand, while Figure 43 presents an XML sample that is consumer-friendly.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<electronics_ontology>
    <component>
        <general>
            <libraryRef> E10001919.000</libraryRef>
            <comment>* </comment>
            <description>Resistor, Chip, R0402, 475R, 1%, 100ppm/°C, Gold Plating </description>
            <designator> R </designator>
         </general>
        <parameters>
            <footprint>
                <footprintRef> EEE-10-RES-RM0402</footprintRef>
                <footprintPath>/PCB/EEE-10-RESISTORS.PcbLib</footprintPath>
            </footprint>
            <attributes>
                <CompnentNumber> 510602001 </CompnentNumber>
                <EPPLFamily> 10 RESISTORS</EPPLFamily>
            </attributes>
         </parameters>
    </component>
</electronics_ontology>
```

Figure 42: Provider XML Example.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<product>
    <type>Physical Part </type>
    <name> PRD00096582</name>
    <revision>1.1</revision>
    <title> E10001919.000</title>
    <description>Resistor, Chip, R0402, 475R, 1%, 100ppm/°C, Gold Plating </description>
    <owner> user </owner>
    <maturityState> In Work </maturityState>
    <family> E.-EEE </family>
    <subFamily> 10 RESISTORS</subFamily>
    <quality>0.0</quality>
    <manufacturer>KEMET</manufacturer>
    <enterpriseItemNumber> E10001929</enterpriseItemNumber>
    <symbol>
        <cad>ALTIUM </cad>
        <name> xcadnonps00007606 </name>
    </symbol>
    <package>
        <footprint>
            <title> EEE-10-RES-RM0402</title>
        </footprint>
        <cad>ALTIUM </cad>
    </package>
</product>
```

Figure 43: Consumer XML Example.

The DITAG can automatically generate a translator for this example, provided it has access to the syntax and semantics of both XML messages/documents. The syntax and semantics can be provided in their XML Schemas (XSDs). To illustrate the XSDs, Figure 44 presents the provider XSD with semantic annotations to a reference ontology.

To run DITAG, the BAT file named `ditag-tool.bat` (Figure 45) can be used. DITAG generates a report file (`Report.json`) and a translator (`Translator.java`) when the provider and consumer are compatible.

A key component of the report is the ”match” information. If the provider and consumer are not compatible, the ”match” will be null. If compatible, the report will indicate the number of ”obligatory matches” and ”optional matches” provided by the provider for the given consumer.

```
<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">
  <xs:element name="electronics_ontology">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="component">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="general">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="libraryRef" sawsdl:modelReference="/Product/hasTitle"/>
                    <xs:element type="xs:string" name="comment"/>
                    <xs:element type="xs:string" name="description" sawsdl:modelReference ="/Product/hasDescription"/>
                    <xs:element type="xs:string" name="designator"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="parameters">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="footprint">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="footprintRef" sawsdl:modelReference ="/Product/hasPackage/Package/hasFootprint/Footprint/hasTitle"/>
                          <xs:element type="xs:string" name="footprintPath"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="attributes">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:float" name="CompnentNumber"/>
                          <xs:element type="xs:string" name="EPPLFamily" sawsdl:modelReference="/Product/hasSubFamily"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:annotation>
    <xs:appinfo>
      <a3st:model a3st:ontology="http://gres.uninova.pt/ditag/validation/usecase17/pmlontology01.owl" />
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasType" a3st:value="Physical Part "/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasName" a3st:value=" PRD00096582"/>
      <a3st:comp-property-value type="xs:float" a3st:property="/Product/hasRevision" a3st:value="1.1"/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasOwner" a3st:value=" user "/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasMaturityState" a3st:value=" In Work "/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasFamily" a3st:value=" E.-EEE "/>
      <a3st:comp-property-value type="xs:float" a3st:property="/Product/hasQuality" a3st:value="0"/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasManufacturer" a3st:value="KEMET"/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasEnterpriseItemNumber" a3st:value=" E10001929"/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasSymbol/Symbol/hasCAD" a3st:value="ALTIUM "/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasSymbol/Symbol/hasName" a3st:value=" xcadnonps00007606 "/>
      <a3st:comp-property-value type="xs:string" a3st:property="/Product/hasPackage/Package/hasCAD" a3st:value="ALTIUM "/>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

Figure 44: Provider XML Schema with Semantic Annotations.

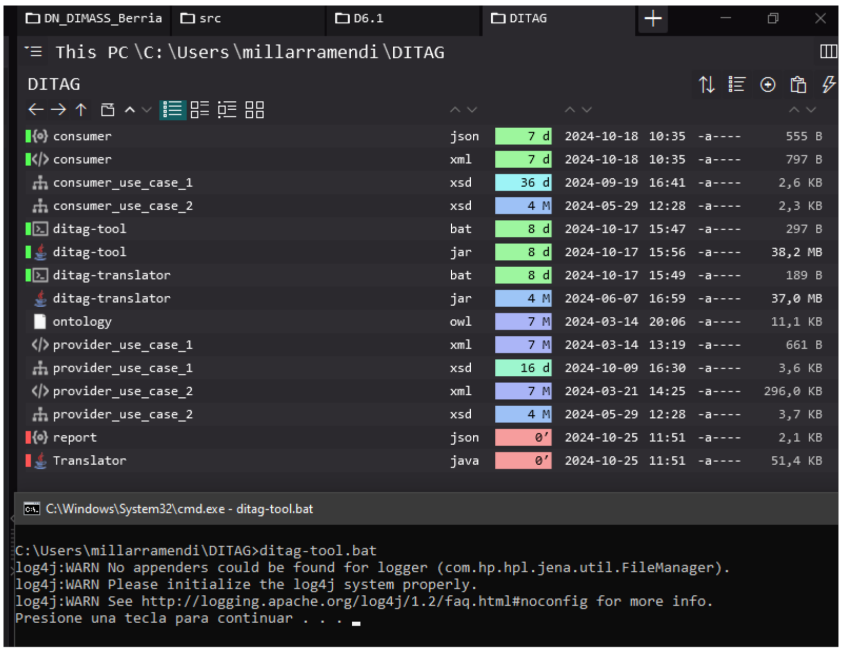
The final step in this process involves running the BAT file named `ditag-translator.bat`, which will use the generated Translator and the provider's XML as inputs, to create the consumer XML data.

Looking ahead, the plan is to deliver a demo or prototype by M24. This demo will showcase the use of the DITAG tool for data model translation, demonstrating the progress and validating the approach. The prototype will help solidify the workflow for future deployments and refine the integration with the Arrowhead platform.

5.2.3 Model-based Translation

WP4 is currently focused on translating data from Developair's Rely tool to the Arrowhead Requirements Model, in collaboration with Task 4.3. This work involves integrating Rely into the Papyrus environment, a key part of the ongoing effort to ensure compatibility with broader industry frameworks.

The translation process includes converting several key inputs from Developair's Rely format into the


DN_DIMASS_Berria
src
D6.1
DITAG
This PC \C: \Users \millarramendi \DITAG
DITAG
C:\Windows\System32\cmd.exe - ditag-tool.bat

```
consumer               json   7 d  2024-10-18 10:35 -a----    555 B
consumer               xml    7 d  2024-10-18 10:35 -a----    797 B
consumer_use_case_1    xsd   36 d  2024-09-19 16:41 -a----    2,6 KB
consumer_use_case_2    xsd    4 M  2024-05-29 12:28 -a----    2,3 KB
ditag-tool             bat    8 d  2024-10-17 15:47 -a----    297 B
ditag-tool             jar    8 d  2024-10-17 15:56 -a----   38,2 MB
ditag-translator       bat    8 d  2024-10-17 15:49 -a----    189 B
ditag-translator       jar    4 M  2024-06-07 16:59 -a----   37,0 MB
ontology               owl    7 M  2024-03-14 20:06 -a----   11,1 KB
provider_use_case_1    xml    7 M  2024-03-14 13:19 -a----    661 B
provider_use_case_1    xsd   16 d  2024-10-09 16:30 -a----    3,6 KB
provider_use_case_2    xml    7 M  2024-03-21 14:25 -a----  296,0 KB
provider_use_case_2    xsd    4 M  2024-05-29 12:28 -a----    3,7 KB
report                 json    0′  2024-10-25 11:51 -a----    2,1 KB
Translator             java    0′  2024-10-25 11:51 -a----   51,4 KB
```

```
C:\Users\millarramendi\DITAG>ditag-tool.bat
log4j:WARN No appenders could be found for logger (com.hp.hpl.jena.util.FileManager).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Presione una tecla para continuar . . .
```

Figure 45: DITAG initialization.

Arrowhead Requirements Model. The goal is to enable the automatic translation of these inputs into the Arrowhead Requirements Model, which will be integrated into Papyrus for system modeling and validation.

This work is being carried out in close collaboration with Task 4.3, and the integration of Rely into Papyrus is a key focus. The translation process will allow for more efficient test case generation and system validation in CPS, aligning Developair's proprietary tools with the Arrowhead framework for improved interoperability and system design.

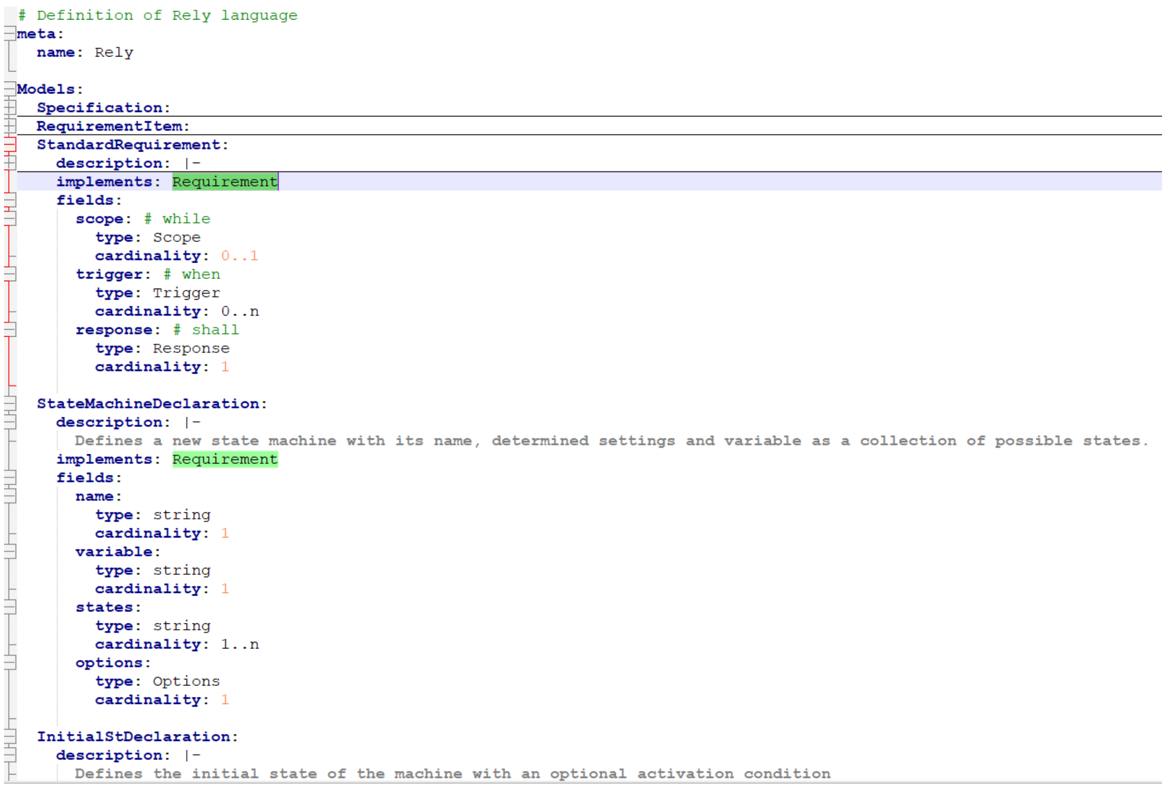
It has been studied the Rely Metamodel provided by the UC, four kinds of requirements are provided: *StandardRequirement*, *StateMachineDeclaration*, *InitialStDeclaration* and *StateTransition*. An overview of the Rely Language is given in Figure 46.

In the current version of the Rely2SysML model-based translation tool, only the StandardRequirement is supported. The three other types of requirements are state-machine-related. For these, the user must specify the Finite State Machine (FSM) with its various states, transitions, and other elements, as the requirements are specific to the FSM. A UML State Machine diagram can be generated from such requirements.

For the structural part of the Requirement (the properties), SysML1.6 is suitable. However, for the expression part, SysMLv2 is more appropriate due to its new Expression and Constraint elements within the SysMLv2 Requirement definition and usage. Additionally, the textual format of SysMLv2 makes it more user-friendly compared to SysML1.6. An example of a StandardRequirement is illustrated in Figure 47

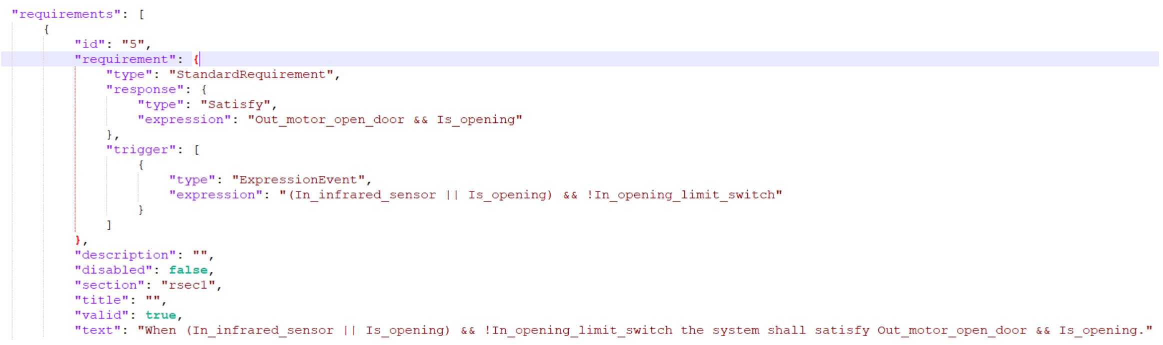
Although SysMLv2 enhances the expression language and adds the capability to associate constraints directly with requirements, the SysMLv2 Requirement definition should be extended to account for the Scope, Trigger, and Response of the Rely Requirements, along with the valid and disabled attributes.

Task 4.3 has therefore studied various mechanisms for extending the SysMLv2 language, primarily focusing on Specialization (using the redefine and subset mechanisms) and the metadata mechanism. The latter will be utilized.



```
# Definition of Rely language
meta:
  name: Rely

Models:
  Specification:
  RequirementItem:
  StandardRequirement:
    description: |-
    implements: Requirement
    fields:
      scope: # while
        type: Scope
        cardinality: 0..1
      trigger: # when
        type: Trigger
        cardinality: 0..n
      response: # shall
        type: Response
        cardinality: 1

  StateMachineDeclaration:
    description: |-
      Defines a new state machine with its name, determined settings and variable as a collection of possible states.
    implements: Requirement
    fields:
      name:
        type: string
        cardinality: 1
      variable:
        type: string
        cardinality: 1
      states:
        type: string
        cardinality: 1..n
      options:
        type: Options
        cardinality: 1

  InitialStDeclaration:
    description: |-
      Defines the initial state of the machine with an optional activation condition
```

Figure 46: A snippet of the Rely Language MM.



```
"requirements": [
    {
        "id": "5",
        "requirement": {
            "type": "StandardRequirement",
            "response": {
                "type": "Satisfy",
                "expression": "Out_motor_open_door && Is_opening"
            },
            "trigger": [
                {
                    "type": "ExpressionEvent",
                    "expression": "(In_infrared_sensor || Is_opening) && !In_opening_limit_switch"
                }
            ]
        },
        "description": "",
        "disabled": false,
        "section": "rsec1",
        "title": "",
        "valid": true,
        "text": "When (In_infrared_sensor || Is_opening) && !In_opening_limit_switch the system shall satisfy Out_motor_open_door && Is_opening."
```

Figure 47: An example of a StandardRequirement.

5.3 UC 2_7

In the first year of the project, it was presented to WP3 and WP4 participants the proposed workflow for the development of "AeroBridge", the S5000F automatic translator. The software is being developed by WP7 with the support of WP3 and WP4 expertise, both to acquire know-how and to provide specific Aerospace-related requirements. To this matter, contributions have also been provided to WP1, WP2, and WP10 surveys.

In the Aerospace Industry, the data flow between stakeholders is very often not standardized and subject to continuous changes over time, leading to slowdowns and possible errors during data transcription on a local repository. To solve this issue, the ASD S-SERIES specifications provide guidelines about how to establish an *Integrated Product Support* program and provide the means to exchange data between the different specifications, different partners, contractors, and customers. This results in a messaging specification used to define the structure and format of messages exchanged, where structure might be represented using UML and then data converted into an XML message.

In particular, S5000F, one of the S-SERIES specifications has been selected for the project since it specifies information and data elements required to provide in-service data feedback. Though originating mainly from the aerospace domain, S5000F can be applied to any products, whether mobile or not, in the air, land, sea, and space domains, both for civilian and military programs. Therefore, the translation aims for the UC "AHMS for Trend Monitoring, Predictive Maintenance, and Fleet Operations & Maintenance Simulation" is to develop a translation method based on S5000F, to lay the foundations of a new automatic translator and validator prototype, "AeroBridge", that converts local data to XML message in compliance with S5000F, and vice versa.

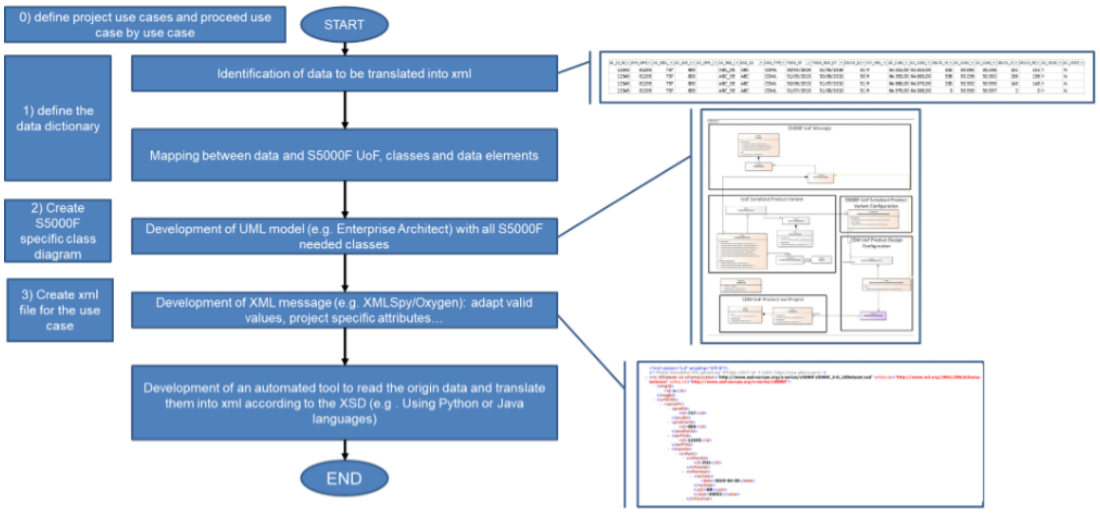

0) define project use cases and proceed use case by use case
START
1) define the data dictionary
Identification of data to be translated into xml
Mapping between data and S5000F UoF, classes and data elements
2) Create S5000F specific class diagram
Development of UML model (e.g. Enterprise Architect) with all S5000F needed classes
3) Create xml file for the use case
Development of XML message (e.g. XMLSpy/Oxygen): adapt valid values, project specific attributes…
Development of an automated tool to read the origin data and translate them into xml according to the XSD (e.g . Using Python or Java languages)
END

Figure 48: Approach followed for developing the model-based translation.

The process adopted to proceed with the development is shown in Figure 48. Information about the Use Cases will be provided in the next section. It has been chosen to adopt the UML as a modeling language in accordance with the S-Series. The UML consists of a set of different modeling 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. The UML Data Model for the S-SERIES Specifications is available online, to be adopted as a starting point for developing (S-Series). To tailor the S5000F Use Cases to the data exchanges selected for this project, Enterprise Architect software has been chosen. It allows one to open the UML download from the S-SERIES site and tailor it by selecting specific Classes, Attributes, and relationships that describe the data model. In terms of the actual translation and data exchange XML is adopted, in accordance with the S-SERIES XSD file available online on the same website as the UML one.

Finally, in terms of the last step of the approach, a Python prototype, the "AeroBridge" is developed to provide two different capabilities in the exchange of data, the Transmission (TX) and the Reception (RX), see Table 7

At this stage, three use cases of data exchange have been identified (Ref. D7.2 "Aerospace Use case first year progress and next step specification"). At this stage, mapping has been done with respect to the Equipment Removal use case (reported in Figure 50), which provides information about the equipment removed from the aircraft due to maintenance.

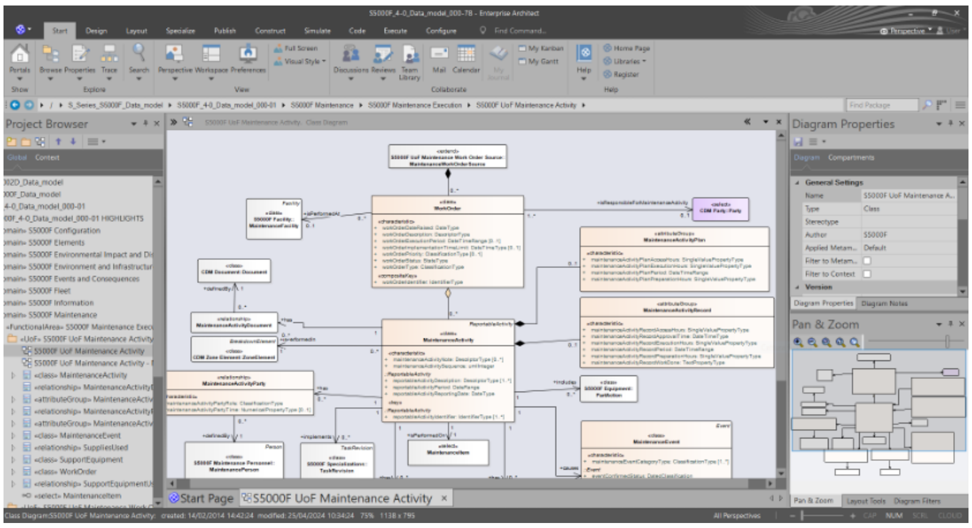

S5000F_4-0_Data_model_000-78 - Enterprise Architect
Project Browser
Diagram Properties
Pan & Zoom
Start Page
S5000F UoF Maintenance Activity

Figure 49: Example of S5000F UML within Enterprise Architect software.

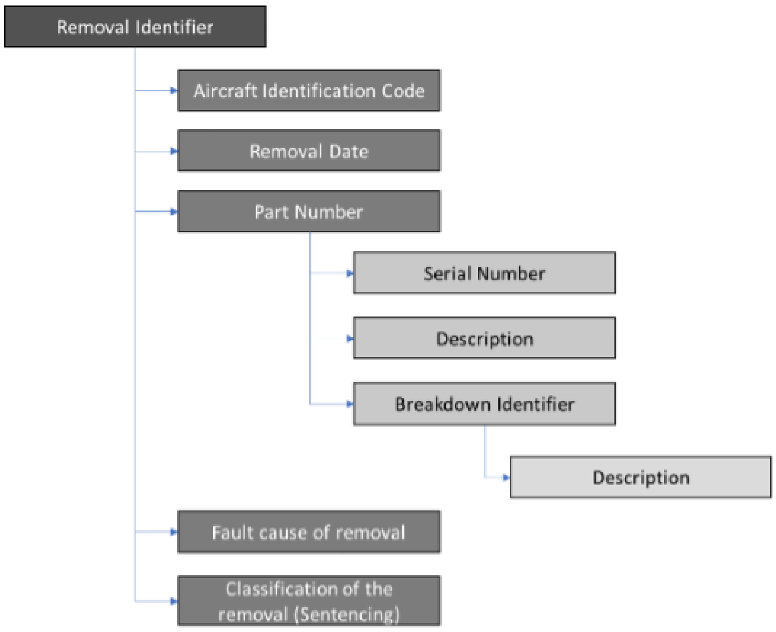


Figure 50: Equipment removal.

The S5000F Use Case that best fits it is the UC50403, Report Maintenance Performed. It is shown in Figure 51

This represents the base from which the tailored model is developed. Once the UML is completed, the XSD is checked in order to generate an XML compliant with the model and with the Specification. This action is performed inside the AeroBridge, where a series of guided steps allows the user to produce the XML message. In the overall architecture of the WP7 AHMS Use Case, the AeroBridge will be the link between the raw data and the Ground Framework, which has the capability to process the data in order to provide digital services.

Table 7: Exchange Capabilities.

No	Data Transmission (TX)	Data Reception (RX)
1.	**Data Extraction**	**Input XML Message**
	A connection with the source system is established to retrieve the necessary raw data. Note: in this stage of development, the raw data are directly provided to the prototype, no connection is established.	The XML message is received and imported.
2.	**XML Generation**	**XML Reading**
	The raw data are processed in accordance with the UML model, the mapping is done with respect to S5000F attributes and the XSD schema, to translate the information and produce a message compliant with the S-SERIES. The steps are: • Generation of XML message • Validation of valid values and XSD types • Full message Validation • Verification of Business Rules (Schematron)	The XML message is read and de-structured. The steps are: • XML Parser • Full message Validation • Validation of valid values and type • Verification of Business Rules (Schematron)
3.	**Output**	**Data Extraction**
	The generated XML message is then transmitted. Note: in this stage of development, the message is not sent but just exported for storage purposes.	A connection with the receiving system is established to provide the necessary raw data. Note: in this stage of development, the data are exported by the prototype, and no connection is established.

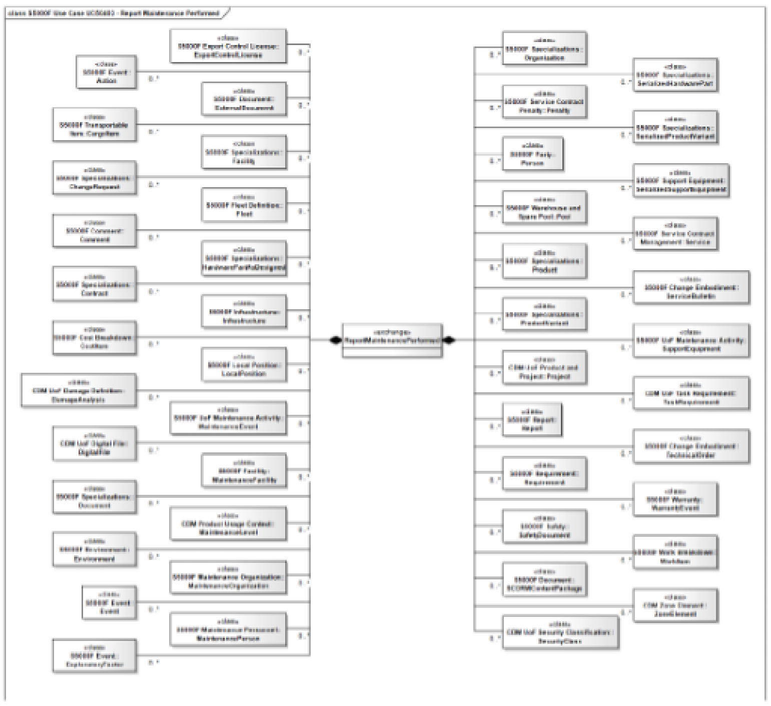


Figure 51: S5000F UC50403.

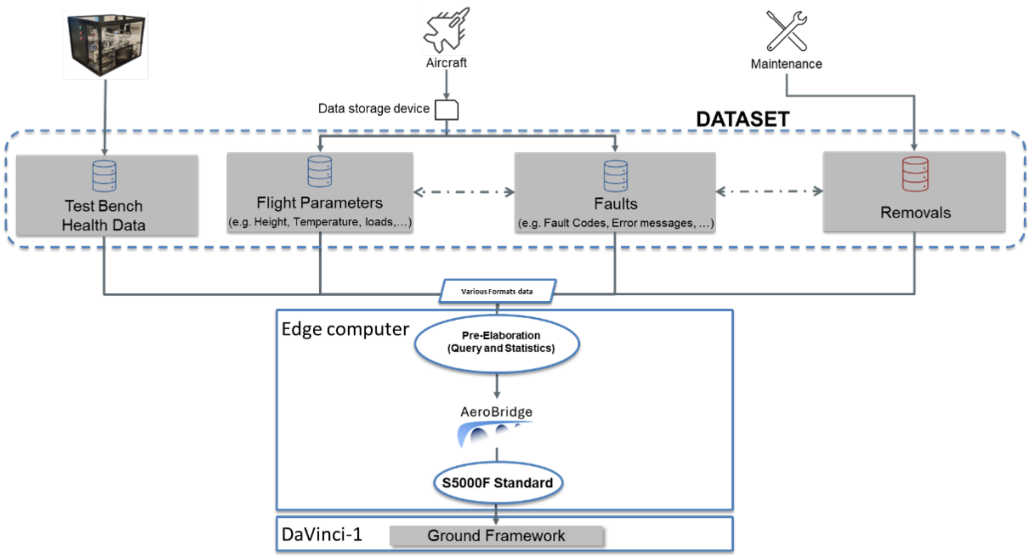


Figure 52: Integration of Aerobridge with WP7 AHMS UC next steps.

5.4 UC 3_9

UC 3_9 is a complex use case that includes various components and systems, each considered individually with the intention of integration in the near future. In this deliverable, several aspects are described: the life cycle assessment and the adapter for serializing process flow diagrams is the subject matter of Section 5.4.1 and 5.4.2, while the recent advancements for systems engineering support are briefly described in Section 5.4.3.

5.4.1 Life Cycle Assessment

The Arrowhead fPVN use case 9.3 on sustainability data exchange involves life cycle assessment (LCA). LCA helps companies better understand and mitigate their environmental footprint. As industries increasingly strive for sustainable operations, LCA offers valuable insights into areas like greenhouse gas emissions, water and air pollution, and waste management. Achieving accurate LCA assessments requires a secure framework for data sharing that respects the confidentiality concerns of all involved parties so that everyone willingly contributes to accurate data.

Methods The proposed platform is designed to enable secure and confidential exchanges of LCA data within and across organizations, facilitated by a combination of data spaces and Secure Multi-Party Computation (SMPC). SMPC allows organizations to conduct necessary computations – including data translation functionality - without exposing underlying data to other entities, maintaining data confidentiality throughout the process. The platform incorporates multiple core components to ensure a secure data-sharing environment: a metadata broker, an SMPC server, a dataspace connector, an Identity Access Manager (IAM), and a Certificate Authority (CA). The metadata broker enables users to locate available data assets across the data space, while the dataspace connector ensures encrypted data exchange. Meanwhile, the IAM and CA support user authentication and maintain secure communication channels. This architecture is structured to allow direct computation through a web application, providing companies with a streamlined, secure way to perform LCA assessments. The encryption of data exchanges guarantees that sensitive information remains confidential, even when multiple parties are involved.

System Architecture At the heart of the system architecture is the dataspace connector, which enables interoperability and secure data transmission. The metadata broker complements this by organizing and facilitating access to data assets, allowing users to efficiently search for and retrieve necessary LCA information. The Certificate Authority (CA) issues digital certificates, establishing a trusted authentication system within the data space that verifies the identities of involved entities. Additionally, the SMPC agent plays a pivotal role by orchestrating LCA calculations securely, ensuring that each participant's data remains confidential throughout the process. This agent, along with the application server, manages user requests and enables interaction with the data space connector and broker. Overall, this architecture facilitates a robust and secure environment for executing LCA while preserving the privacy and sovereignty of each organization's data. Metadata is centrally located to read by anyone while original data is located at the owner's end. Metadata defines a common standard for to exchange of data between organizations. Data space connectors facilitate the transformation of data to the standard data format for the exchange. And they respond to the requests from other organizations.

In this workflow model, the place for applying model-based translations is mostly allocated to the connectors. Based on the study taking into account the maturity of the implementation, the size of the community user, availability of technical support, license, and terms of use, WP4 converges to Eclipse Data Connector (EDC).

Technology stack:

- Data space framework - IDS
- Data space connector (EDC) – Java
- Broker – Java
- JIFF MPC – JavaScript, BGW MPC Protocol-based

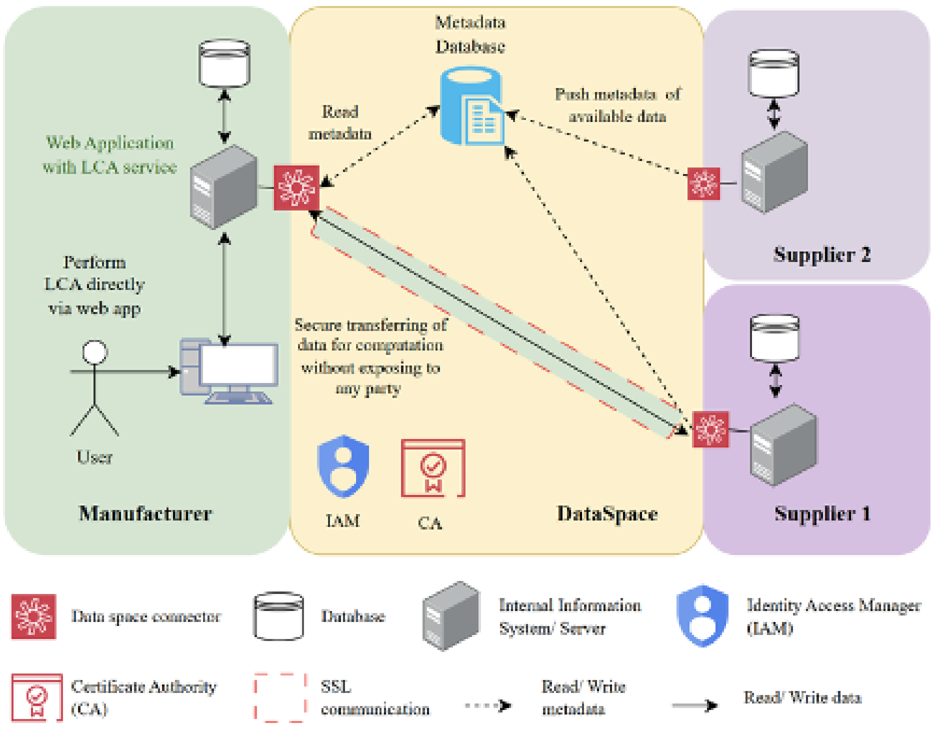

Metadata Database
Read metadata
Push metadata of available data
Web Application with LCA service
Perform LCA directly via web app
User
Manufacturer
Secure transferring of data for computation without exposing to any party
IAM
CA
DataSpace
Supplier 2
Supplier 1
Data space connector
Database
Internal Information System/ Server
Identity Access Manager (IAM)
Certificate Authority (CA)
SSL communication
Read/ Write metadata
Read/ Write data

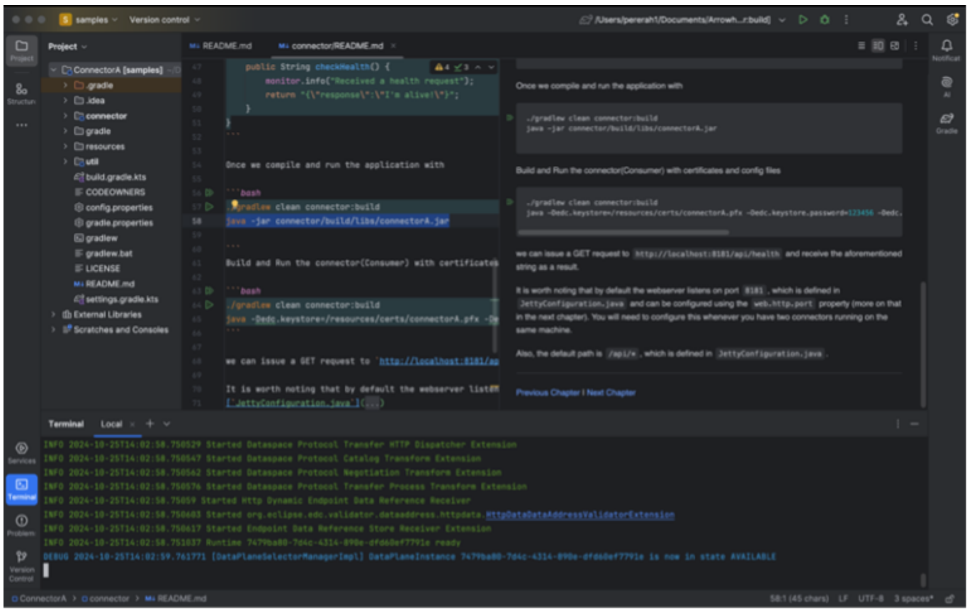
Figure 53: System Architecture Overview.

Challenges and Observations Integrating SMPC within data spaces is challenging due to limited documentation and practical examples of SMPC's application in this context. Other obstacles include inadequate maintenance of critical components in the International Data Spaces Association (IDSA) ecosystem.

- Setting up the IDS Connector. During the setup of the IDS (International Data Spaces) connector, an issue was encountered due to a missing dependency in the Fraunhofer Iais EIS Maven repository. The required dependency was not available in the repository (Fraunhofer Iais EIS repository). This issue was reported.
- Metadata Broker Setup. A similar error occurred when attempting to set up the metadata broker, with the same missing dependency issue. This was documented and raised as an issue for further investigation.
- Alternative Solution. As a workaround, transitioned to using Eclipse EDC (Eclipse Dataspace Connector). During the setup, a bug was identified in Eclipse EDC, which was reported to the development team. The issue was subsequently addressed and resolved. Further details on the bug report and resolution can be found here: Eclipse EDC Bug Report. SMPC – Documentation and examples for SMPC are limited, and current libraries do not support LCA-specific computations. To make SMPC work for LCA, custom logic needs to be developed within these libraries. This will require further exploration of homomorphic encryption methods and protocols like BGW, which the current library under testing, JIFF, is based on. JIFF repository[1].

Preliminary Results Two Eclipse Data Space Connectors have been successfully set up, as shown in Figure 54 and Figure 55. The JIFF library has been modified to support matrix multiplication between two parties. When each party inputs a separate matrix, the resulting matrix is returned after computation. Initially, both parties connect to the MPC server; one party submits its matrix and awaits the other party's submission. Once both matrices are submitted, they are encrypted and shared between the parties. Computation is performed on these encrypted matrices, and only the resulting matrix is decrypted to reveal the output. The implementation involves two client parties, a server, and the MPC logic.

[1] https://github.com/multiparty/jiff



```
    public String checkHealth() {
        monitor.info("Received a health request");
        return "{\"response\":\"I'm alive!\"}";
    }
}
```

Once we compile and run the application with

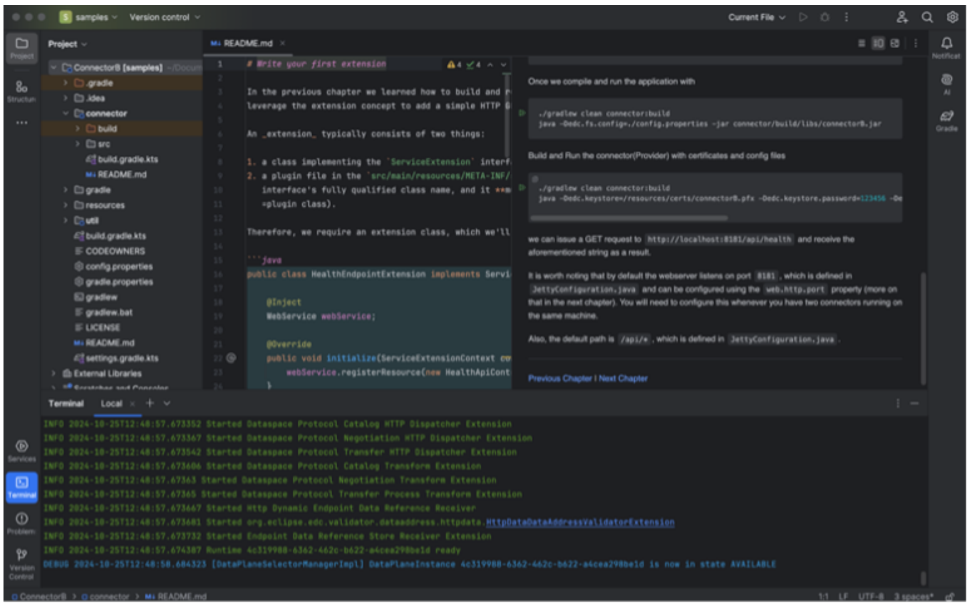
```bash
./gradlew clean connector:build
java -jar connector/build/libs/connectorA.jar
```

Build and Run the connector(Consumer) with certificates

```bash
./gradlew clean connector:build
java -Dedc.keystore=/resources/certs/connectorA.pfx -De
```

we can issue a GET request to `http://localhost:8181/ap

It is worth noting that by default the webserver list
[`JettyConfiguration.java`](...)
```

Once we compile and run the application with

```
./gradlew clean connector:build
java -jar connector/build/libs/connectorA.jar
```

Build and Run the connector(Consumer) with certificates and config files

```
./gradlew clean connector:build
java -Dedc.keystore=/resources/certs/connectorA.pfx -Dedc.keystore.password=123456 -Dedc.
```

we can issue a GET request to `http://localhost:8181/api/health` and receive the aforementioned string as a result.

It is worth noting that by default the webserver listens on port `8181`, which is defined in `JettyConfiguration.java` and can be configured using the `web.http.port` property (more on that in the next chapter). You will need to configure this whenever you have two connectors running on the same machine.

Also, the default path is `/api/*`, which is defined in `JettyConfiguration.java`.

Previous Chapter | Next Chapter

```
INFO 2024-10-25T14:02:58.750529 Started Dataspace Protocol Transfer HTTP Dispatcher Extension
INFO 2024-10-25T14:02:58.750547 Started Dataspace Protocol Catalog Transform Extension
INFO 2024-10-25T14:02:58.750562 Started Dataspace Protocol Negotiation Transform Extension
INFO 2024-10-25T14:02:58.750576 Started Dataspace Protocol Transfer Process Transform Extension
INFO 2024-10-25T14:02:58.75059 Started Http Dynamic Endpoint Data Reference Receiver
INFO 2024-10-25T14:02:58.750603 Started org.eclipse.edc.validator.dataaddress.httpdata.HttpDataDataAddressValidatorExtension
INFO 2024-10-25T14:02:58.750617 Started Endpoint Data Reference Store Receiver Extension
INFO 2024-10-25T14:02:58.751037 Runtime 7479ba80-7d4c-4314-890e-dfd60ef7791e ready
DEBUG 2024-10-25T14:02:59.761771 [DataPlaneSelectorManagerImpl] DataPlaneInstance 7479ba80-7d4c-4314-890e-dfd60ef7791e is now in state AVAILABLE
```

Figure 54: Connector A.

5.4.2 Balas to DEXPI Translator

WP4 is developing an adapter to serialize process flow diagrams (Balas simulation models) into the DEXPI Process standard format [22]. This work relates to UC 3_9: Interoperability for technical information exchange in the process industry. WP4 is performing serialization according to the IDO/PLM RDL ontology [23], as an ontology provides a more expressive data format. Our goal is to integrate the DEXPI process into IDO and extend it with pulp, board, and paper-specific concepts (Requirements R1.6, R1.7, R1.8, and R1.9 [24]).

Balas is a steady-state simulation software developed at VTT, used for modeling chemical processes, particularly in the pulp and paper, food processing, and biochemical industries. With over 40 years of experience in analyzing complex processes, energy, and mass balances, the current version in use is Balas 3.3.

Balas simulation models are constructed using Microsoft Visio, employing symbols representing Balas units and streams that connect these units (Figure 56). The Balas units represent actual process unit operations (i.e., equipment). Each unit features 2 to 16 ports, which can serve as either input or output ports. Streams connect the units, transferring material information, with streams either entering or exiting the unit through the ports. A Balas model can only be simulated after the units are correctly connected with streams, ensuring the flowsheet topology is faultless.

The library contains approximately 220 different Balas units. To model a specific piece of process equipment, such as a filter, multiple Balas units may be required. Each Balas unit can contain a varying number of calculation modules, with Balas offering 118 distinct modules for describing unit operations. Consequently, it is possible to have multiple units utilizing the same calculation module. Each module has its own parameter sets for parameterizing unit operations, which can be either fixed or user-defined. Fixed parameters include temperature, pressure, flow, consistency, dry matter content, solid loss, and split ratio. In contrast, user-defined parameters may involve component-specific retentions, rejections, or reaction yields. Some calculation modules, such as mixers and dividers, do not have parameter sets.

The chemical compounds in the model system are represented in Balas with material components. The Balas component database includes hundreds of chemical compounds and ions, which may be well-known compounds with unique ChEBI numbers or IUPAC identifiers (e.g., water, ethanol), custom Balas-specific compounds without such identifiers (e.g., softwood, cellulose, lignin), or user-defined compounds that are not saved in the Balas component database.

Adapter Architecture for Translations

Figure 57 illustrates the structure of the adapter from the perspective of its export functionality. The exporter manages the serialization process, feeding the IProcessElement with Balas process model information. The ModelWriter implements the IProcessElement and serves as a superclass for converters specific to the standard
```



~~~
# Write your first extension

In the previous chapter we learned how to build and r
leverage the extension concept to add a simple HTTP G

An _extension_ typically consists of two things:

1. a class implementing the `ServiceExtension` interf
2. a plugin file in the `src/main/resources/META-INF/
   interface's fully qualified class name, and it **m
   *plugin class).

Therefore, we require an extension class, which we'll

```java
public class HealthEndpointExtension implements Servi

    @Inject
    WebService webService;

    @Override
    public void initialize(ServiceExtensionContext co
        webService.registerResource(new HealthApiCont
    }
~~~

Once we compile and run the application with

```
./gradlew clean connector:build
java -Dedc.fs.config=./config.properties -jar connector/build/libs/connectorB.jar
```

Build and Run the connector(Provider) with certificates and config files

```
./gradlew clean connector:build
java -Dedc.keystore=/resources/certs/connectorB.pfx -Dedc.keystore.password=123456 -De
```

we can issue a GET request to `http://localhost:8181/api/health` and receive the aforementioned string as a result.

It is worth noting that by default the webserver listens on port `8181`, which is defined in `JettyConfiguration.java` and can be configured using the `web.http.port` property (more on that in the next chapter). You will need to configure this whenever you have two connectors running on the same machine.

Also, the default path is `/api/*`, which is defined in `JettyConfiguration.java`.

Previous Chapter | Next Chapter

```
INFO 2024-10-25T12:48:57.673352 Started Dataspace Protocol Catalog HTTP Dispatcher Extension
INFO 2024-10-25T12:48:57.673367 Started Dataspace Protocol Negotiation HTTP Dispatcher Extension
INFO 2024-10-25T12:48:57.673542 Started Dataspace Protocol Transfer HTTP Dispatcher Extension
INFO 2024-10-25T12:48:57.673606 Started Dataspace Protocol Catalog Transform Extension
INFO 2024-10-25T12:48:57.67363 Started Dataspace Protocol Negotiation Transform Extension
INFO 2024-10-25T12:48:57.67365 Started Dataspace Protocol Transfer Process Transform Extension
INFO 2024-10-25T12:48:57.673667 Started Http Dynamic Endpoint Data Reference Receiver
INFO 2024-10-25T12:48:57.673681 Started org.eclipse.edc.validator.dataaddress.httpdata.HttpDataDataAddressValidatorExtension
INFO 2024-10-25T12:48:57.673732 Started Endpoint Data Reference Store Receiver Extension
INFO 2024-10-25T12:48:57.674387 Runtime 4c319988-6362-462c-b622-a4cea298be1d ready
DEBUG 2024-10-25T12:48:58.684323 [DataPlaneSelectorManagerImpl] DataPlaneInstance 4c319988-6362-462c-b622-a4cea298be1d is now in state AVAILABLE
```

Figure 55: Connector B.

format. The Balas model can be exported according to the DEXPI Process experimental XML schema [25] or the RDF Turtle format [26] related to PLM RDL. Additionally, the ModelWriter can export Balas model elements (e.g., Balas units or streams) in a non-standard JSON format for testing and debugging the adapter's functionality.

Balas exports its model configuration into the adapter according to the following API (application process interface) defined in IProcessElement:

- `add_value(std::string key, std::string value)`: Add a key value to the current element, e.g., port direction for a port.
- `add_element(std::string key, IProcessElement* elem)`: Add an element to the current element, e.g., a port for a stream.
- `append(std::string key, IProcessElement* elem)`: Add an element to a list of elements, e.g., a port for a unit or a unit for a process.
- `write(std::ostream& stream)`: Write the element to an output stream.
- `backendNamespaceInUse(std::string key, std::string namespace)`: Add a backend namespace to be used in the export, e.g., include backend-specific parameters in the export.

WP4 has also defined a REST (Representational State Transfer) API to make different Balas process model parts available in the DEXPI process or PLM RDL format. Currently, the following methods are supported:

- GET /processes/<process_id>: Gets a process (i.e., model).
- GET /processes/<process_id>/process_steps/: Gets all process steps of a process.
- GET /processes/<process_id>/process_steps/<process_step_id>: Gets a specific process step.
- GET /processes/<process_id>/process_connections/: Gets all process connections.
- GET /processes/<process_id>/process_connections/<process_connection_id>: Gets a specific process connection.

The adapter and the REST API are meant to be independent of Balas and, thus, Balas could be replaced with another tool.

The mapping between Balas process model elements and the respective elements of the DEXPI process or PLM RDL is defined in an ontology. For example, Balas calculation module types are defined as subclasses

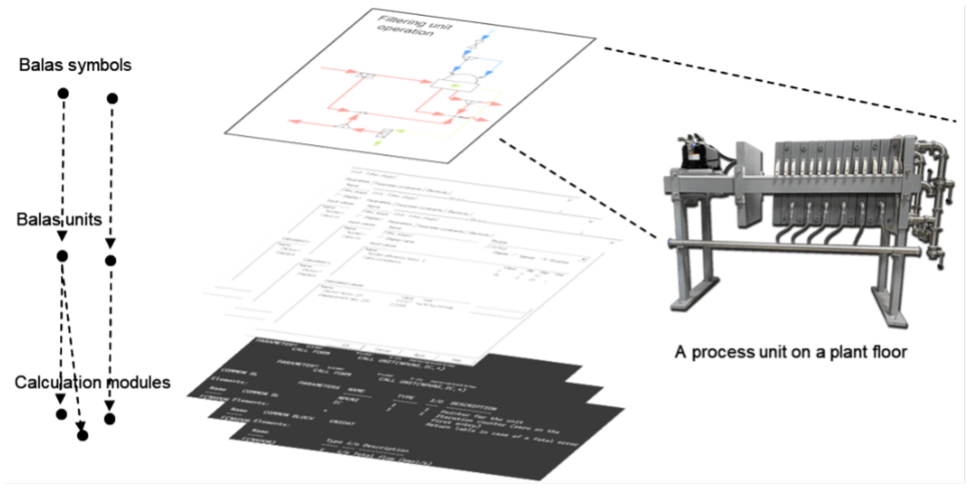


Figure 56: Balas architecture: symbols, units, calculation modules and their relations. A calculation module implements algorithms for solving the mass and energy balances over the unit. Source of the process unit picture: https://www.brewmation.com/brewing/mash-press.

of corresponding DEXPI process and PLM RDL model elements. The ontology can be uploaded into a graph database and the mapped elements can be queried with SPARQL. In the following SPARQL example, WP4 query for the PLM RDL superclass of the Balas calculation module type TSET ('setting exit temperature in a heat exchanger'). The filter statement filters out results that are not PLM RDL superclasses (e.g., each Balas calculation module type is also a subclass of the Balas unit module).

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdl:  <http://rds.posccaesar.org/ontology/plm/rdl/>
PREFIX balas: <http://ww.semanticweb.org/arrowhead/ontologies/balas/classes/>

SELECT ?superClass
WHERE {
    balas:TSET rdfs:subClassOf ?superClass .
    FILTER(isUri(?superClass) && STRSTARTS(STR(?superClass), STR(rdl:)))
}
```

The result of the query is `rdl:PCA_100000005`, which represents the plant process *Enthalpy Change*. The Balas calculation module type `TSET` is mapped directly to *Enthalpy Change* rather than to one of its subclasses (Cooling or Heating), since `TSET` can function both as a heater and a cooler.

Currently, Balas calculation module types have been mapped to corresponding process steps of DEXPI process standard and plant processes of PLM RDL. Therefore, Balas calculation modules are serialized mainly as generic modules without accurate parameter information. In addition, Balas stream information is serialized.

Most of the challenges of the serialization process relate to the mapping between Balas and the standard formats. The main challenge is that many Balas modules can be mapped to several DEXPI process steps or PLM RDL plant processes. Correspondingly, DEXPI process steps or PLM RDL plant processes can be mapped to several Balas modules. Only in a few cases, there is a one-to-one mapping.

Another issue is the mapping of Balas calculation module parameters. For example, the Balas calculation module parameters are quite different from the parameters of the corresponding DEXPI process steps. Our workaround currently has been to define a Balas-specific namespace [27]. Balas calculation module parameters that have not been mapped to DEXPI process step parameters are included as Balas-specific parameters. The usage of the Balas namespace can be controlled by a flag. If the flag is not set non-mapped Balas parameters are not included in the export. Balas supports the development of hierarchical models. A hierarchical model contains many levels; main level, sub-level and sub-level's sub-level which are connected to each other with certain Balas units and streams. Hierarchical models can currently be exported only into DEXPI process format. From the Balas adapter point of view, the main limitations relate to user-defined calculation module parameters that cannot yet be exported.

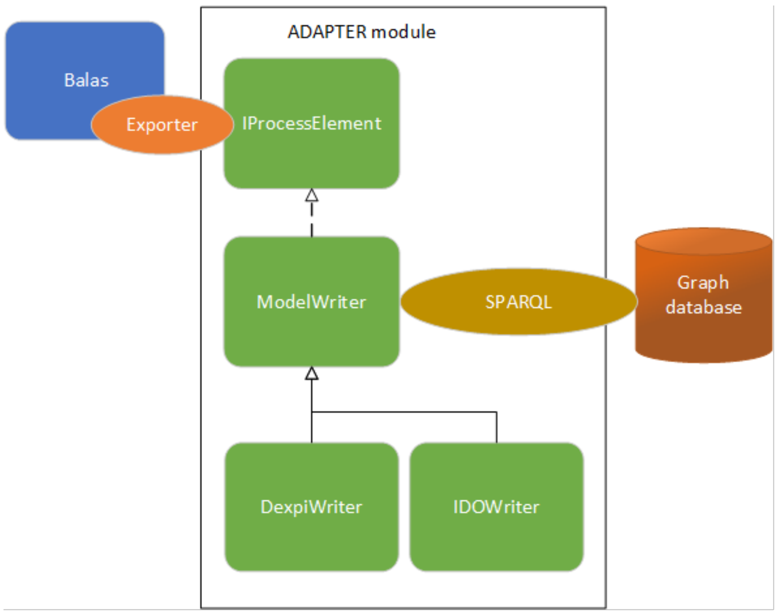


Figure 57: Adapter to serialize Balas simulator models into DEXPI Process and PLM RDL standard formats (the factory method design pattern has been adapted in the adapter).

Models for verification and validation WP4 has developed five different Balas example models to verify and validate the export functionality of the adapter. Each example model contains a varying number of Balas units. Each Balas unit can have a varying number of calculation modules. Each calculation module has its own parameter sets that are either fixed or user-defined. In the example models, the units are either connected with streams or left unconnected. Each example model contains a varying number of material components that represent chemical compounds. They can be either well-known compounds, custom Balas-specific compounds, or user-defined compounds. Below the studied example models with increasing complexity are described.

The simplest Balas example model consists of two Balas units; a pump and a heat exchanger (Figure 58). The example model doesn't contain any pulp, board, or paper industry-specific unit operations. The example model includes as a material component only water which is a well-known chemical compound having a unique ChEBI number/IUPAC identifier and being therefore identifiable. This example model doesn't contain any user-defined parameter sets. Because the units are connected properly with streams, this example model represents a real process that can simulated. There are no recycling streams in this model. This example model supports the testing of the export functionality of the adapter on Balas units with simple and fixed parameter sets and well-known material components.

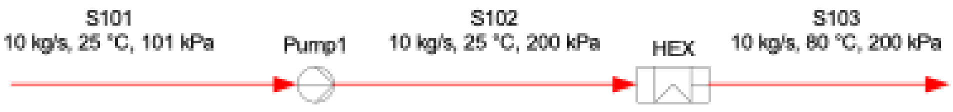


Figure 58: A simple Balas example model with one pump and a heat exchanger to test the export functionality of the adapter on simple Balas units and well-known chemical compounds.

The second Balas example model consists of five Balas units; two pumps, a tank, a heat exchanger, and a mixer (Figure 59). The example model doesn't contain any pulp, board, or paper industry-specific unit operations. Like the previous example model, this example model includes material component water and has no user-defined parameter sets. The units are properly connected with streams making it possible to simulate the model. This example model contains one recycling stream complicating the topology of the model a bit compared to the previous one. This example model supports the testing of recycling streams in exports. It is also used to test the export functionality of the adapter on Balas units without any parameter sets.

The third Balas example model contains a large variety of Balas units, but still not any pulp, board, paper, or power production-specific units (Figure 60). The model includes as the material components water, ethanol, and

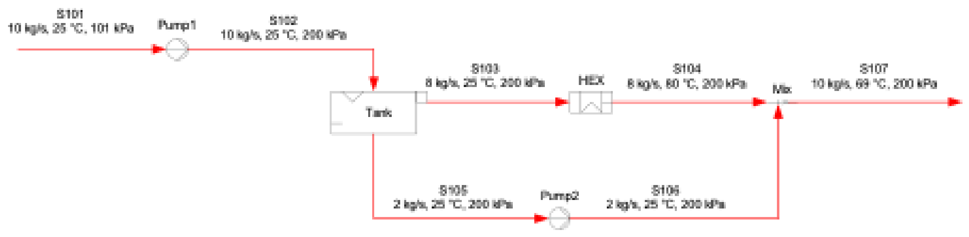


Figure 59: A simple Balas example model with two pumps, a tank, a heat exchanger, a mixer, and one recycling stream to test the export functionality of the adapter on recycling streams and Balas units without any parameter sets.

solid CaCO3 which all are well-defined compounds having their identifiable ChEBI numbers/IUPAC identifiers. The model doesn't contain any user-defined parameter sets. The units are not connected to each other with streams, i.e., the process cannot be simulated. This example model supports the testing of the export functionality of the adapter on Balas units with complicated parameter sets.

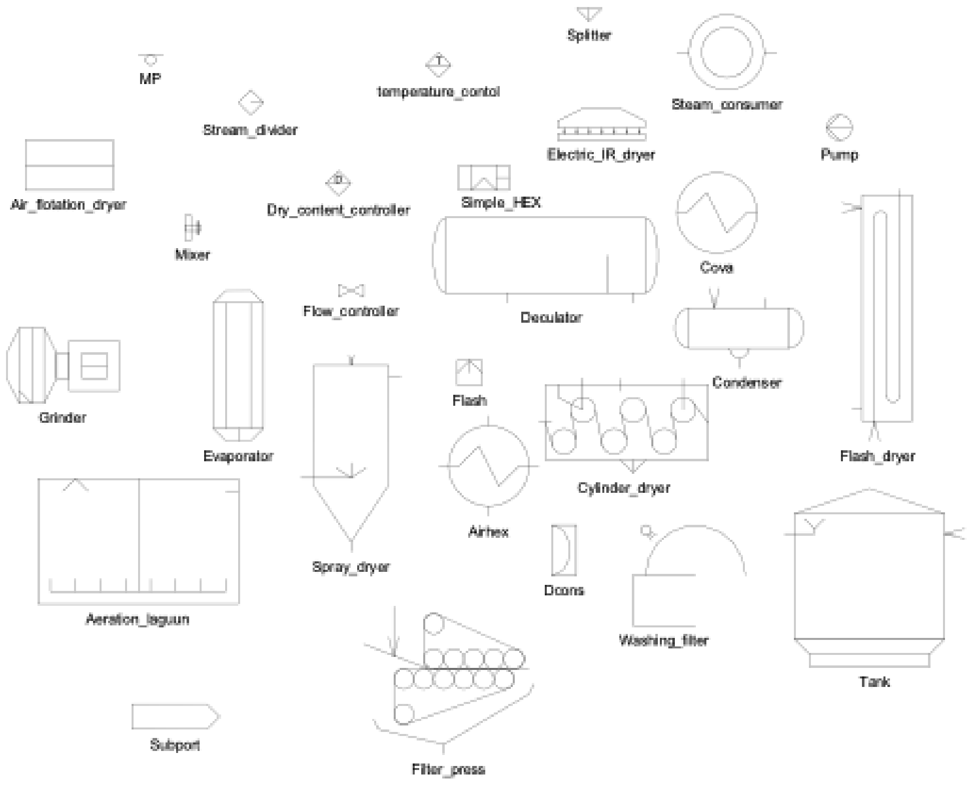


Figure 60: A Balas example model containing several complicated unit operations to test their export functionality.

The fourth Balas example model is a hierarchical model with three levels: main level (Figure 61), sub-level (Figure 62) and sub-level's sub-level (Figure 63). The model contains only simple units like valves, mixers, pumps, and splitters. The material component in the process is water. The model doesn't contain any user-defined parameter sets. The units are properly connected with streams making it possible to simulate the model. This example model supports the testing of the export functionality of the adapter on hierarchical process models.

Status of adapter implementation

WP4 used the adapter to attempt to serialize the example models in DEXPI process or PLM RDL formats. When exporting the example models, the main challenge relates to mapping Balas calculation module parameters to DEXPI process or PLM RDL. The adapter can quite well export the two simplest example models into both standard formats. However, not all relevant Balas calculation module parameters have been mapped to DEXPI process. Regarding serializing the other example models the following issues (in addition to parameter

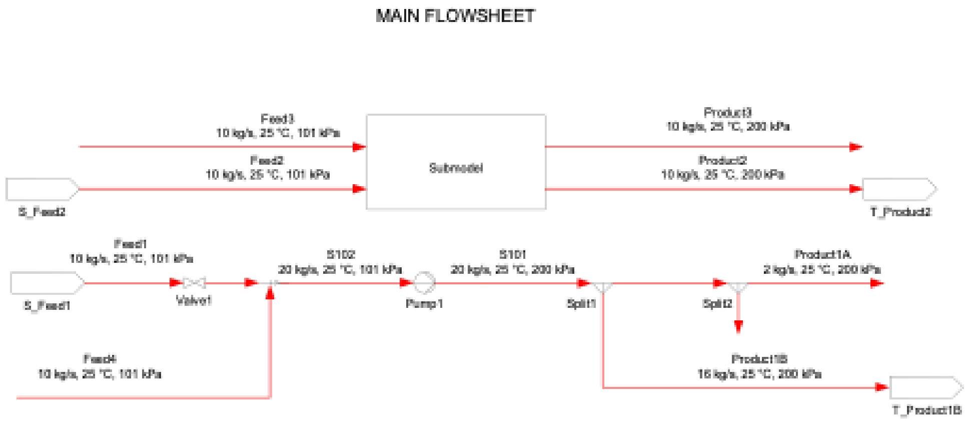


Figure 61: An example of a hierarchical Balas model; main level of the hierarchical model.

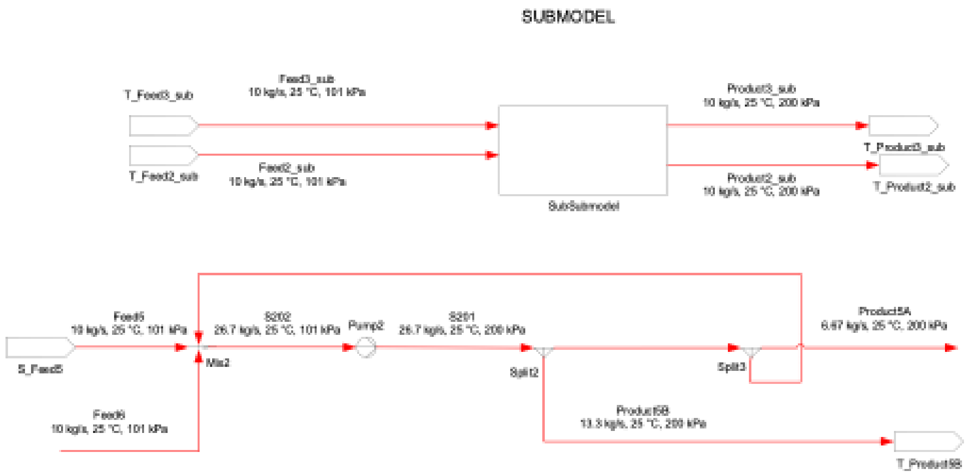


Figure 62: An example of a hierarchical Balas model; sub-level of the hierarchical model.

mapping issues) were identified:

- User-defined parameters cannot be serialized.
- In each model, ca. 5 Balas calculation modules were mapped only to the most generic process step.
- Some Balas calculation modules are mapped to several different process steps in DEXPI process.
- Hierarchical models cannot yet be serialized into PLM RDL format.
- All material components are serialized as PureMaterialComponents without ChEBI or IUPAC identifiers in DEXPI process. (Neither ChEBI nor IUPAC identifiers have been defined for Balas material components).

While evaluating standard data formats (objectives O1.2 and O1.5), a concrete case study is that in addition to exporting Balas models in standard format, models in standard format should also be imported unambiguously back to the Balas simulator. However, this can be quite challenging because, e.g., between all DEXPI process steps and Balas modules there is currently no unambiguous mapping.

5.4.3 Towards SysML-based model governance for DEXPI

In parallel to the activities detailed above, WP4 has also initiated an investigation of SysML in relation to DEXPI architectural modeling tasks.

The particular novelty of our approach lies in a parallel consideration of SysML v1.6 and SysML v2. The intention is to rely on the existing Arrowhead SysML v1.6 profile for basic architectural documentation while

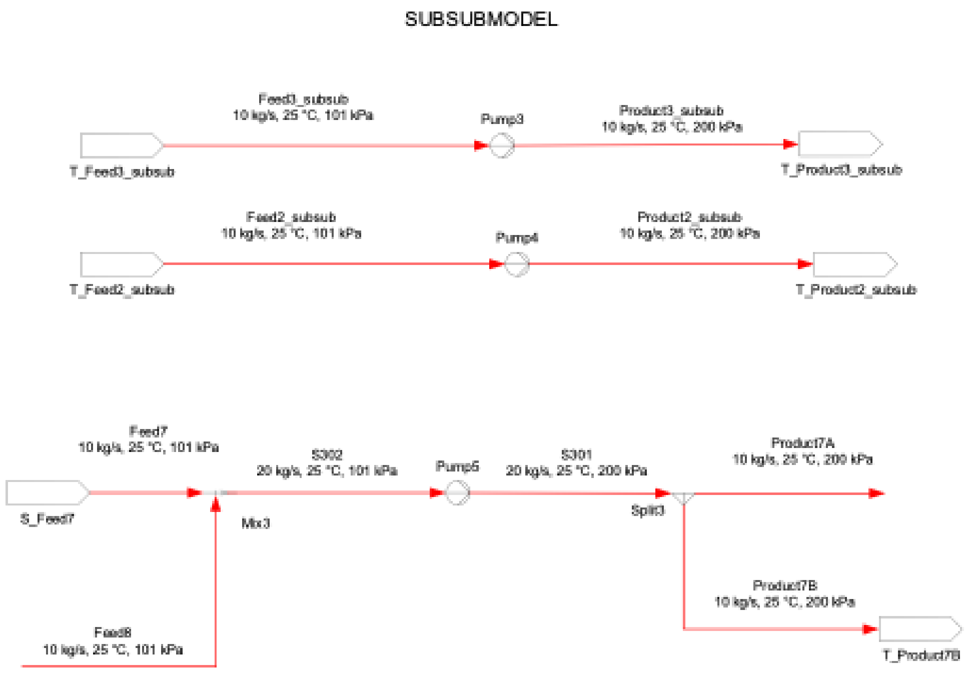


Figure 63: An example of a hierarchical Balas model;sub level's sub-level of the hierarchical model.

creating a pendant of it in SysML v2, in order to establish the upcoming language for integration and validation purposes.

As a pilot example, WP4 uses a basic DEXPI example, coming from the respective professional community, on which further details can be found in a blog entry at DEXPI's website.[2].

WP4 is in the process of establishing a parallel, SysML-based modeling of this sample use-case, to be used as the baseline for further translation and integration activities.

[2]https://dexpi.org/dexpi-modelling-of-the-process-design-behind-block-flow-diagrams-and-process-flow-diagrams/

6 Validation and Verification Processes

This section describes a preliminary proposal of high-level processes conceived to verify and validate (V&V) the three types of translators. The main objective is to identify a set of concrete steps to verify and validate the translators' functionalities and the results obtained from their adoption in the project use cases. As illustrated in the previous sections of this deliverable, the internal of the three types of translators are profoundly different technology-wise, but their functionalities present several commonalities that WP4 tried to reflect defining common phases in the three different processes.

This preliminary proposal will be consolidated in the next project semester, and for each phase, WP4 will define the set of concrete actions, methodologies, and tools required to operate the V&V processes.

6.1 V&V Process for the Ontology-based Translators

Figure 64 illustrates the preliminary V&V process WP4 has defined for the ontology-based translator. It is composed of 8 phases, from the V&V of the translator's requirements to the documentation of the entire V&V process.

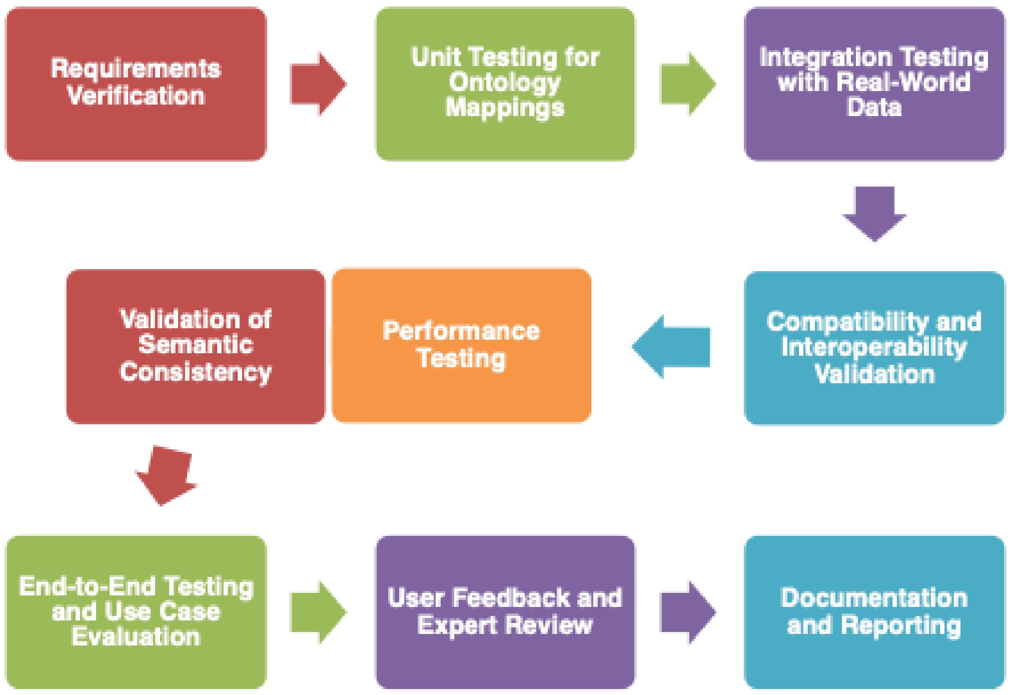

Requirements Verification
Unit Testing for Ontology Mappings
Integration Testing with Real-World Data
Compatibility and Interoperability Validation
Performance Testing
Validation of Semantic Consistency
End-to-End Testing and Use Case Evaluation
User Feedback and Expert Review
Documentation and Reporting

Figure 64: V&V process for the ontology-based translators.

The role, objectives, and actions of the 8 phases are described in more detail in the following:

1. Requirements verification. The objective is this initial phase is to ensure that the translator aligns with functional and technical requirements, particularly around interoperability between data models. This phase consists of conducting a review of the specified requirements against the implemented functionality, focusing on core features such as semantic alignment, data structure compatibility, and annotation support.
2. Unit testing for ontology mapping. In the second step, unit testing is used to verify that individual ontology mappings and translations work as expected. This phase consists of creating unit tests for each mapping and translation rule to ensure accuracy. Test cases should cover a range of scenarios, including common edge cases, mismatched data types, etc.
3. Integration testing with real-world data. This step is intended to confirm that the translator performs as expected in a simulated environment corresponding to a real scenario and/or use case. This step consists of using synthetic data, or data collected in a real scenario and/or use case, to validate the translation

process in a controlled (lab) environment. Different data formats (e.g., JSON, XML) and types are tested, as well as the accuracy, and consistency, of the translation and the handling of data annotations.

4. Compatibility and interoperability validation. This phase is intended to ensure that translated data is compatible with the consuming system's data model. In this step, WP4 need to test the translated outputs with various consuming systems (simulated or real) to verify interoperability. Evaluate performance, focusing on data completeness, semantic accuracy, and quality of the result, trying to understand whether the consumer system can use the data as they are translated or if further adjustments of the translation are required.

5. Performance testing AND Validation of Semantic Consistency. The next two steps can be carried on in parallel. If the performances of the translator don't scale it is not reasonable to continue the evaluation but, at the same time, it is also crucial to validate the semantic consistency of the translation before proceeding with further V&V steps.

   (a) Performance testing. This phase consists of assessing the efficiency and scalability of the translator, measuring the translator's response time, memory usage, CPU load, etc. with different data volumes. The objective is to verify that the translator performs within acceptable thresholds under expected operational loads.

   (b) Validation of semantic consistency. This phase is needed to confirm that the translator preserves the meaning and intent of the data during translation. It consists of conducting tests with varied semantic content to confirm that translations reflect the intended meaning, using ontology alignment tools and semantic reasoning engines as appropriate.

6. End-to-end testing and use case evaluation. Validate the translator in end-to-end scenarios across the project use cases. The validation consists of implementing end-to-end tests that simulate actual use cases, from input data ingestion to final translation output, ensuring the correct integration of the translator in the Arrowhead ecosystem.

7. User feedback and expert review. At this stage of the evaluation, it is possible to refine the translator based on user feedback and/or expert insights. It consists of collecting feedback from users and experts sharing the results of the translation collected in the previous phases of the V&V process to identify potential improvements in usability, reliability, performance, and translation accuracy. Incorporate this feedback into iterative testing and refinement.

8. Documentation and Reporting. The last phase of the process consists of ensuring clear documentation and transparency in test results. It includes the documentation of the V&V process, including test cases, results, issues, and how they have been resolved, and generates reports to support traceability and future audits.

6.2 V&V Process for the AI-based Translators

The following diagram, Figure 65 illustrates the preliminary V&V process WP4 have defined for the AI-based translator. It is composed of 7 phases, from the V&V of the preparation and quality of the data used for AI training to the final step evaluating the need for further iterations of the entire process.

The role, objectives, and actions of the 7 phases are described in more detail as follows:

1. Data preparation and quality assessment. AI algorithms largely rely on the data adopted for their training and this phase ensures high-quality input data for model training, testing, and validation. It consists in preparing diverse, representative datasets, applying data cleaning, preprocessing (e.g., handling inconsistencies, filling missing values), and augmentation if necessary. It includes also statistical analyses to verify that datasets capture necessary variations in data models for robust training and evaluation.

2. Model training and fine-tuning. This phase focuses on training the AI translator effectively on relevant data model languages, ensuring semantic accuracy. It is based on the use of large language models (LLMs), neural machine translation (NMT), or specific AI frameworks tailored for translation. The objective is to fine-tune the model on domain-specific data to enhance semantic understanding and optimize it

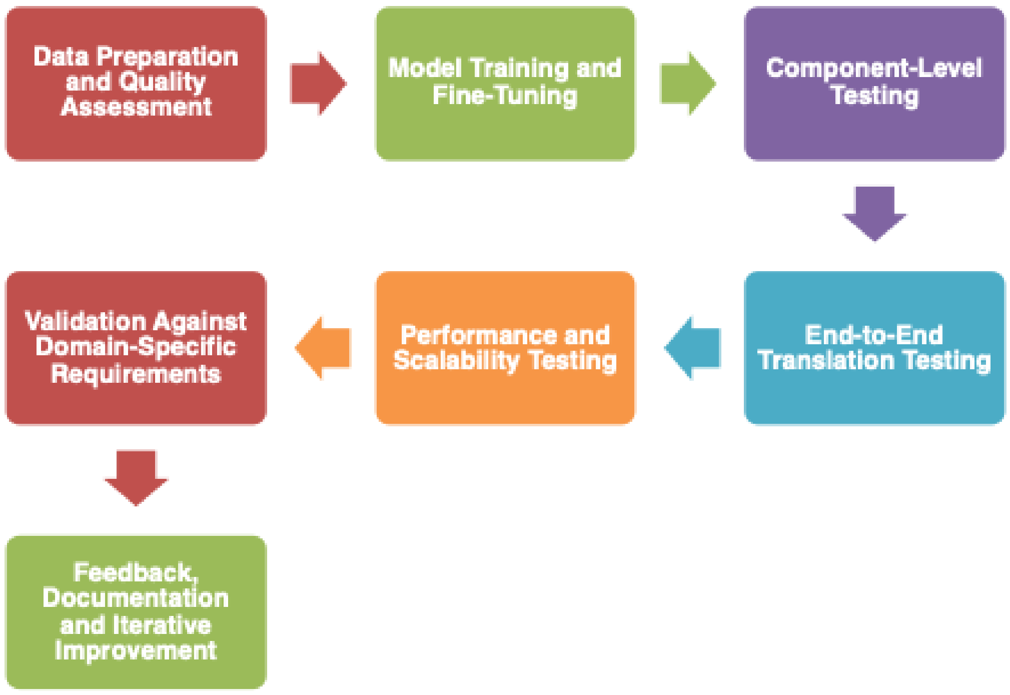

Data Preparation and Quality Assessment
Model Training and Fine-Tuning
Component-Level Testing
End-to-End Translation Testing
Performance and Scalability Testing
Validation Against Domain-Specific Requirements
Feedback, Documentation and Iterative Improvement

Figure 65: V&V process for the AI-based translators.

for specific translation tasks. The training activities could be iterative to achieve high performance with reduced error rates.

3. Component-level testing. The component level testing consists of validating each component of the AI-based translator to ensure functionality and accuracy. In this phase WP4 conducts unit tests on core components, such as context recognition, text clustering, and label extraction, to ensure accurate recognition of data structures and semantic relationships. The phase includes also the evaluation of each module's output against expected values before integrating them into the overall translator.

4. End-to-end translation testing. This phase aims at ensuring the accuracy and robustness of the entire AI translation workflow. Real-world scenarios inspired by the use cases are used to evaluate the AI translator's performance, covering different data model formats (e.g., XML, JSON). The accuracy of the translation is assessed by comparing translated outputs with expert-reviewed and correct translations. Metrics such as BLEU scores, ROUGE, and human evaluation can help gauge performance.

5. Performance and scalability testing. This phase evaluates the translator's efficiency and reliability under different load conditions. It allows us to measure the AI translator's response times, memory usage, CPU load, and throughput. The objective is to evaluate how well the translator scales when processing large volumes of data and to assess if latency remains within acceptable limits. The phase includes also the adjustment of model configurations as necessary to enhance performance.

6. Validation against domain-specific requirements. This phase ensures that the translator meets specific needs related to the targeted domain. It is intended to validate that the AI translation maintains semantic integrity when handling specialized domain terminologies. It is necessary to involve and collaborate with domain experts to review samples of the translated output, confirming that critical meanings are preserved and that translations meet domain standards.

7. Feedback, documentation, and iterative improvement. The process concludes with continuous improvement of model accuracy and performance based on real-world feedback. The V&V process is fully documented, and WP4 collects user feedback on translation quality, accuracy, and ease of use. This phase includes also the refinement of the model by addressing common errors and retraining on identified

weaknesses, but also incorporating new datasets or domain-specific data if needed to keep the model up-to-date and robust.

6.3 V&V Process for the Model-based Translators

Figure 66 illustrates the preliminary V&V process WP4 has defined for the model-based translator. It is composed of 8 phases, from the requirements V&V to the documentation of the entire V&V process and of its results.

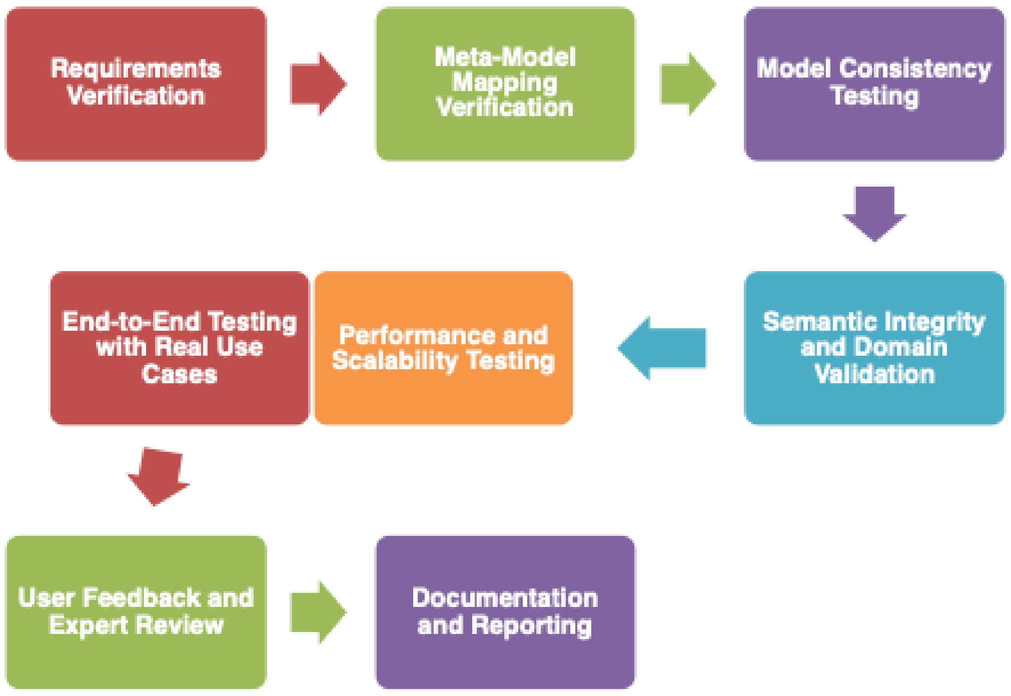

Requirements Verification
Meta-Model Mapping Verification
Model Consistency Testing
Semantic Integrity and Domain Validation
Performance and Scalability Testing
End-to-End Testing with Real Use Cases
User Feedback and Expert Review
Documentation and Reporting

Figure 66: V&V process for the model-based translators.

The role, objectives, and actions of the 8 phases are described in more detail as follows:

1. Requirement verification. The first phase is intended to ensure alignment with all functional, technical, and domain-specific requirements of the model-based translator. It consists of reviewing requirements to confirm they clearly define the expected model mappings, data transformation rules, and output formats. In this phase, it is important to ensure that each requirement has traceability to specific validation tests for comprehensive coverage.

2. Meta-model mapping verification. The second step aims at confirming that the model accurately maps elements between source and target data models. For each meta-model, WP4 verifies the mapping rules and relationships by creating unit tests for key attributes and elements in the mapping schema. Finally, WP4 cross-checks these mappings against domain experts' reviews to confirm semantic consistency.

3. Model consistency testing. The translation must ensure that each model transformation is consistent and adheres to specified semantic rules. It implies performing consistency checks using for example automated model verification tools to validate structure, relationships, and constraints across transformed models. The objective is to confirm that all dependencies, constraints, and inheritance rules are maintained accurately during translation.

4. Semantic integrity and domain validation. This phase consists of verifying that translated models preserve semantic meaning across domains, especially for specialized terminologies. It implies conducting validation tests for domain-specific elements to confirm that translations uphold semantic accuracy and engaging with domain experts to validate critical transformations and align them with domain standards.

5. Performance, scalability, and end-to-end testing. The performance and scalability test can be carried out on single parts of the translator and on the complete translator, but must always cover also the end-to-end test in real use cases, to concretely prove the capabilities of the translator.
   (a) Performance and scalability testing. This phase ensures that the translator operates efficiently under various load conditions, processing both large and complex models within acceptable timeframes. The objective is to test the translator's scalability with increasing model size and complexity and monitor the response times and resource usage to verify that the system meets performance thresholds.
   (b) End-to-end testing with real use cases. Confirm that the translator performs correctly in actual scenarios inspired by the project use cases and, finally, in the use cases themselves. The phase consists of maintaining compatibility with downstream applications. It includes the simulation of real-world translation scenarios by using representative input data models and validating the output against expected results. These scenarios should cover typical use cases, project use cases, and edge cases that mirror actual model transformations in deployment contexts.
6. User feedback and expert review. This phase incorporates iterative feedback from users and domain experts to refine and enhance the translator's accuracy and usability. It consists in collecting feedback from end users and experts on translation quality, accuracy, and ease of integration. Using this feedback it is possible to adjust mapping rules and improve model transformation accuracy.
7. Documentation and reporting. This final step aims to ensure comprehensive documentation of the V&V process and test results. It consists of documenting each stage of verification and validation, detailing test cases, outcomes, and any necessary adjustments. It could include the preparation of a summary report for stakeholders that includes insights into performance, areas for improvement, and compliance with defined requirements.

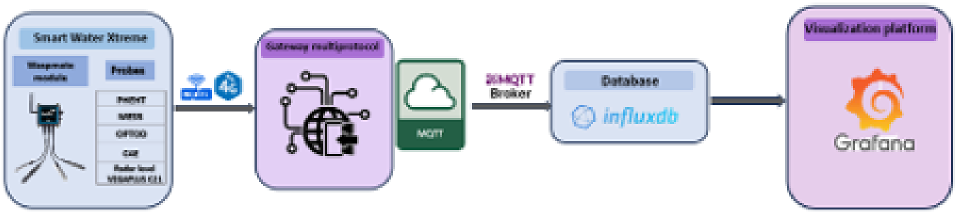


Figure 67: IoT Architecture of the environmental monitoring system.

7 Collaboration With Other WPs

WP4 maintained regular collaboration with WP3 and the use case providers of WP6, WP7, and WP9. These weekly meetings serve as platforms for discussing and analyzing the data model structures utilized in the use cases, as well as strategizing how translation solutions can be tailored to meet the needs of these use cases.

In addition, future collaboration between WP4 and WP2 is anticipated to enhance the integration of translation solutions within the Arrowhead framework and facilitate the provisioning of microservices.

The team has continued to focus on comprehensively modeling solutions down to the integration and interface level, incorporating industry de facto standards such as OpenAPI and AsyncAPI. Together with the work being carried out in WP2, this will enable solutions to automatically identify the data being sent for service by retrieving API specifications from deployed services within an Arrowhead environment. This information can then be used to later deploy translations autonomously.

Furthermore, Task 4.3 has started a collaboration with WP2 to advance the development of the Arrowhead profile [28] This collaboration aims to seamlessly integrate the efforts of Arrowhead-Copilot, spearheaded by WP2, into the Papyrus environment. A possible collaboration between Task 4.1 and Task 4.3 on the use of the semantic variable concept to define the requests in the Arrowhead framework is under analysis. In fact, the Papyrus4Manufactoring [29] a variant of Papyrus is integrated with a similar concept semanticID inspired by the AAS standard [30] to identify referable elements.

7.1 Dataset Provision

Some UC and other WP have contributed to the WP4 work by providing datasets. One of the examples is the datasets provided by BEIA. BEIA has selected and implemented monitoring for a comprehensive set of environmental quality parameters, which are crucial for assessing the environmental impact and ensuring compliance with relevant standards.

The parameters provide an overview of water quality, ensuring that the monitoring system can detect and report changes in water conditions accurately and promptly. The data visualization on Grafana allows for real-time monitoring and analysis, enhancing the project's capability to respond to environmental changes efficiently.

The design of sensor deployment strategies in the project was a part of BEIA's work to ensure optimal data collection and accuracy. The monitoring stations were strategically installed on the off-shore oil and gas platform. The deployment timeframe for these installations spanned from autumn 2023 to summer 2024. This period allowed for thorough planning, installation, and initial testing to ensure the sensors operated correctly under the specific environmental conditions of the Black Sea. The choice of the platform provided a stable and secure location for the sensors, minimizing potential disruptions and ensuring continuous data collection. By positioning the sensors in this offshore location, BEIA ensured that the environmental monitoring system could capture a wide range of data, reflecting the true conditions of the marine environment. This strategic deployment supports the project's goals of comprehensive environmental monitoring and contributes to the overall success and reliability. The data was collected using Libelium sensors. It was then transferred via a Libelium gateway to the database used within the project for visualization and processing. The Libelium solution implemented in the project consists of the sensing module (Plug&Sense) and the Meshlium device that serves as an IoT gateway. The Plug&Sense device is connected to the Internet or another (private) network via 4G and sends the data to the Meshlium. Once the data arrived in Meshlium, it was stored in the InfluxDB database. Then, the collected data was transferred to the Grafana visualization platform via the MQTT broker, see Figure67.

To ensure the accuracy and reliability of the environmental monitoring system, the sensors were rigorously tested and calibrated under a variety of environmental conditions that mirrored the operational challenges expected at the offshore installations. Before deployment, each sensor underwent a thorough calibration process in controlled laboratory conditions. Calibration ensured that the sensors met the required precision standards for measuring parameters such as water quality (e.g., pH, turbidity, dissolved oxygen), air quality (e.g., CO2, NO2), and weather conditions (e.g., wind speed, temperature). This process aligned sensor output with standardized reference values, ensuring accurate measurements when transferred to the data processing platform.

8 Appendixes

- **IEC_IPC2581C_AI**
- **PDFPreprocessing_AITIA**

9 Conclusions

The document presents a comprehensive report on the development of second-generation translation solutions, detailing progress made through Month 18. This deliverable includes an analysis of how objectives have been met, prototypes and proposals for advanced translation solutions, and supporting activities that further the state of the art and facilitate integration within designated use cases. The report organizes translation advancements into three primary categories: Ontology-based translation, AI-based translation, and Model-based translation.

Additionally, the document outlines the planned structure and workflow for validation and verification in quality assurance, featuring examples of collaborative work with other work packages. Collaboration, particularly with the UC, has been identified as essential to the project's success, with WP4 placing significant emphasis on this synergy.

The report specifically addresses WP4's objectives:

- Exploring the potential and feasibility of a super ontology approach.
- Investigating the capabilities and feasibility of an ML/AI-based approach.
- Assessing the potential of a model-based approach.

Furthermore, it highlights early progress toward:

- Strengthening UC collaboration to ensure translation solutions meet UC requirements.
- Establishing validation and verification processes for translation quality assessment.

In summary, WP4's contributions to the AfPVN project are progressing well, effectively aligning with the project's goals and strategic roadmap.

10 References

[1] J. Kopeckỳ, 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.

[2] 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.

[3] 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.

[4] D. L. McGuinness, F. Van Harmelen *et al.*, “Owl web ontology language overview,” *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[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] W3C, “SPARQL 1.1 Query Language, W3C Recommendation,” 2013, accessed: 2024-10-15. [Online]. Available: https://www.w3.org/TR/sparql11-query/

[8] P. Pinheiro, M. Bax, H. Santos, S. M. Rashid, Z. Liang, Y. Liu, J. P. McCusker, and D. L. McGuinness, “Annotating Diverse Scientific Data with HAScO,” in *Proceedings of the Seminar on Ontology Research in Brazil 2018 (ONTOBRAS 2018). São Paulo, SP, Brazil*, 2018.

[9] M. Dumontier *et al.*, “The Semanticscience Integrated Ontology (SIO) for biomedical research and knowledge discovery,” *Journal of Biomedical Semantics*, vol. 5, p. 14, 2014.

[10] T. Lebo, S. Sahoo, and D. L. McGuinness, “PROV-O: The PROV Ontology,” W3C, W3C Recommendation, 2013. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-o-20130430/

[11] P. Fox, D. L. McGuinness, L. Cinquini, P. West, J. Garcia, J. L. Benedict, and D. Middleton, “Ontology-supported scientific data frameworks: The virtual solar-terrestrial observatory experience,” *Computers & Geosciences*, vol. 35, no. 4, pp. 724–738, 2009.

[12] Apache Software Foundation, “Apache Jena,” https://jena.apache.org, 2021.

[13] P. Fox, D. L. McGuinness, L. Cinquini, P. West, J. Garcia, J. L. Benedict, and D. Middleton, “Ontology-supported scientific data frameworks: The Virtual Solar-Terrestrial Observatory experience,” *Computers & Geosciences*, vol. 35, no. 4, pp. 724–738, Apr. 2009.

[14] Aitia Industrial Internet of Things, “Rag tutorial,” 2023, accessed: 2024-11-05. [Online]. Available: https://github.com/Aitia-IIOT/ah-ai-translation-poc/tree/master/RAGTutorial

[15] LangChain, “Langchain,” 2024, accessed: 2024-11-05. [Online]. Available: https://www.langchain.com/

[16] ——, “Introduction to langgraph - setup,” 2024, accessed: 2024-11-05. [Online]. Available: https://langchain-ai.github.io/langgraph/tutorials/introduction/#setup

[17] LangChain Academy, “Intro to langgraph - lesson 1: Motivation,” 2024, accessed: 2024-11-05. [Online]. Available: https://academy.langchain.com/courses/take/intro-to-langgraph/lessons/57712261-lesson-1-motivation

[18] LangChain AI, “Langgraph studio,” 2024, accessed: 2024-11-05. [Online]. Available: https://github.com/langchain-ai/langgraph-studio

[19] T. Tothfalusi, Z. Csiszar, and P. Varga, "TACO – A copilot with Generative AI Actions for Telecommunication Core Network Analysis," in *IEEE/IFIP Network Operations and Management Symposium*, Hawaii, USA, 2025, (under review).

[20] SMAP2024, "Towards Seamless Data Translation Based on Data Models: A Hybrid AI Framework for Smart Transportation and Manufacturing," 2024, accessed: 2024-11-05. [Online]. Available: https://smap2024.athenarc.gr

[21] SDF2024, "A Hybrid AI Framework Integrating Ontology Learning, Knowledge Graphs, and Large Language Models for Improved Data Model Translation in Smart Manufacturing and Transportation," 2024, accessed: 2024-11-05. [Online]. Available: https://www.fkie.fraunhofer.de/en/events/sdf2024.html

[22] DEXPI Initiative, "DEXPI Process Information Model Version 1.0," 2024.

[23] POSC Caesar Association, "Reference Data Library: Industrial data ontology & Product Lifecycle Management," 2024, accessed: 2024-09-25. [Online]. Available: https://rds.posccaesar.org/doc/

[24] Arrowhead fPVN, "Deliverable D9.2: Process industry use case first year progress and next step specification," May 2024.

[25] D. Cameron, W. Otten, H. Temmen, and G. Tolksdorf, "DEXPI Process Specification, Release 1.0," 2023.

[26] "RDF 1.1 Turtle - Terse RDF Triple Language," 2014, accessed: 2024-04-04. [Online]. Available: https://www.w3.org/TR/2014/REC-turtle-20140225/

[27] W3C, "Namespaces in XML 1.1 (Second Edition), W3C Recommendation," 2006, accessed: 2024-10-17. [Online]. Available: https://www.w3.org/TR/xml-names11/

[28] [Online]. Available: https://github.com/asmasmaoui/profile-library-sysml

[29] [Online]. Available: https://youtu.be/p4gTzzc95hw?si=OhCZp061WdM0cTLg

[30] "Details of the asset administration shell - part1, version 3.0rc02," Tech. Rep., 11 2020. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.pdf?__blob=publicationFile&v=5

11 Revision History

11.1 Contributing and reviewing partners

Contributions	Reviews	Participants	Representing Partner
X	X	Cristina Paniagua	LTU
	X	Carl Borngrund	LTU
X	X	Filipe Moutinho	UNINOVA
X		Asma Smaoui	CEA
X		Paolo Azzoni	ETH
X		Tamas Tothfalusi	AITIA
X		David Rutqvist	SIN
X		Géza Kulcsár	IQL
X		Vahid Tavakkoli	KLU
X		Jaakko Niemisto	SEM
X		Miren Illarramendi	SEA
X		Teemu Mätäsniemi	VTT
X		Kim Björkman	VTT
X		Lotta Sorsamäki	VTT
X		Luis Lino Ferreira	ISEP
X		Valeriy Vyatkin	AALT
X		David Hästbacka	TAU
X		João Rosas	UNINOVA
X		Alberto Calvo	LEO
X		George Suciu	BEIA

11.2 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2024-10-21	0.1	Creation of the draft and structure of the document	Cristina Paniagua
2	2024-10-21	0.2	Introduction and Objectives Sections	Cristina Paniagua
3	2024-10-30	0.3	Darft of generation of translation solutions	WP4 partners
4	2024-10-31	0.4	Collaboration with other WP Section	Cristina Paniagua
5	2024-11-05	0.5	Refinement and update of the text	Cristina Paniagua
6	2024-11-08	0.6	Final update of the text	WP4 partners
7	2024-11-14	0.7	Validation and verification section	Paolo Azzoni
8	2024-11-18	0.8	Review and correction	Filipe Moutinho, Carl Borngrund
9	2024-11-19	1.0	Final version	Cristina Paniagua

11.3 Quality Assurance

No.	Date	Version	Approved by
1	2024-11-20	1.0	Jerker Delsing