

Deliverable D 4.1

First Generation of Translation Technology

Work package leaders:

Cristina Paniagua
cristina.paniagua@ltu.se

Filipe Moutinho
fcm@fct.unl.pt

Abstract

This document constitutes deliverable D4.1 of the Arrowhead fPVN project. This document outlines proposals for the initial version of translation technology, built to meet the specific needs detailed in the provided use cases.

Grant agreement no.	101111977
Project acronym	Arrowhead fPVN
Project full title	Arrowhead flexible Production Value Network
Dissemination level	PU
Date of Delivery	x
Deliverable Number	4.1
Deliverable Name	First 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, Modelling
Abstract	This document constitutes deliverable D4.1 of the Arrowhead fPVN project. This document outlines proposals for the initial version of translation technology, built to meet the specific needs detailed in the provided use cases.

Contents

1	Introduction	5
2	Overview of WP4 Aims and Tasks	5
2.1	The main purpose of Task 4.1	5
2.2	The main purpose of Task 4.2	5
2.3	The main purpose of Task 4.3	5
2.4	The main purpose of Task 4.4	6
3	WP4 Objectives	6
3.1	Major Project Objectives	6
3.2	WP Objectives	7
4	1st Generation Translation Technologies	8
4.1	Ontology–Based Translation	8
4.2	AI–Based Translation	18
4.3	Model–Based Translation	25
5	Datasets Provision and Other Supporting Activities	38
5.1	Dataset categories and sources	38
5.2	Dataset selection and utilization process	40
5.3	Use-case support activities	42
6	Collaboration With Other WPs	46
7	Appendixes	47
8	Conclusions	48
9	References	49
10	Revision History	51
10.1	Contributing and reviewing partners	51
10.2	Amendments	51
10.3	Quality Assurance	51

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 year of the project is described in this comprehensive report, D4.1, which outlines the first 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 reflects the work initiated by Task 4.4 to gather datasets from industrial partners and support the use cases. Finally, Section 6 describes the collaboration with other WPs to achieve the common project objectives. At the end of the document, a list of appendices has been included.

2 Overview 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.

2.1 The main purpose of Task 4.1

Task 4.1 will explore the utilization of ontologies as an intermediary bridge between significant data modeling languages, aiming to resolve property mismatches inherent in these languages. This task will investigate the capabilities of ontology translation tools, focusing on facilitating microservices translations between major data modeling languages relevant to workflow management and execution. The solutions aim to autonomously address non-matching properties. Actions include identifying and studying relevant ontologies, contributing expertise in workflow languages to enable seamless translation, and investigating machine learning techniques to tackle mapping challenges between different data modeling languages.

2.2 The main purpose of Task 4.2

Task 4.2 will explore Machine Learning/Artificial Intelligence (ML/AI) approaches and algorithms to assess their effectiveness in translating between major data modeling languages. By testing translation accuracy using real-world data from diverse use cases, the aim is to determine the feasibility of these approaches across different language structures. Furthermore, the task seeks to develop translation microservices based on existing architecture and libraries from WP2, aiming to validate the practical viability of these methods. In this endeavor, various approaches are employed, including dictionary learning and algorithms tailored for translation, as well as ML solutions such as Natural Language Processing (NLP) techniques. The emphasis lies on analyzing the feasibility, precision, and applicability of these methods, ultimately aiming to facilitate seamless data model translation across various languages in practical scenarios.

2.3 The main purpose of Task 4.3

Task 4.3 will investigate model-based approaches, such as Unified Modeling Language (UML) and System Modeling Language (SysML), to assess their capability and feasibility in facilitating translation between major

data modeling languages. Translation testing will encompass the utilization of data from various use cases to determine the feasibility of the employed data modeling languages. Additionally, the task aims to develop translation microservices based on architecture and libraries from WP2, focusing on verifying feasibility in selected use cases. Actions include exploring UML metamodels to model major data modeling languages, identifying similarities, and addressing them through formal semantic reasoning or learning methods. Furthermore, technology evaluation based on industrial production requirements, provision of aerospace domain knowledge and requirements for model-based translation, and evaluation of usability through use cases will be conducted. Investigation into model-based translation's inclusion in frameworks, guidelines for its application, and collaboration for transformation between different data models will be pursued. Lastly, an analysis of model-based translation in safety-critical domains, cooperation for guidelines in aerospace use cases, and exploration of its usage in requirements-based test generation will be undertaken.

2.4 The main purpose of Task 4.4

Task 4.4 will support the other tasks with the provision of appropriate datasets for development, training, and early validation of translation quality. Certainly, here's the extended version: Task 4.4 will collaborate closely with the use cases and other work packages, playing a fundamental role not only at the beginning but throughout the entirety of the project lifecycle. This collaboration ensures the procurement of comprehensive datasets essential for the development of the project's translation. Furthermore, as the project progresses, Task 4.4 remains instrumental in facilitating the rigorous testing and evaluation of translation outcomes. By collaborating with use cases and leveraging insights from other work packages, Task 4.4 ensures that the translation models align closely with the project objectives and meet the specified 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 initial generation of translators and details the efforts made during the first year 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 that 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. It includes an examination of translation solutions such as DITAG, SEMVAR.CC, broker translator, and proposed work within the Industrial Data Ontology.
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 encompasses the analysis of the translation scenario and the initial application of AI techniques, particularly those related to deep learning.
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, meta-modeling, metadata database translation, P&ID translation, and the transformation between SysML v1.6 and v2.
Provision of data set for translation development and early assessment of translation quality.	This document includes in Section 5 the dataset provision process and support to the Use Case (UC) translation. The focus of the work has been on initiating dialogues with the use cases and other industrial partners to secure access to the necessary datasheets and devise a strategy to abstract the data structure and essential information required for translation without compromising industrial confidentiality.
In cooperation with WP3, to provide translation microservices based on the above approaches	This document includes the first actions taken in collaboration with WP3 to provide translation microservices. Section 6 describes the efforts and results from this initial collaboration.

4 1st Generation Translation Technologies

For effective communication, systems must possess compatible interfaces and a shared understanding of data. This encompasses factors like communication protocols, encoding, encryption, compression, message structure, payload key values, and semantics. Once data is received, comprehension is essential for proper utilization. However, mismatches in interfaces or data compatibility can hinder communication. Translation techniques can resolve such disparities, serving as temporary or permanent solutions. One scenario where translation is vital is during system failures, where an alternate system may lack compatible interfaces, necessitating translation until the original system is restored. Additionally, integrating systems from different vendors or versions may require translation to minimize engineering time. Similarly, updates to legacy systems may disrupt compatibility with others, necessitating translation mechanisms.

In the realm of autonomous translation technologies, significant challenges exist. Deep semantic understanding across diverse data modeling languages is paramount, requiring advanced techniques and semantic reasoning. Moreover, handling multimodal data poses a substantial challenge. Ensuring interoperability of translated models across various systems is crucial, often necessitating standardized data exchange formats and well-defined APIs.

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 first 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 first 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 the first year, Task 4.1 has focused on the development of a tool prototype for data interoperability verification and automatic generation of translators. This tool, named Data Interoperability Translators Automatic

Generator (DITAG), utilizes an ontology-based approach. DITAG receives metadata with semantic annotations, verifies semantic and data compatibility, and automatically generates translators when compatibility is confirmed.

DITAG is a Java application that translates XML and/or JSON data between application systems (*Providers* and *Consumers*). 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]. SAWSDL is an extension to the XSD standard that allows for semantic annotations of elements. 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 greatly disturbing the original structure of the XSD. Figure 1 illustrates the use of both SAWSDL and A3ST to add more context to XSD elements. To support JSON data, DITAG uses JSON Schemas with semantic annotations, as proposed in [6]. Figure 2 presents the equivalent information found in Figure 1, formatted using JSON Schema.

```

1 <?xml version="1.0"?>
2 <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">
3 <xs:element name="dataValues" >
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element type="xs:string" name="units" sawSDL:modelReference="/TemperatureSensor{1;2;3;4}/hasTempUnits/TemperatureUnits{1}"
  a3st:mdi-ref="1;2"/>
7 <xs:element type="xs:float" name="sensor1temp" sawSDL:modelReference="/TemperatureSensor{1}/hasDecValue" />
8 <xs:element type="xs:float" name="sensor2temp" minOccurs="0" maxOccurs="1" sawSDL:modelReference="/TemperatureSensor{2}/
  hasDecValue"/>
9 </xs:sequence>
10 </xs:complexType>
11 </xs:element>
12 <xs:annotation>
13 <xs:appinfo>
14 <a3st:map-data-ind a3st:mdi-id="1" a3st:individual="Fahrenheit">Fah</a3st:map-data-ind>
15 <a3st:map-data-ind a3st:mdi-id="2" a3st:individual="Celsius">Cel</a3st:map-data-ind>
16 </xs:appinfo>
17 </xs:annotation>
18 </xs:schema>
19

```

Figure 1: 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. Here, these annotations specify the accepted types of units for the `units` element

```

1 {
2   "$schema": "http://json-schema.org/draft-07/schema", "type": "object",
3   "a3st": { "type": "object", "properties": { "a3st:map-data-ind": { "type": "array",
4     "items": { "type": "object", "properties": [
5       { "a3st:mdi-id": "1", "a3st:individual": "Fahrenheit", "a3st:node-value": "Fah" },
6       { "a3st:mdi-id": "2", "a3st:individual": "Celsius", "a3st:node-value": "Cel" } ] } } }
7 },
8   "properties": { "product": { "type": "object", "properties": { "layer": { "type": "object", "properties": {
9     "dataValues": { "type": "object", "required": [ "units", "sensor1temp" ],
10    "properties": {
11      "units": { "type": "string", "modelReference": "/TemperatureSensor{1;2}/hasTempUnits/TemperatureUnits{1}", "a3st:mdi-
12      ref": "1;2" },
13      "sensor1temp": { "type": "number", "modelReference": "/TemperatureSensor{1}/hasDecValue" },
14      "sensor2temp": { "type": "number", "modelReference": "/TemperatureSensor{2}/hasDecValue" },
15      "sensor3temp": { "type": "number" },
16      "sensor4temp": { "type": "number" } } } } } } } }
17 }
18 }

```

Figure 2: JSON Schema Example. The information contained in this file is equivalent to that presented in Figure 1.

DITAG architecture and general workflow are presented in Figure 3. It operates through four distinct stages:

- Initially, data from the *Consumer* and *Provider* Schema files are gathered and organized into a JDOM2

Document, from which is extracted a dictionary of annotated elements. Using the XJC API, two Java classes are also dynamically generated, one for each input.

- Next, the data undergoes compatibility testing. All elements from the *Consumer* and *Provider* are compared and matched based on data type and semantic annotation. To facilitate semantic comparison, DITAG uses the Pellet Semantic Reasoner [7], a software tool designed for reasoning tasks over ontologies represented in OWL. Ontologies are utilized to represent knowledge in a machine-readable format, particularly in domains where complex relationships and semantics need to be captured. Reasoning over OWL ontologies involves inferring new knowledge from existing data based on logical rules and axioms defined in the ontology.
- Once preliminary compatibility is achieved, we proceed to pair the *Consumer* and *Provider* elements based on the number of matches requested by the *Consumer*. Pairing only occurs if all required matches of the *Consumer* can be provided by the *Provider*.
- After analyzing the pairings made, DITAG verifies whether the *Provider* and *Consumer* support translation between them. If so, DITAG generates a formal translator as a JAVA file that includes the *Consumer* and *Provider* Java classes derived earlier. This translator enables immediate translation between *Consumer* and *Provider*, supporting *Provider* data in XML or JSON formats. Additionally, an HTML report is generated, including all elements collected from both inputs, all testing performed and their results, the achieved pairings, and an assessment of transaction possibility.

DITAG was tested in case study 7.1 with promising initial results. DITAG was able to automatically generate a translator that converted data files from an electronic design tool into files, with a different data model, of a PLM tool.

4.1.2 Integration of the DITAG tool into Arrowhead clouds (Proposal)

Integrating DITAG into the Arrowhead framework will allow service consumers to translate the messages resulting from the provision of services into messages whose structure and semantics consumers can understand. The adapted architecture with DITAG and additional interactions is schematized in Figure 4.

As illustrated in Figure 4, the original architecture of the Arrowhead framework is adapted to integrate a new module, namely DITAG. It should be noted that this integration may, in the future, be done in two alternative ways: (1) Integration of DITAG as an Arrowhead Core System; (2) Implementation of DITAG as a service provider running within the Arrowhead cloud.

In the first approach, DITAG would be available as a component within the Arrowhead framework and could be used by the other Arrowhead core systems, providers, and consumers. The advantage of this approach is that it is built into the core of the framework, potentially accessible in a more agile way.

In the second approach, DITAG could exist as a translation service provider. The advantage of this approach is that it will enable the creation of more than one DITAG service provider. This way, beyond allowing load balancing, each provider can specialize in more specific document translations, depending on the application scenarios.

With the integration of DITAG, scenarios characterized by consumers who require a specific service but whose result needs to be translated go through the following phases:

- 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 to select suitable providers among the ones received from orchestration. The request carries both the consumer schema annotation and the schemas of each provider.

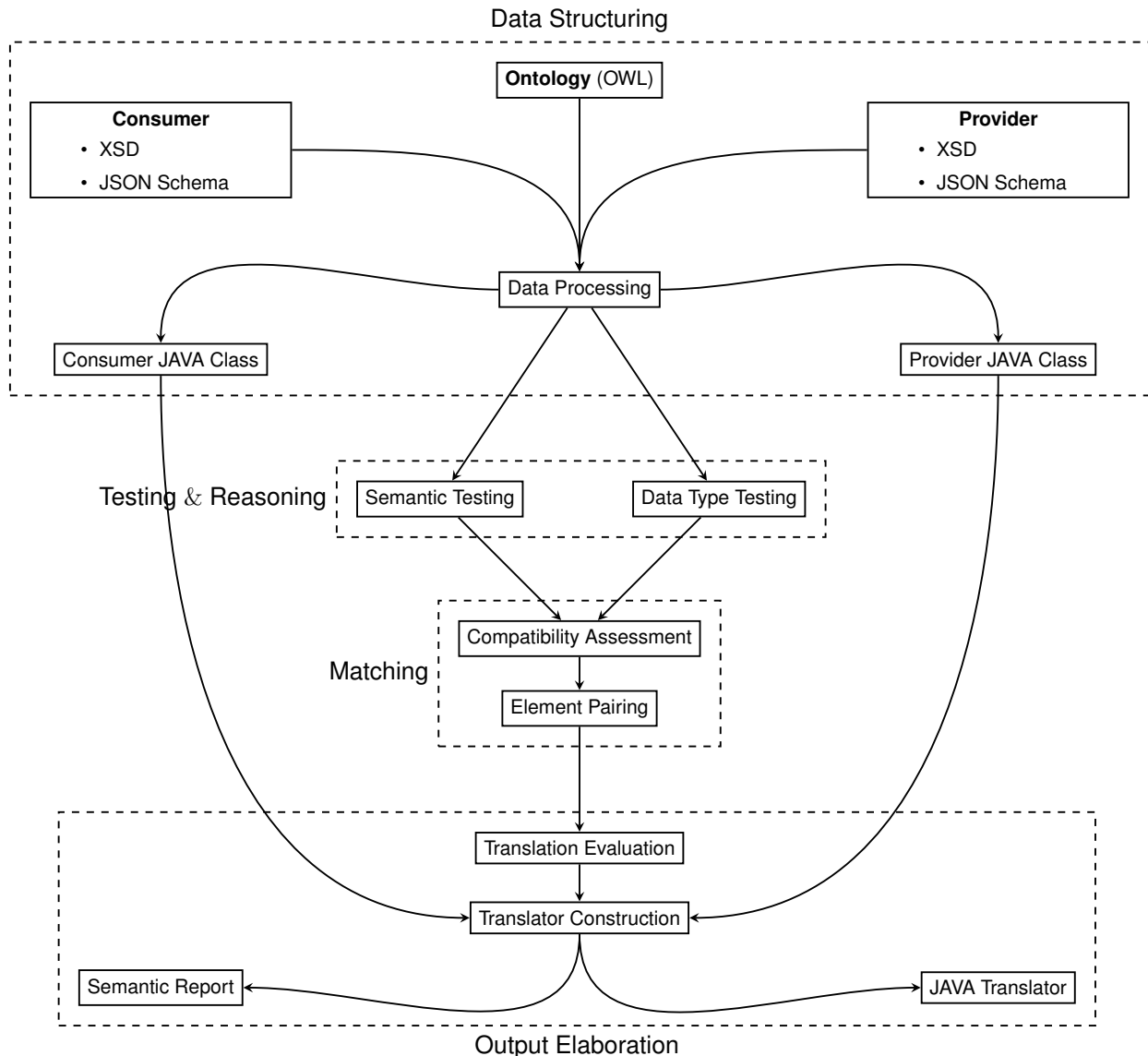


Figure 3: General DITAG Architecture/Workflow.

4. DITAG replies with compatible providers and corresponding translators.

• 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.

During the translation phase, in step 3, the consumer accesses the metadata that characterizes the service, which the service provider specified during step 1. The metadata contains the schema of the service, which, together with the schema of the consumer, allows DITAG to provide a translator that can translate the original document/message into a structure that the consumer can interpret and use.

The translation stage can be enhanced with the integration of "SEMVAR.cc" as described below.

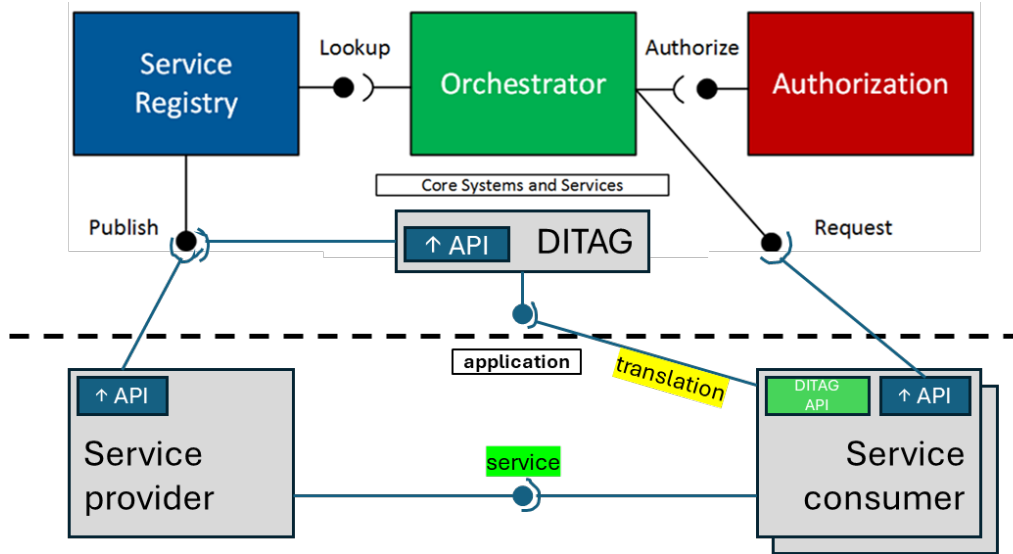


Figure 4: Integration of the DITAG tool in the Arrowhead cloud (figure adapted from <https://arrowhead.eu/technology/architecture/>)

4.1.3 Integration scenario with SEMVAR.CC, DITAG, and the Arrowhead Framework (Proposal)

Baseline Scenario

As initial work, Task 4.1 outlined a data interoperability approach and associated assumptions, which will be later aligned with use cases (UC) in the project. This initial description is the *baseline scenario* illustrated in Figure 5.

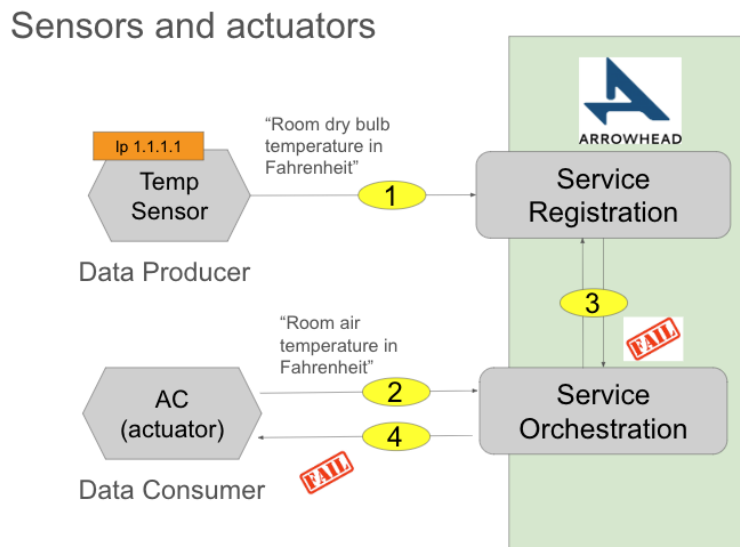


Figure 5: Scenario without interoperability support.

The baseline scenario begins with a specific set of information being registered into the Arrowhead framework: 'a given data producer generates data of type "room dry bulb temperature in Fahrenheit" that is streaming at IP address 1.1.1.1. This registration is carried out by the data producer itself, which calls the Service Registry to store this information (*Action 1*).

Once one or more data producers have registered the types of data they produce and the IP addresses

where the data can be accessed, data consumers can use the Arrowhead Orchestration Service to request producers capable of streaming the required data types. For example, an air conditioning system might send a request to the framework asking for the IP address of a data producer capable of supplying "room air temperature in Fahrenheit" (*Action 2*). This request is received by the Orchestration, which checks the registry and identifies producers capable of fulfilling the data requests (*Action 3*).

The Orchestration Service is expected to perform service compositions that will expand the range of data types supported by the framework. However, for the purposes of simplicity in the technical solutions within this report, we are assuming that the Orchestration Service is currently restricted to lookup services.

Even if the Orchestration Service is not capable of composing services, many are the potential semantic differences between available data types and requested data types, and that **any semantic difference** is currently resulting in a failed request, as shown in Figure 5. For instance, in the example in the baseline scenario, requested and registered data are "room temperature", but one is "room air temperature" while another is "room dry bulb temperature." From a machine point of view, it has no additional knowledge that allows the machine to quantify how different are these two types of data, i.e., these two kinds of "room temperature". Even the fact that we, humans, may know that both variables are "room temperatures" is not something that a machine is capable of inferring without additional knowledge. This results in the framework returning a fail message to the data consumer (*Action 4*).

Scenario with Ontology-based Data Interoperability Support

Figure 6 presents a data integration scenario based on knowledge coming from community-built ontologies. In this scenario, the Arrowhead framework has been integrated with both DITAG and SEMVAR.CC infrastructures.

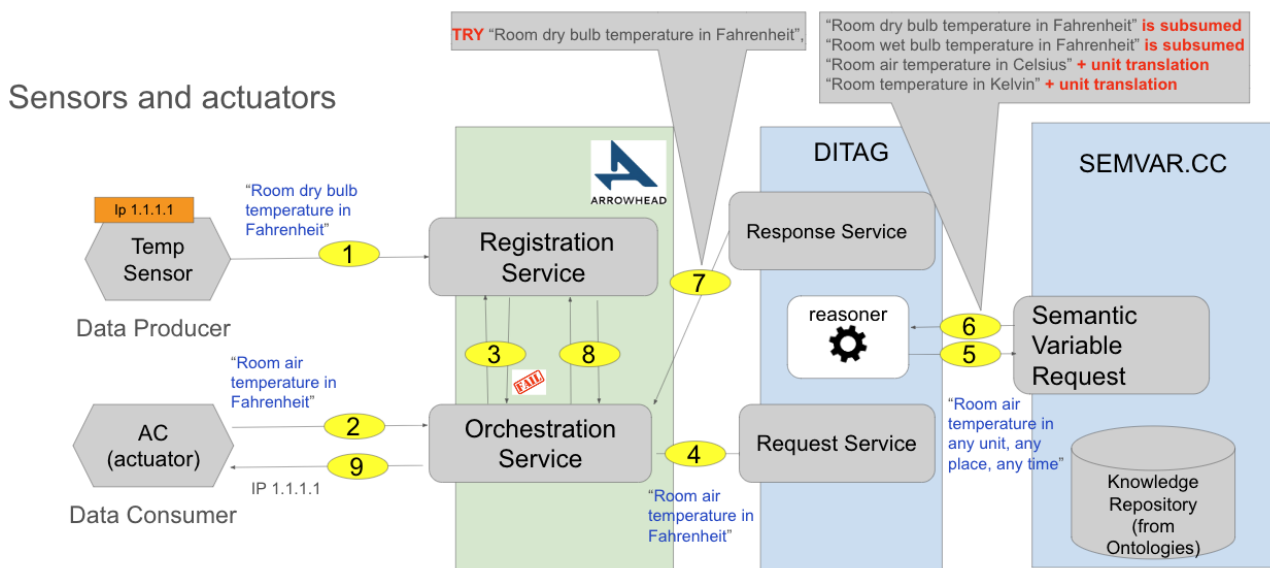


Figure 6: Scenario with DITAG and SEMVAR.CC support for data interoperability.

These are key technical characteristics of such a scenario:

- Before failing a Consumer's request when performing *Action 3*, Arrowhead calls DITAG to see if DITAG can provide a possible translation for the request (this is the new *Action 4* in the data interoperability scenario).
- DITAG may have its own knowledge base based on past requests that may be used to answer Arrowhead's request. But if DITAG has no knowledge about the current request, DITAG may call SEMVAR.CC's service will provide DITAG with a list of possible translations of current requests (*Action 5*).
- SEMVAR.CC uses a broad range of harmonization strategies to feed DITAG with the knowledge that is relevant for DITAG to answer the framework request (*Action 6*):
 - Equivalence rules;

- Class subsumptions rules [8];
 - Restriction relaxations [9].
- Harmonization strategies are executed against knowledge from both SEMVAR.CC's internal knowledge graph and current request. New ontologies are added to SEMVAR.CC knowledge graph when terms used to specify semantic variables are coming from ontologies that are not registered within SEMVAR.CC. In this case, SEMVAR.CC will import the newly informed ontology into its knowledge graph and pre-process the ontology's terms to identify alignments with terms and ontologies already included inside the SEMVAR.CC knowledge base. In addition to importing new ontologies from new requests, SEMVAR.CC is also constantly learning about semantic variables by extracting them from semantic data dictionaries [10] used in repositories of evidence data.
 - DITAG then analyze SEMVAR.CC's translations, select first those that provide one-time translation solutions and provide those back into Arrowhead (*Action 7*).
 - If provided translations do not match Arrowhead's existing registered services, Arrowhead can then call back DITAG looking for other translations, in which case DITAG can feed additional translations back into Arrowhead as services translations instead of one-time translations (this may translate into a sequence of *Actions 7 and 8*).
 - Finally, there is a higher chance of providing a response to the consumer with the use of data interoperability support (i.e., DITAG and SEMVAR.CC) (*Action 9*) than without such support.

Underlying Technology - Software Infrastructure SEMVAR.CC back-end is derived from the open-source Human-Aware Data Acquisition Framework (HADatAc) [11]. HADatAc is a schema-free, evolutionary, scalable, and provenance-aware infrastructure for managing data and metadata content from multiple scientific studies. Three key goals of HADatAc are: (1) to extract relevant data value from instrument-generated files and to move these values into queryable content repositories, (2) to extract relevant metadata from scientist-generated documents and to move these values into queryable content repositories, and (3) to semantically annotate these values in a way that the entire content is logically linked and harmonized (i.e., unified representation) according to evolving collections of well-established scientific ontologies. Semantic variables inside of SEMVAR.CC is also supported by HASCO API, and derived from the general concept of Semantic Data Dictionaries [10].

Underlying Technology - Community-Built Ontologies SEMVAR.CC back-end is based on HASCO API, which is based on the HASCO ontology [12]. HASCO has been developed to encode knowledge about scientific studies, study types, data elicitation from humans, and data simulation from computer models. SEMVAR.CC's core ontologies are fully integrated, aligned, and used in multiple scientific domains, and in addition to HASCO, they include the following core ontologies: W3C's Provenance Ontology (PROV), encoding provenance knowledge, Virtual Solar-Terrestrial Observatory (VSTO) [13], encoding knowledge about instruments and platforms, Human-Aware Science Ontology and the Semantic science Integrated Ontology (SIO) [14], encoding knowledge about science-related entities and their characteristics.

4.1.4 Online Seminar on Semantic and Ontology Translation (Support Activity)

The data integration scenario described in subsection 4.1.3 was presented and discussed during Arrowhead's Winter 2024 PI meeting in Luleå on January 23rd, 2024. During the discussion, the need for members of Task 4.1 to further explore semantic variables through a seminar was identified. Subsequently, an Online Seminar on Semantic and Ontology Translation was developed and presented on March 1st, 2024, to the project partners. From the seminar, it became apparent that there was a need to provide concrete examples of semantic variables based on Industry 4.0 data models. The semantic concept necessitates the complex reuse of terms from multiple ontologies. Without such examples, it is challenging for scientists unfamiliar with ontologies to comprehend how semantic variables are constructed from existing ontologies.

Further steps were taken during March 2024 and the subsequent two months to identify and understand existing data models used in the automotive industry, with the specific aim of identifying datasets applicable to Arrowhead. This effort partially overlaps with WP6 but is specifically focused on identifying concrete semantic variables that can serve as examples for the data integration scenario described above. Our current assumption

is that the integration of Industry 4.0 data models required by Arrowhead will be accomplished through the incorporation of semantic variables utilized within Industry 4.0's data models.

4.1.5 Use case Development (Proposal)

When developing the data interoperability scenario discussed in subsection 4.1.3, we recognized the necessity of customizing certain aspects of our general-purpose solutions to suit industry-specific needs. For instance, in analyzing messages exchanged within car micro-controllers, we find it crucial to identify the car's subsystem responsible for a specific data piece, distinguishing, for example, between thermometers in different parts of the vehicle. To deepen our understanding of industry applications based on Arrowhead, DITAG, and SEMVAR.CC, we've chosen the automotive sector as our vertical domain for in-house exploration. This selection stems from the automotive industry's widespread market presence. To achieve this, we've (1) studied data models outlined in academic literature and (2) characterized the types of data commonly exchanged within vehicle components. Additionally, we're committed to adapting the automotive scenario for implementation within the Arrowhead ecosystem. This scenario will involve a strategy for obtaining data from existing automotive data models and datasets, encompassing both synthetic and real data.

Current Findings As part of the process of understanding how data communicates inside vehicles, it is necessary to comprehend the structure of communications between Sensors and ECUs. Figure 1 in [15] enumerates the groupings of sensors: Powertrain (for vehicle performance), Chassis, and Body (for occupants' needs). Furthermore, even considering that some data flow in proprietary protocols, we can identify several open-source protocols that concatenate data into the final messages flowing in the CAN network, useful for isolating the signal data for further analysis. Identified protocols include: Flexray [16], LIN-LIN [17], CAN-FD [18], CANopen [19], DeviceNet [20], and TTP [21]. The port available for reading data in real-life scenarios of car readings is the OBD connector, through which we will be able to read CAN BUS data.

Next Steps - Reverse Engineering Having identified the proprietary protocols in signal data, communicating from sensors to ECUs, there is a need to capture the real data, decrypt the origin of binary streams, values, units, and timestamps, and provide the output to the Arrowhead ecosystem. We propose the following steps to achieve the above requirements:

- Setup a Virtual Controller Area Network to simulate data transmission.
- Study and implement hardware cracking software to identify the models.
- Implement stream communication to Arrowhead through the MQTT [22] protocol.

4.1.6 The Industrial Data Ontology (IDO) (Proposal)

Task 4.1 together with WP3 and WP9 initiated the development of a novel approach to modeling industrial installations, utilizing ontologies aligned with the semantic web and the Industrial Data Ontology (IDO) to promote semantic interoperability.

The approach to modeling industrial installations embraces the integration of ontologies with the semantic web and IDO, marking a significant advancement in the field. This alignment is pivotal for achieving semantic interoperability, facilitating seamless communication and data exchange across diverse systems within the industrial landscape.

Central to the methodology is the validation of models through reasoning mechanisms. By employing reasoning techniques based on the semantic web, LTU plans to ensure the accuracy and coherence of the modeled industrial installations, enhancing reliability and applicability across various domains.

IDO, recognized under the ISO standard ISO 23726-3, is being developed under the leadership of the POSC Caesar Association. This standard serves as an upper ontology specifically designed for industrial data, aiming to enhance semantic interoperability in various industrial settings.

Consider a typical laboratory setup, which includes a hydraulic pump connected to two tanks, each equipped with level sensors. Using IDO to model this system does more than simply represent data; it provides a comprehensive description of the pump's functions, activities, and connections. Moreover, it contextualizes

the pump within the broader system, highlighting its role as a cyber-physical system, its dependencies, and its interactions with other components.

The application of IDO allows for the transformation of raw data into meaningful information by detailing the relationships and contexts of the components involved. For instance, the ontology helps articulate the pump's purpose, how it interacts with the tanks, and its role within the entire system.

A key advantage of using IDO is evident in scenarios requiring regular maintenance and replacement of components like the pump. By including data such as flow rate and vibration readings into the ontology, it is possible to enhance predictive maintenance strategies. This integration facilitates the analysis of historical data, making maintenance more efficient and timely.

The ontology supports reasoning processes that can deduce insights from the data. This capability allows for the automatic generation of error prompts, pinpointing deviations, or completing axioms based on the observed data. Such features not only improve operational reliability but also extend the lifespan of the equipment by ensuring timely interventions.

Overall, the use of IDO not only enriches data representation but also empowers industrial systems with improved maintenance capabilities and enhanced operational efficiency.

4.1.7 Broker Translator (Prototype)

Another proposed solution is the Broker Translator, which facilitates message translation and redirection among brokers, enabling communication across multiple devices of any protocol within a high-performance environment. It offers advanced configurability while ensuring the fulfillment of Quality of Service (QoS) delivery requirements. Additionally, instantiation within the Arrowhead Framework as a protocol translation service enhances flexibility, scalability, and availability. This configuration enables consumers to request sensor information from a specific middleware by issuing a request to the multi-protocol translation service.

The Middleware ensures that each group of connected devices includes at least one producer and one consumer. Internally, it generates a consumer and a producer for each connection.

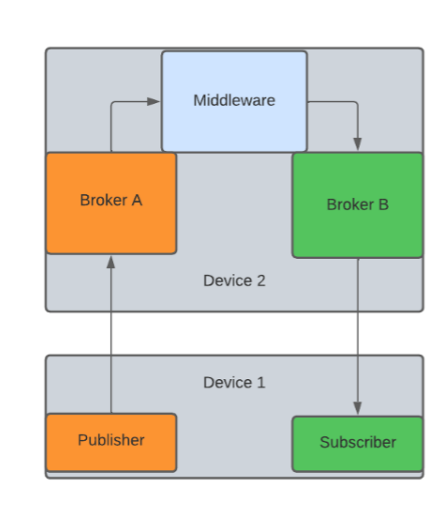


Figure 7: Modular view of the translation.

This way, messages will travel in this sequence, Figure7:

1. Client Publisher of type A (i.e., a Sensor)
2. Broker of a protocol of type A
3. Middleware's subscriber for type A
4. Middleware's publisher for type B
5. Broker of a protocol of type B

6. Client Subscriber of type B (i.e. a Data Center)

The protocol translation also incorporates support for various types of Quality of Service (QoS). In MQTT, Kafka, and AMQP, there are 3 levels that specify the reliability of message delivery:

- QoS 0: Messages are delivered “At Most Once.”
- QoS 1: Messages are delivered “At Least Once.”
- QoS 2: Messages are delivered “Exactly Once.”

Architecture The application’s architecture was designed with protocol scalability in mind, making it so any protocol can be added, and any protocol can redirect to any other, Fig . 8.

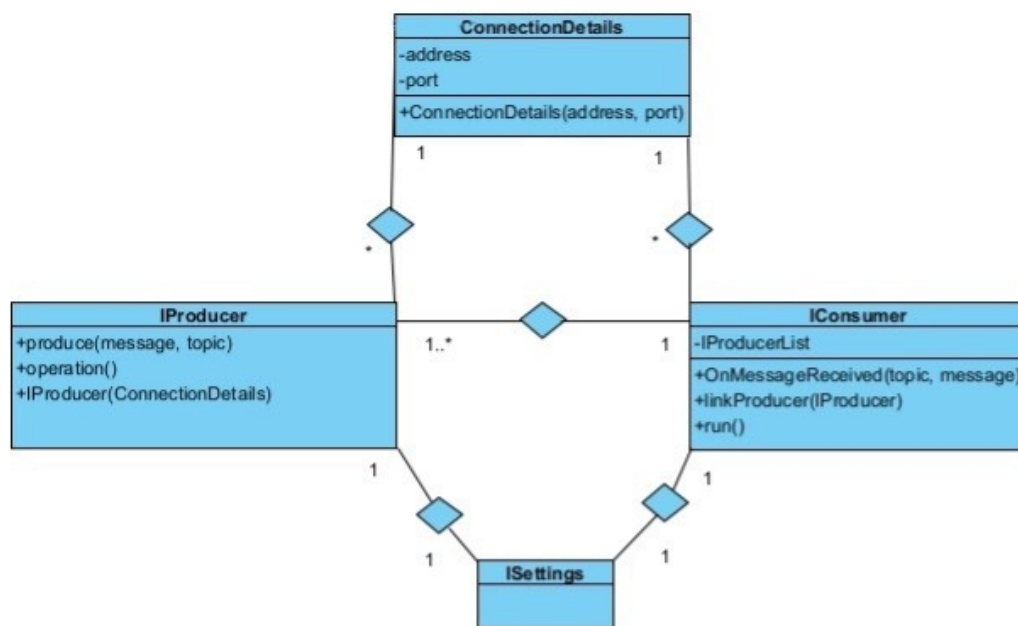


Figure 8: Architecture

To enable support for any new protocol, new publishers or subscribers must implement the two abstract classes, "IProducer" and "IConsumer." Notably, the "IConsumer" class includes a list of "IProducers" and an already implemented method named "OnMessageReceived(topic, message)." This method must be invoked whenever a new message is received. Through this method, every "IProducer" associated with that "IConsumer" will execute their implemented "produce(topic, message)" method, generating a new message to a broker based on the protocol's implementation.

This approach isolates the implementation of each protocol from one another, while also simplifying the support for new protocols or new classes for already supported ones. This flexibility accommodates users who may desire special behavior beyond simple message redirection.

Configuration File Regarding the mapping of translations, QoS configurations, and topics, there exists a JSON file where users can configure all these options, Figure 9

- **Streams** – This component will be responsible for mapping the producer with the consumer.
- **Internal ID** – This ID concerns the "Streams" entry only, so the application can know which devices are supposed to be connected.
- **Protocol** - Refers to the messaging protocol this device is producing/consuming to/from.

```

"Producers": [
  {
    "internalId": "device01",
    "protocol": "kafka",
    "additionalProps": {
      "topic": "test",
      "connectionTimeout": "20",
      "qos": 0
    }
  }
],
"Consumers": [
  {
    "internalId": "device03",
    "protocol": "kafka",
    "additionalProps": {
      "clientId": "client01",
      "topic": "teste",
      "qos": 0
    }
  }
],
"Streams": [
  {
    "fromProducers": "device01",
    "toConsumers": "device03"
  }
]

```

Figure 9: Configuration File

- **Additional Props** - These concern the properties of the equivalent consumer/producer the application will create to communicate with the user's device, and not necessarily the device itself (the user's device may, for example, produce messages with a QoS of 2, but the user may declare for its Middleware consumer equivalent to receive messages with a QoS of 0).

4.2 AI-Based Translation

The following section summarizes the efforts and progress accomplished during the first 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 AI Translation Scenario Analysis (Support Activity: Analysis)

Effective communication between systems relies on compatible interfaces and shared data understanding, encompassing factors like communication protocol, encoding, encryption, compression, message structure, payload key values, and semantics. Understanding the received data is crucial for its proper utilization. However, communication may fail due to interface mismatches or data incompatibility. Translation techniques can bridge these gaps, enabling autonomous translation through an adaptor that modifies and reformats data for compatibility between systems.

Autonomous translation is vital in various scenarios, such as temporary system substitution to maintain operations during issues, integrating new systems with vendor or version differences, and updating old systems while preserving compatibility.

To simplify analysis, consider a basic scenario where two systems, System A (provider) and System B (consumer), communicate via an automated translator acting as an adaptor. Each system is described in a metadata-containing description file, serving as a reference for further discussion, Figure 10.

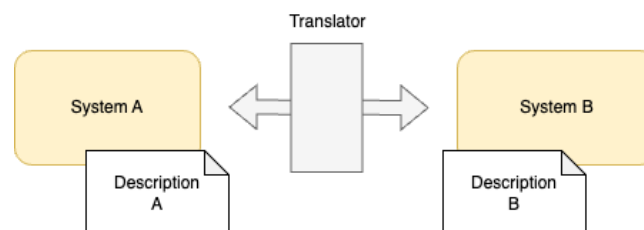


Figure 10: Translation Scenario

The scenario has been analyzed, uncovering issues linked to autonomous translation between heterogeneous systems. This undertaking aids in comprehending the challenges that may confront the WP and

establishes a foundational framework for alternative solutions. The identified challenges have been categorized into three distinct groups:

Mismatch Identification When considering autonomous translation, one of the initial challenges is autonomously identifying mismatches between components. Without autonomous identification, proposed solutions cannot be implemented, thus hindering the primary objective of achieving successful communication.

Several challenges are associated with addressing this issue:

- Comparing interfaces and systems is necessary to identify mismatches. While system descriptions can aid in this task, their use entails assumptions and challenges, as detailed in the following subsection.
- Utilizing a description file necessitates a comparison mechanism capable of not only detecting mismatches but also determining their type and identifying the optimal solution.
- Careful definition of system comparison and communication monitoring is crucial to prevent network and system disruptions.

Description Abstraction As noted earlier, having an accurate and comprehensive system description aids in identifying communication mismatches and supports autonomous tasks such as translator generation, monitoring, and documentation.

However, abstracting such a document poses several challenges:

- Defining the content and level of detail for the description is inherently challenging, as it raises questions about what information should be included and in what format.
- Achieving autonomous abstraction of system information presents another hurdle. Documents that require significant developer effort without immediate benefit are often underutilized. Autonomous generation of descriptions from system code would enhance usability.
- Ensuring the completeness and accuracy of the information is crucial. The description must be reliable and trustworthy.
- Ambiguity in the description further complicates its abstraction and leads to errors.

Translator Generation The translator plays a pivotal role in addressing communication mismatches. Its generation is a critical endeavor fraught with various challenges:

- Autogenerating a translator requires a vast amount of accurate, complete, and accessible information. The generation mechanism must comprehend this information to produce suitable transformations.
- The generation process must adapt to diverse scenarios, including cases where consumer-requested information is incomplete or ambiguous.
- The mechanism must exhibit intelligence in generating modifications and interpreting data across different semantics, encodings, and protocols.
- Semantic understanding presents a complex challenge, requiring nuanced handling.

4.2.2 AI-based Frameworks (Support Activity: Analysis)

Based on the previous scenario, Task 4.2 has conducted a comprehensive investigation into AI-based frameworks that can be applied to the task of translation. The goal was to collect and identify the feature list of the currently available frameworks and libraries that can be used to translate between different data models or protocols. As a preprocessing task, the labeling and information-extracting stages should also be considered during the investigation.

Based on the WP3 D3.1 Deliverable and the use-case-related meetings, XML and JSON are widely used representation formats of the currently applied standards. To solve possible interoperability problems, AITIA

started to collect and investigate AI frameworks to process human readable, text-based information, and create XML – JSON, or XML – XML translation models.

The AI-based text clustering, context recognition, and text generation significantly improved in the last few years. Focusing on the text translation solutions, the following categories could be differentiated:

- Natural Language Processing (NLP)
- Neural Machine Translation (NMT)
- Large Language Models (LLM)

NLP is a branch of artificial intelligence that focuses on the interaction between computers and humans through natural language. The goal of NLP is to enable computers to understand, interpret, and respond to human language in a way that is both meaningful and useful. NLP combines computational linguistics (rule-based modeling of human language) with statistical, machine learning, and deep learning models. Techniques used in NLP include syntactic analysis and semantic analysis. Syntactic analysis involves parsing the grammatical structure of sentences, whereas semantic analysis involves understanding the meanings conveyed by them. Recent advances in NLP are largely driven by machine learning models, particularly deep learning, which has led to significant improvements in processing and generating natural language.

The features of the Natural Language Processing (NLP) models are:

- Applies Neural Networks
- Analyse patterns in text (emotion identification, classification tasks)
- Chatbots, text generation, document summarization

Neural Machine Translation (NMT) is a method of machine translation that utilizes artificial neural networks to predict the likelihood of a sequence of words, typically in a target language, given a sequence of words in a source language. This approach to translation is fundamentally different from earlier statistical methods. NMT models the entire translation process as a single integrated system, which can be trained end-to-end. NMT works by using large neural networks, particularly those based on the architecture of Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, or more recently, Transformer models. These networks are capable of handling sequences of data and can maintain an internal state to remember previous inputs, which is beneficial for understanding context in language.

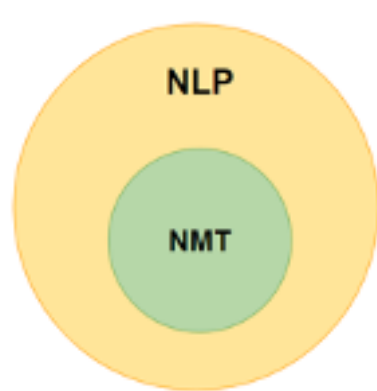


Figure 11: The relationship between the NLP and NMT models

The features of the Neural Machine Translation (NMT) models are:

- Applies Neural Networks
- Translates source text to a target text (e.g., Google translate)
- Requires little supervision
- Can understand the context of the words – > high accuracy!

Large Language Models (LLMs) are advanced artificial intelligence systems that are trained on massive datasets of text to generate, understand, and predict human language. These models, which are a subset of machine learning models known as transformers, have dramatically increased the capabilities of natural language processing (NLP) tasks.

The features of the Large Language Models (LLMs) are:

- Learns text patterns.
- LLMs are particularly good at understanding the context of the input text.
- It could be used for translation services.
- The models could be fine-tuned with task-specific datasets.

The following list summarizes the set of Machine Learning tools and libraries, which are in the scope of the initial phases.:

- PyTorch
- Scikit-learn
- SpaCy
- Wit
- NLTK
- OpenNMT
- Word2Vec
- Microsoft CoPilot
- ChatGPT
- LangChain

During the first year of the Arrowhead fPVN project, the following frameworks were selected as possible tools to perform translation:

Scikit-Learn Scikit-learn is an open-source machine-learning library for the Python programming language. It's built on NumPy, SciPy, and matplotlib, and it provides simple and efficient tools for data mining and data analysis. Scikit-learn is widely used in academic and commercial settings due to its broad range of algorithms and its ease of use.

The library includes support for various machine learning tasks such as:

- Classification: Identifying to which category an object belongs to.
- Regression: Predicting a continuous-valued attribute associated with an object.
- Clustering: Automatic grouping of similar objects into sets.
- Dimensionality reduction: Reducing the number of random variables to consider.
- Model selection: Comparing, validating, and choosing parameters and models.
- Preprocessing: Feature extraction and normalization.

Scikit-learn is known for its consistent API, which allows users to easily experiment with different models, and its comprehensive documentation that helps users and developers understand and use its functions effectively.

NLTK Natural Language Toolkit (NLTK) is an open-source Python library designed for working with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text-processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. This makes NLTK a leading platform for building Python programs to work with human language data.

It is particularly suited for educational and research purposes in the field of natural language processing (NLP) and computational linguistics due to its comprehensive features and documentation. NLTK allows for the handling of a wide variety of linguistic data types and tasks, including:

- Text tokenization
- Part-of-speech tagging
- Syntax parsing
- Sentiment analysis
- Named entity recognition

NLTK is widely used for prototyping and building research systems, and it's often employed in teaching and learning due to its accessibility and the breadth of its capabilities.

Word2Vec Word2Vec is a popular method in Natural Language Processing (NLP) for generating vector representations of words, where words that have similar meanings are embedded in close proximity to each other in a high-dimensional space. Word2Vec is based on neural networks.

The word vectors capture many linguistic regularities and patterns. For example, vector operations can uncover semantic relationships between words, such as analogies. Word2Vec can be used for various applications including:

- Semantic word similarity,
- Document clustering,
- Part-of-speech tagging,
- Named entity recognition.

The model's ability to capture syntactic and semantic word relationships using relatively simple architectures has made it a fundamental tool in the field of NLP.

OpenNMT OpenNMT is an open-source Neural Machine Translation framework that provides implementations in both PyTorch and TensorFlow. OpenNMT supports various machine learning models and techniques that go beyond translation, including general sequence learning tasks.

Key features of OpenNMT include:

- **Modularity:** It allows for easy customization and extension of its architecture to suit different machine learning tasks beyond just machine translation, such as summarization, image-to-text, and speech processing.
- **Efficiency:** OpenNMT is optimized for high performance, supporting multi-GPU training which makes it suitable for enterprise-level deployment.
- **Pre-trained Models:** It provides access to a range of pre-trained models that can be used as is or fine-tuned for specific languages or tasks.
- **Wide Language Support:** The framework supports translation between numerous languages, facilitated by its flexible architecture.

OpenNMT is widely used in the research community and industry, as it provides a robust, scalable solution for a variety of sequence-to-sequence learning problems. It has become a popular choice for developing production-ready solutions that require language processing capabilities.

ChatGPT and Microsoft CoPilot ChatGPT is a variant of the GPT (Generative Pre-trained Transformer) models developed by OpenAI. It is designed specifically for understanding and generating human-like text based on the input it receives. ChatGPT can answer questions, provide explanations, generate creative content, simulate conversation, and even assist with coding tasks. It's trained on a diverse range of internet text but designed to follow strict ethical guidelines and provide useful, accurate, and safe information. It's utilized in various applications, from chatbots to educational tools, providing users with a conversational interface to interact with AI. Future work In the next phase of the fpvn project, AITIA Internation Inc. will investigate the following AI-related frameworks regarding translation solutions:

- Anthropic (Claude 3 model)
- LangChain framework
- ChatGPT fine-tuned models

4.2.3 AI-based labeling and metadata filling in Eclipse Arrowhead (Prototype)

Task 4.2 introduced a new, AI-based Eclipse Arrowhead Support System module, which name is "Service Inventory".

The "Service Inventory" module is a new Arrowhead Core System module to extract information and create labels based on text-related information (e.g., text-based data, XML, JSON). During the Arrowhead registration procedure, before calling the "Service Registry" module, the Producer should communicate with the "Service Inventory" module to get the metadata content, which will be registered into the "Service Registry". This metadata content will be filled by the "Service Inventory", which applies AI submodules to understand the context, extract meaningful information, and produce several labels. The AI-filled metadata could be used in later procedures, e.g., during the metadata search procedure, or in translation tasks. Based on this information, the Orchestrator could select (or recommend) Translator modules to solve interoperability problems, and to grant the expected input for Consumers from an originally inappropriate format. The module is under construction. In the initial phase, it will apply NLTK-based models.

GitHub link: <https://github.com/Aitia-IIOT/ah-service-inventory-poc>

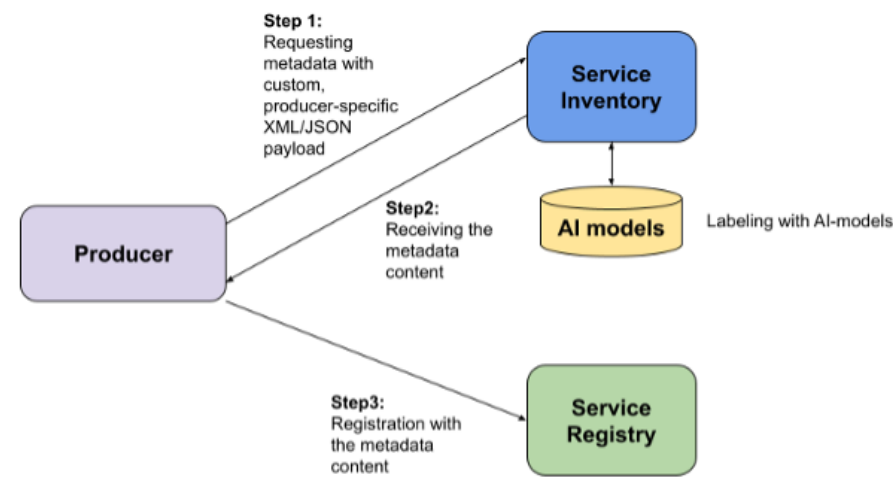


Figure 12: Service Inventory module as an Arrowhead Support System

4.2.4 AI Translation Solutions (Prototypes)

The following sections describe the efforts to investigate and test the use of the aforementioned frameworks and models in different examples.

Word2Vec-based models The python_scripts/Wod2Vec folder demonstrates how to train a Word2Vec model using text files, and then use the trained model for translation purposes.

The text file should be cleaned and preprocessed for better results, which means:

- lowercasing
- removing punctuation
- removing stopwords

preprocessTextFiles.py presents a possible preprocessing example.

Regarding the training procedure, the following parameters are relevant:

- The vector_size parameter specifies the dimensionality of the word vectors.
- The window parameter specifies the maximum distance between the current and predicted word within a sentence.
- The min_count parameter ignores all words with a total frequency lower than this.

Example #1: Creating Word2Vec model from text file modelCreationExample.py is an example script to create a Word2Vec model. In this example, the content of the input text file (inputText_TemperatureWikipedia.txt) is the Wikipedia page of “Temperature” [23].

Before training the model, the preprocessTextFiles.py script was used to preprocess the content.

ListWord2VecKeyWords.py could be used to print the list of the words which are learned and stored in the model.

Example #2: Clustering with Word2Vec The second example demonstrates the usage of a Word2Vec model for clustering purposes. using_trained_word2vec.py applies a pre-trained model for K-Means clustering.

Example #3: Translation with Word2Vec using similarity search The third example demonstrates the similarity search mechanism based on a trained Word2Vec model to translate JSON keys. The Word2Vec model was trained with the “Temperature” Wikipedia website.

The translation script (translationWithWord2Vec.py) calculates a similarity value between the keys in the input JSON object, and a predefined list: temperature, energy, and sound.

Since the “humidity” word doesn’t exist in the trained Word2Vec model, it will be not translated, because the similarity value couldn’t be calculated. The script changes the JSON key based on the highest similarity value.

```
{
  "c":25.0,
  "kelvin":42,
  "celsius":33,
  "energy":10,
  "humidity":42,
  "pressure":12
}
```

Figure 13: The input JSON object before translation

```
{
  "temperature": 25.0,
  "temperature_1": 42,
  "temperature_2": 33,
  "energy": 10,
  "humidity": 42,
  "temperature_3": 12
}
```

Figure 14: The output JSON object after translation

Example #4: Applying more Word2Vec models for similarity search and for translation The fourth example demonstrates the usage of more Word2Vec models (similaritySearch.py).

In this example, the models were trained with two Wikipedia pages: “Temperature” [23] and “Humidity” [24]. ListWord2VecKeywords.py could be used to list the words that are learned by the models.

The similaritySearch.py calculates a similarity value based on the trained models for a predefined word list: “temperature, c, k, celsius, kelvin, energy, water, vapor”

The script selects from “temperature” or “humidity”, and prints the best match for each word based on the models. It also prints the calculated similarity value based on the trained models. Based on the results, the similarity search with more pretrained, specific models could be used for translation tasks.

NLTK-based labeling The NLTK framework could be used for context recognition and labeling purposes.

The model can be trained with text-based information that can be JSON or XML content as well. After the training procedure, sentences, JSON objects (or part of a JSON object), XML contents (or part of an XML content) could be used as input information, and the model makes a decision about the meaning.

The NLTK_Labeling.py script demonstrates the possible use of the NLTK framework. The script requires an input folder, where the subfolders will be used for training. The name of each subfolder will be the label, and the content of the subfolders will be the training data. As an example use-case, the NLTK model has been trained with EC and PLM XML files from UC 7.1. The trained NLTK model could differentiate the input XML content with 100% accuracy and label it with the right subfolder name.

OpenNMT-based translation OpenNMT is an open-source Neural Machine Translation framework. It offers two Python frameworks:

- OpenNMT-py, which is a PyTorch-based solution.
- OpenNMT-tf, which is a TensorFlow-based system.

As a first step, it has been investigated the OpenNMT-py framework [24] regarding XML to JSON, and JSON to SenML (JSON format with predefined keys) translation.

The results were presented in [25] 3rd International Workshop on Analytics for Service and Application Management.

The accuracy of the OpenNMT-py model was 82% in the XML-JSON-related case, and 87% in the JSON-SenML-related case. After the examination of the failed translation cases, results concluded that OpenNMT-py is not suitable for translating Key-Value pairs with random numeric values, or where the value is from a wide numeric range.

4.3 Model-Based Translation

The following section summarizes the efforts and progress accomplished during the first 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 Papyrus Tool (Prototype)

Papyrus [26] is a Model-Based Engineering tool implementing both OMG standards: UML for software engineering and SysML[27] for system engineering. Papyrus is Eclipse-based (The architecture of Papyrus is illustrated in Figure 15 and relies on EMF (Eclipse Modeling Framework) to implement model-to-model translations. In Papyrus, to implement any model-based translation from a Source data model (A) to a Target data model (B); the user should perform three steps as follows:

1. Build the Meta Model of the source data model A using one of the 3 technologies: UML, Ecore, or XSD
2. Build the Meta Model of the target data model (B) using one of the 3 technologies: UML, Ecore, or XSD
3. Define a mapping between concepts in the Meta Model A and their equivalent in the Meta Model B (in a table)

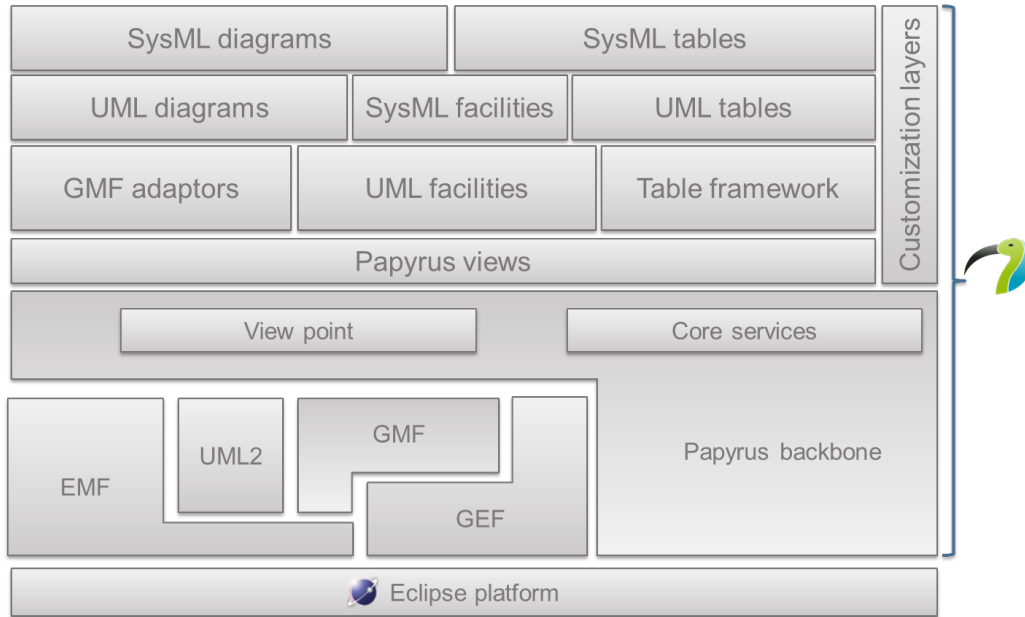


Figure 15: The Papyrus tool architecture

4. Implement using 3rd generation languages (Java, Python...) or model transformation specific languages (ATL, Qvto, Xtend, Acceleo...) the mapping.

Several examples of model-based translation are implemented in Papyrus in addon plugins for instance AAS2UML, SysML v2OPCUA, UML2JSON, UML2Java, C, C++, or Python... The transformation from the SysML model to the OPCUA model explained in detail in [28] is illustrated in Fig 16. This model-based translation follows the steps described above.

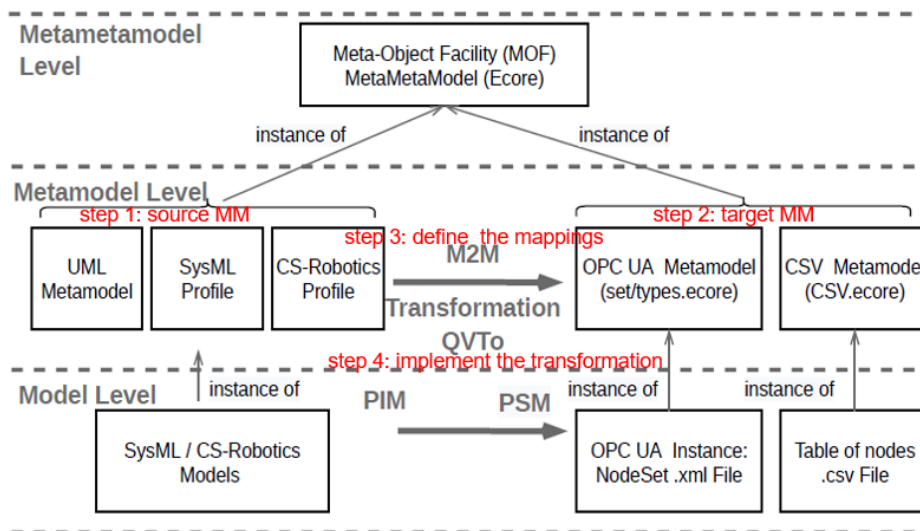


Figure 16: Example of a Model 2 Model transformation process from SysML to OPCUA

WP3 deliverable highlights the list of the most used data model standards by the project use cases, and possible pairs (source/target). This permits the selection of the model-based translation task with the most

appropriate standards. The DEXPI [29] and CFIHOS [30] were chosen for the reasons below:

- Both standards are open sources
- Both standards are used for the same domain and conform to the same STEP standard ISO 15926 (possible use of both of them for the same system/scenario)
- Both of them propose a UML/XML serialization for their data model

Task 4.3 has started the analysis of both standards (steps one and 2 of the model-based translation explained above) However, to prototype a translation (step 4), we need to define a mapping (step 3) and this is only feasible if we have examples and scenarios where the two data models coexist. We planned (before the end of the year) to prototype a model-based translation involving DEXPI and CFIHOS once examples and scenarios of the use of such translation are presented and explained by the use cases.

SysML v1	SysML v2
class / property	item definition / item usage
block / part property	part definition / part usage
value type / value property	attribute definition / attribute usage
interface block / proxy port (flow property)	port definition / port usage (directed usage)
association block / connector	connection definition / connection usage interface definition / interface usage
item flow	flow connection definition / flow definition usage

Figure 17: Mapping proposed by the OMG between some SysML v1 and SysML v2 concepts

To illustrate the model-based framework in Papyrus, it is proposed to develop a model-to-model translation from SysML v1 to SysML v2 [31] standards. This will illustrate the technologies used in Papyrus. Unlike the couple (DEXPI, CFIHOS), The OMG already defined the mapping [32] between some SysML1 and SysML v2 (step 3) as shown by Figure 17. Some examples in the literature [33] illustrate the mapping.

4.3.2 SysML 1.6 to SysML 2 translation (Proposal)

In this subsection, the Papyrus model-based translation from SysML1.6 to SysML2 is detailed. The translation is under development in an addon plugin to be installed in Papyrus as an update site. In order to test the translation, the user should build using Papyrus its source model (for example the model presented graphically in Figure 18 and Figure 19 and right-click on the semantic file in the Papyrus Model Explorer View and select the menu "Translate from Sysml1 to SysML v2". The translation will run and an equivalent SysML v2 (According to the defined mapping between the SysML1 concepts and the SysML v2 concepts) file will be generated.

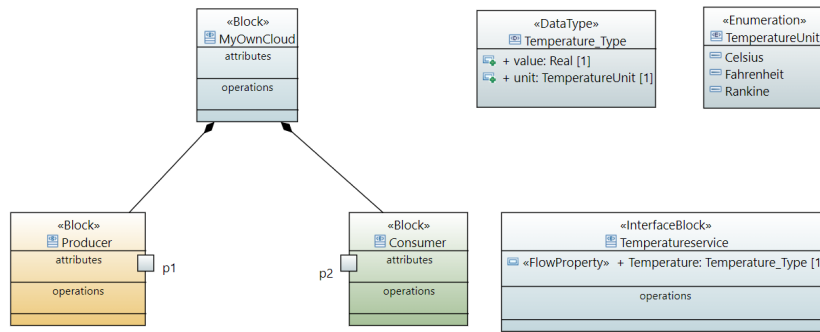


Figure 18: SysML1 source model example (a BDD diagram)

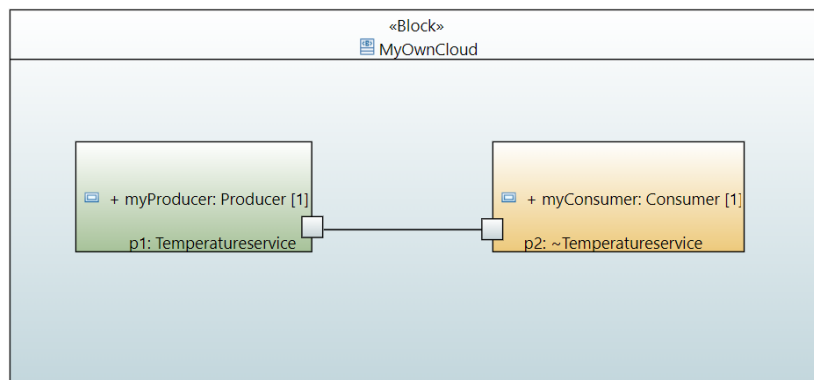


Figure 19: SysML1 source model example (an IBD diagram)

The SysML v2 file generated using the model-based translation can be illustrated in Figure 20. It is worth mentioning here that the translation is generating a file .sysml that is conformant to a snippet of the SysML v2.ecore Metamodel provided in the OMG pilot Implementation [34]. The serialization of the .sysml file in the textual and graphical view of Figure21 is supported in the OMG Implementation.

Our translation takes as source the SysML1 MM (provided by Papyrus as an encore file), an example of a sysml1 model (built with Papyrus), and the snippet SysML v2 MM (provided by the OMG Implementation as an ecore file). As a result, the translation will generate the output SysML v2 file according to the defined mapping.

Our translation is implemented using Java. Another SysML v1 to SysML v2 translation is proposed by the OMG but is not open source. Only the mapping is open source [32]

We can notice by studying this example, that apart from the technical serialization of the input and the output metamodels (ecore or XSD files), the most important task is the definition of the mapping itself. As an example, we have chosen to translate the SysML1 Temperature DataType (with its Temperature Unit Enumeration) to the pre-defined SysML v2 element **ThermodynamicTemperatureValue** defined in the **ISQBase** Library. There is no unique mapping to define a translation, even between the same input and output metamodel. In the illustrated example, the developer could choose another mapping for the Temperature DataType and translate the SysML1 DataType to a new definition of SysML v2 elements (a new DataType and Enumeration with the same literals as defined in the input SysML1 model). The idea here is to reuse as much as possible the predefined SysML v2 libraries such as the ISQBase and its predefined Types and Units.

The Graphical representation of the actual OMG pilot implementation relies on PlantUml [35], a widely used representation to rapidly illustrate UML-like models. However, we can notice from Figure 21 that the graphical representation of the SysML1 Figure 19 mainly for the IBD diagram is more user-friendly and conforms to the standardized notation in the OMG standard. In fact, and as it was stated on the PlantUml website, PlantUML does not restrict the creation of inconsistent diagrams — such as mutual inheritance between two classes. Consequently, it functions more as a drawing tool rather than a modeling tool. SysON [36], An Eclipse open source project is aiming to provide a new graphical and textual editor for the SysML v2 standard, our translation could in the future target this editor as well.

```

import ScalarValues::*;
import ISQBase::*;

package 'ProducerConsumerBDD' { // BDD
  port def TemperatureService //InterfaceBlock
  {
    attribute Temperature:ThermodynamicTemperatureValue;
  }
  part def MyOwnCloud{ // Block
    part def Producer;
    part def Consumer;
  }
}

package 'ProducerConsumerIBD' { //IBD
import 'ProducerConsumerBDD'::*;
part myOwnCloud: MyOwnCloud{ // Part
part myProducer:Producer {
port p1: TemperatureService; // ProxyPort
}
part myConsumer:Consumer {
port p2: ~TemperatureService; // ProxyPort
}
connect myProducer.p1 to myConsumer.p2; // Connector
}
}

```

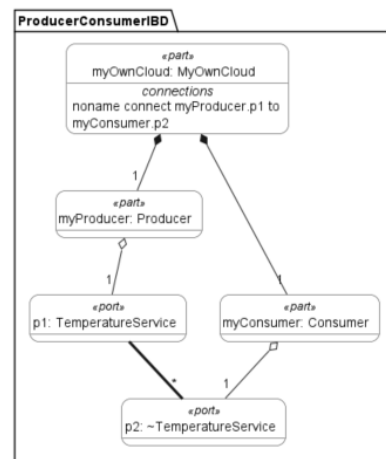


Figure 20: SysML v2 target model example (textual view)

Figure 21: SysML v2 target model example (graphical view)

4.3.3 Cross-Domain Translation and Validation Workbench (Proposal)

In this section, relying on the aforementioned components, we provide an overview of an envisioned multi-domain workbench, combining the power of other data formats, the still most-established SysML v1 standard, as well as the emerging SysML v2, arguing for its suitability as a validation base language – validation being a particularly important general use-case in dynamic contexts such as fPVN, with a lot of ad-hoc translation scenarios getting supported.

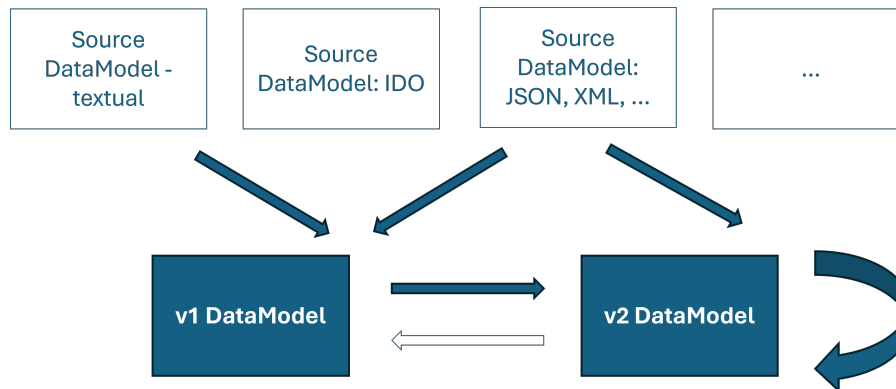


Figure 22: A Cross-Domain, SysML-centric Translation and Validation Workbench for Arrowhead fpvN

Figure 22 shows how different source data formats, each according to their own, standard or non-standard data models, even on varying abstraction levels, can interact with and through SysML v1 and v2 intermediate models (dark boxes in the bottom row). The v1 Data Model can be fitted with the rest of the Arrowhead SysML DSL mentioned before and developed by LTU. The "internals" of the arrow from v1 to v2 has been described in the previous section. In the following, we describe the standardization context and the validation and integration capacities of SysML v2 as of today.

SysML v2 incorporates enhanced features such as improved usability, expressiveness, and consistency, which are essential for modeling complex systems spanning various domains. Validation processes in SysML v2 allow engineers and developers to verify that every aspect of the system model aligns with the operational needs and complies with specified requirements. This is particularly important in mission-critical industries like those in fPVN, where system failures can have severe implications. Effective validation ensures that models are not only syntactically correct but also semantically precise, reducing the risk of costly errors and increasing reliability in the operational phase – supporting the latter being the principal aim of the design-time workbench described here.

Adopting SysML v2 and its validation techniques offers substantial benefits. First, it leads to higher quality outcomes by enabling early detection and correction of errors, which significantly reduces the cost and time associated with reworks – especially important for design-time models for ad-hoc translations. Furthermore, SysML v2's validation capabilities support the creation of more accurate and robust models, facilitating better communication and understanding across different teams. This cross-functional collaboration is essential for complex engineering scenarios like those in fPVN. Additionally, validated SysML v2 models can be integrated with other engineering tools and processes, enhancing automation and providing a more streamlined engineering workflow.

SysML v2 is designed to be a robust base language suitable for cross-domain validation due to its comprehensive modeling capabilities that address both hardware and software components of a system. Its rich set of features enables the depiction of complex interactions and dependencies within and across different system domains, from electrical and mechanical to software and network systems. This holistic approach is pivotal when validating systems that operate in an interconnected and interdependent environment. Moreover, SysML v2 supports extensibility and customization, allowing industries to adapt the language to specific needs and domain-specific characteristics, thus enhancing the effectiveness of cross-domain validation. The language's ability to seamlessly incorporate various validation techniques, such as simulation and formal verification, further underscores its adequacy as a foundational tool in modern systems engineering.

We have already created a generic validation demonstrator showcasing the potential of using SysML v2 models in such a context, also involving integration with some established central to managing complex engineering (e.g., Confluence). In the remaining project phases, we aim at enhancing these capacities and integrating them tighter with use cases and other technologies (SysML v1, Papyrus, the DSL as well as ontology-based workbenches).

4.3.4 Deep-Learning Based Meta-Modeling for Data Translation (Proposal)

Moreover, a comprehensive analysis of the research project's goals, objectives, pivotal technologies, and significance provides stakeholders with valuable insights into the revolutionary capabilities of AI-driven translation models. Through the utilization of ontology learning and Graph Neural Networks (GNNs), this conceptual approach holds the promise of revolutionizing data translation practices. Enhancing interoperability and communication across diverse data models is poised to usher in a new era of efficiency and collaboration within the field. A series of activities has been undertaken to explore this multidisciplinary approach.

ACTIVITY 1: A comprehensive literature review regarding DL-based data-model translation technologies

The primary objective is to explore state-of-the-art AI technologies in-depth, particularly focusing on ontology learning in conjunction with GNNs and/or Large Language Models (LLMs). By exploring these cutting-edge methodologies, the research explores innovative approaches to enhance the accuracy and efficiency of data translation processes.

ACTIVITY 2: Explore the potential of Large Language Models, alone in combination with Ontology learning and/or GNN

Furthermore, the research explores how LLMs can enhance data translation. By leveraging the semantic understanding capabilities of LLMs, the aim is to improve the accuracy and resilience of translation models and effectively tackle the challenges posed by semantic ambiguities and nuances inherent in data models. This exploration seeks to uncover new insights into how LLMs can significantly contribute to refining and optimizing data translation techniques, ultimately leading to more reliable and robust translation outcomes.

ACTIVITY 3: Conceptually Develop and Analyze a specific translation Use-Case, the one IEEE1451 and ISO15925-4

Another key objective is conceptualizing and evaluating a use case scenario, examining the interaction between IEEE1451 and ISO15925-4 data models. This exploration will remain conceptual, prioritizing the theoretical aspects of data translation while avoiding specific implementation details. Through analyzing these standardized data models, the research aims to gain a comprehensive understanding of the intricacies and potentialities involved in translating data across various formats. By integrating the findings from previous activities (Activities 1 and 2), the emphasis will be on developing a case study that serves as a conceptual framework and provides theoretical insights to guide future developments in data translation methodologies. Implementation will not be pursued due to budget constraints.

ACTIVITY 4: Use some available translation frameworks (Ontology/GNN/LLM) on the Use-Case in Activity 3 without new training for evaluation purposes

In this activity, existing translation frameworks such as Ontology, GNN, and LLM will be applied to the use case identified in Activity 3 without undergoing new training. The aim is to evaluate the effectiveness and performance of these frameworks in translating data between the IEEE1451 and ISO15925-4 data models. This evaluation will provide valuable insights into the feasibility and applicability of leveraging these frameworks for real-world data translation tasks. By assessing the outcomes of this evaluation, the research seeks to identify strengths, limitations, and potential areas for improvement in existing translation methodologies.

4.3.5 AeroBridge (Prototype)

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 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, and different partners between 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 detail, S5000F, one of the S-SERIES Specifications, specifies information and data elements required to provide in-service data feedback. Though originating mainly from the aerospace domain, S5000F can be applied to any products, whether mobile or not, in the air, land, sea and space domains, both for civilian and military programs.

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

The process involves, Figure 23:

1. Identification of the use cases with the relevant data involved
2. Understanding of the model behind the raw data
3. Translation of this raw data model into a UML model in accordance with the S5000F
4. Translation of the UML model into an XML format to effectively represent the message exchanged between different systems.

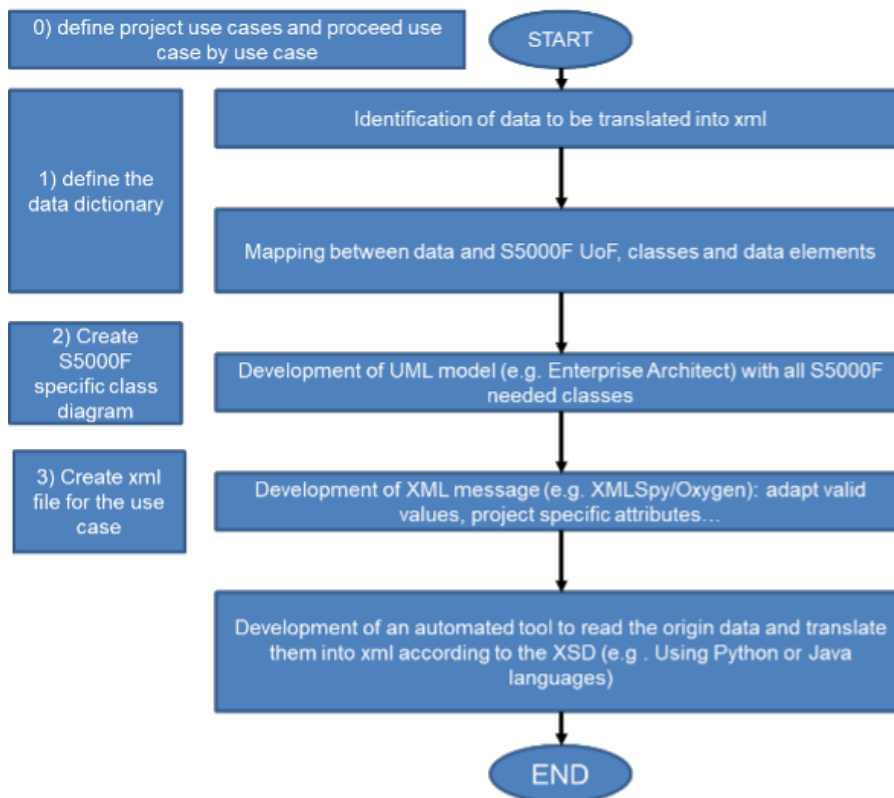


Figure 23: Logical representation of the process

In detail, for each identified use case (e.g. Feedback on Flight Parameters, Faults, Equipment removals, Supplier Work Reports, etc.) a mapping between the raw data and S5000F elements is required to produce the XML message.

The first step is to understand whether there is a use case within the S5000F that already describes the feedback needs and its data. Building upon it, a series of Units of Functionality (UoF) are selected, which summarize the main data elements relevant to a specific topic, including classes and attributes.

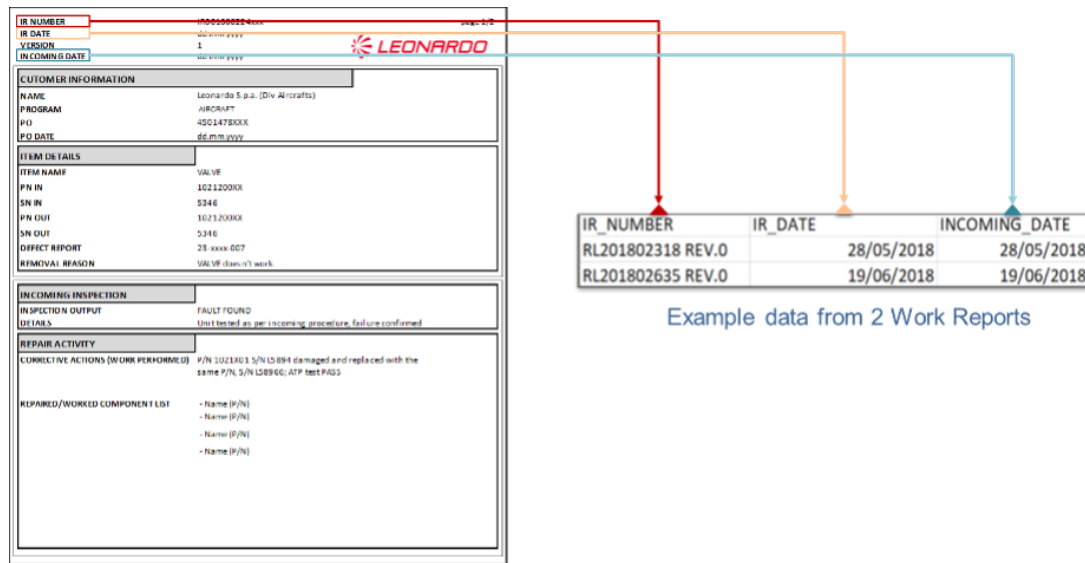


Figure 24: Example of a Use Case, Supplier Work Report

A UML (Unified Modeling Language) model can then be defined starting from the official S5000F data model to describe the required elements and their relationships.

To develop a UML model based on the official S5000F Data Model, commercial software such as Enterprise Architect can be employed. Utilizing the S5000F elements and an existing UML model as references, the mapping process is conducted.

Raw field	S5000F attribute
IR NUMBER	shopFindingsIdentifier
IR DATE	shopReceivedDate
INCOMING DATE	deliveryDate

Table 3: Attributes mapping example, Supplier Work Reports-Report Shop Findings

The UML is subsequently transformed into an XSD, which delineates the structure of S5000F messages in XML format. Each message must conform to an XML schema based on the structure outlined in the UML model. The XSD serves not only to translate the message into XML format but also to validate the data sent by stakeholders, as it enables the definition of permissible values against which the actual content can be checked. Ultimately, the XML Message is generated, as illustrated in Figure 26.

The AeroBridge prototype is an application under development written using Python, that will evolve throughout the entire AfPVN project. Its main capabilities will be:

1. To receive in input the mapping between the raw data and S5000F attributes
2. To gather all the information from the official XSD of S5000F
3. To automatically generate the XSD for the specific Use Case and produce the XML message.

The prototype will work in both ways, from raw data to S5000F XML message and vice versa, while validating the results against the S5000F and the user-defined values rules.

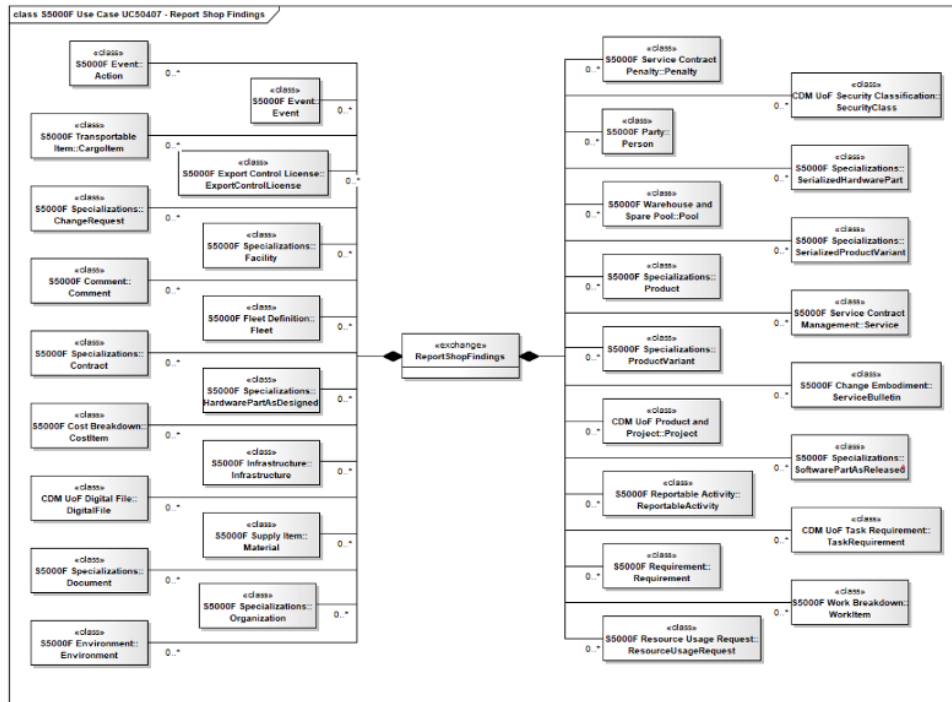


Figure 25: Example of S5000F Use Case, Report Shop Findings (UC50407)

```

<message>
<exampleClass crud="I">
  <shopFindingsIdentifier>RL201802318 REV.0</shopFindingsIdentifier>
  <shopReceivedDate>2018-05-28</shopReceivedDate>
  <deliveryDate>2018-05-28</deliveryDate>
</exampleClass>
<exampleClass crud="I">
  <shopFindingsIdentifier>RL201802635 REV.0</shopFindingsIdentifier>
  <shopReceivedDate>2018-06-19</shopReceivedDate>
  <deliveryDate>2018-06-19</deliveryDate>
</exampleClass>
</message>
  
```

Figure 26: XML Message example.

4.3.6 Other Modeling Languages (Support Activity: Initial Analysis)

Task 4.3 investigated the modeling languages and tools currently employed within the industry. This exploration is essential for understanding how the Arrowhead Framework can facilitate automated translations using established industrial tools and methods.

While UML and SysML are widely recognized, ArchiMate has also been identified as a solution architecture level. ArchiMate is primarily utilized at the enterprise and system-of-systems levels, presenting a challenge in bridging to the interface level, which is crucial for effective integrations and interoperability. Modeling at the system-of-systems level describes what translations are needed to get the system to a running state, while the actual translation occurs at the interface level. Over the past years, a considerable increase in the adoption of machine-readable specifications using various interface definition languages (IDLs) for documenting interfaces has been noticed. Examples include the more modern OpenAPI, AsyncAPI, and protocol buffers, as well as the more traditional WSDL. During the subsequent years of the project, Task 4.3 will continue to explore how to comprehensively model solutions down to the integration and interface level, incorporating industry de facto standards like OpenAPI and AsyncAPI.

4.3.7 P&ID symbol translation (Prototype)

Digitizing old piping and instrumentation diagrams (P&ID) (dumb drawings to intelligent diagrams) brings an additional challenge. Not only do we need to recognize old diagrams and their content, but sometimes we need to change the original symbols to new ones used in the target CAD system. Replacing the symbols is not a straightforward process. A new symbol may be in a different size and have connection terminals in different positions. Thus, without any additional actions, connection lines will not be correctly aligned. As background material for this project, we have used Semantum's Model Broker for Diagrams for digitizing old P&IDs. The Target CAD system is Autodesk's AutoCAD Plant3D, which has P&ID features. We have chosen the software because the industry is widely using the software, thus giving us access to test diagrams. The solution we have developed is not dependent on the chosen CAD system and can be used with other software as well. With the symbol swap process we have developed, we:

1. Extract new symbols from the target CAD system.
2. We digitize an old P&ID with Model Broker for Diagrams. MB4D saves the P&ID as a DEXPI / Proteus file.
3. We replace the old symbols in the DEXPI file with the new symbols.
4. We run a layout fixing algorithm on the symbol swapped P&ID and save the results as a DEXPI file.

The layout fixing algorithm, which we originally developed for different projects to layout automation system operator displays, has been adjusted to handle P&IDs, and operates on the following principles:

1. When a symbol is moved, other symbols and connection lines in front of the symbol are moved along.
2. Object alignments are kept, if possible.
 - (a) Translating an object is decomposed into separate horizontal and vertical translations.
 - (b) Moving an object vertically tries to keep horizontally aligned objects (and vice versa) aligned by translating the objects the same distance.
 - (c) The algorithm includes cycle detection and can break alignments if deemed necessary.
3. Layout is fixed in stages:
 - (a) Line alignments from the original P&ID are forced on the symbol replaced by P&ID.
 - (b) Symbol / Symbol overlaps are resolved (if any).
 - (c) Symbol / Line overlaps are resolved.
 - (d) Line / Line overlaps are resolved.
 - (e) Diagram contents are aligned to the grid (optional).

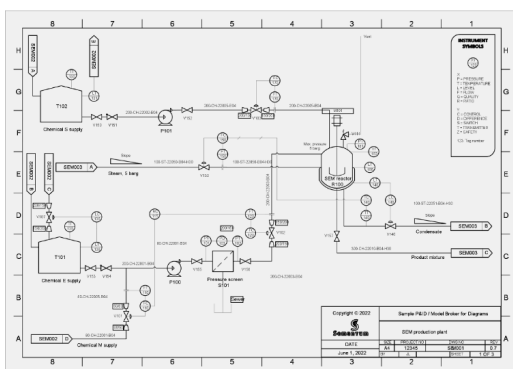


Figure 27: MB4D tutorial P&ID

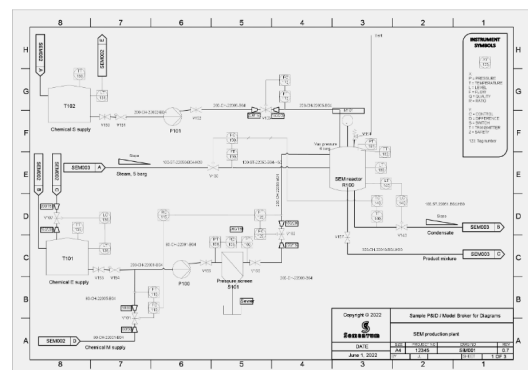


Figure 28: Tutorial P&ID with replaced symbols.

For demonstrative purposes, in the following example, we are using an MB4D tutorial diagram. The tutorial P&ID is given as a PDF file, which is then converted to a DEXPI file with MB4D (Figure 27). The result of

applying symbol replacements is shown in Figure 28. We have picked up the most obvious cases where connectivity is no longer valid in Figure 29. The result of applying the layout fixing algorithm is shown in Figure 30.

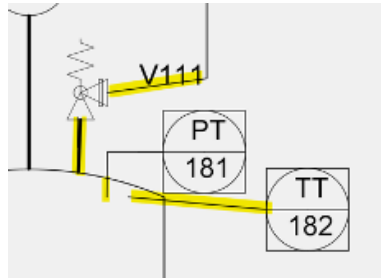


Figure 29: Some highlights of the connection lines that require adjustments.

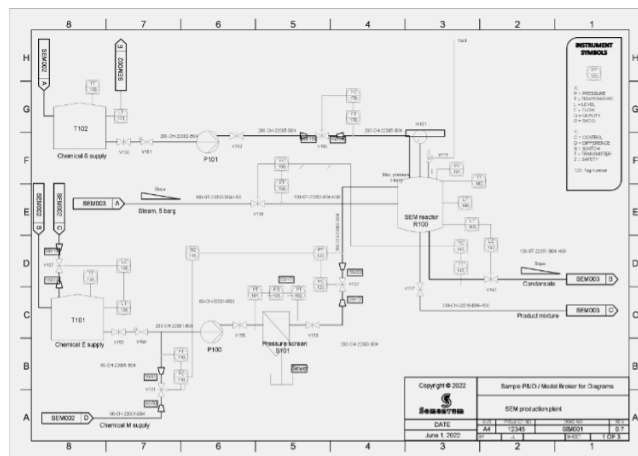


Figure 30: P&ID with replaced symbols and fixed layout.

While the results are promising, there are a few problematic cases that require explanation. In Figure 31 we see that vertical instrument lines and horizontal pipelines are no longer connected. This is caused by the DEXPI file missing required branch points at the instrument line/pipeline crossing, thus effectively the lines are not connected at all. Therefore, the layout algorithm does not handle the lines as connected and separates them. Additional problems caused by the source material are in Figure 32 and Figure 33, where duplicated graphics on the original PDF and missing label configuration cause elements to be included in the background of the diagram, and the layout fixing algorithm completely ignores the background.

The current solution lacks certain features necessary for achieving improved results. These encompass annotation handling, encompassing labels and flow arrows. Presently, component labels are translated concurrently with components, but potential overlaps are not verified. Similarly, line labels are relocated alongside lines, yet the existing implementation lacks a suitable mechanism for checking line equivalence in instances where line routing alters. Consequently, this may lead to erroneous translation of line labels.

Moreover, another constraint is the absence of support for terminal directions in the layout algorithm, potentially resulting in line overlaps with connected symbols.

It is believed that through additional development efforts, the aforementioned limitations can be addressed, culminating in a robust solution for P&ID digitization, inclusive of symbol replacement.

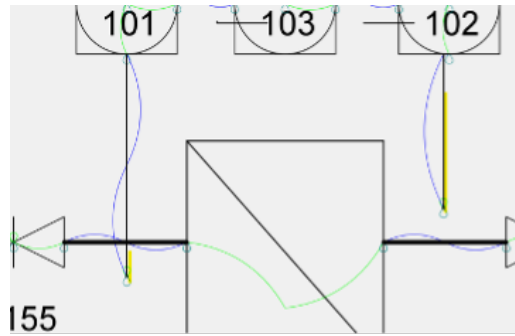


Figure 31: Vertical instrument lines on both sides of the pressure screen are not connected to horizontal pipelines. Logically the lines are not connected in the DEXPI file.

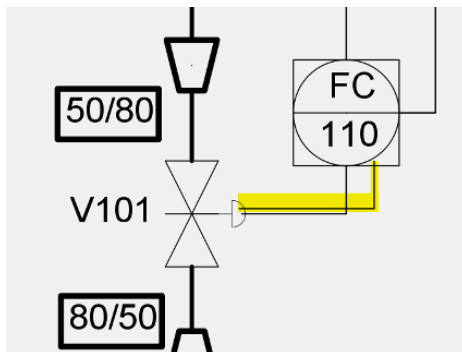


Figure 32: Secondary connection line is caused by having duplicated graphics in the source PDF.

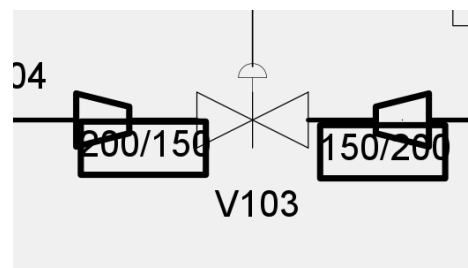


Figure 33: Reducer labels have not been attached to reducers and are part of the background.

5 Datasets Provision and Other Supporting Activities

This Section 5 introduces the initial efforts regarding the dataset provision process and support to the UC translation. The focus of the work has been on initiating dialogues with the use cases and other industrial partners to secure access to the necessary datasheets and devise a strategy to abstract the data structure and essential information required for translation without compromising industrial confidentiality.

5.1 Dataset categories and sources

Datasets play a crucial role in the development, training, and tuning of translators. Groups consider datasets from various perspectives, such as real vs synthetic, static vs dynamic, etc., which can be viewed as 'orthogonal' or 'complementary' categories. Further details about the specific nature of standards and related dataset format/structure are provided by WP3. The categories are summarized in the following diagram, and Table 4 outlines the key groups of dataset categories that can be utilized for these activities

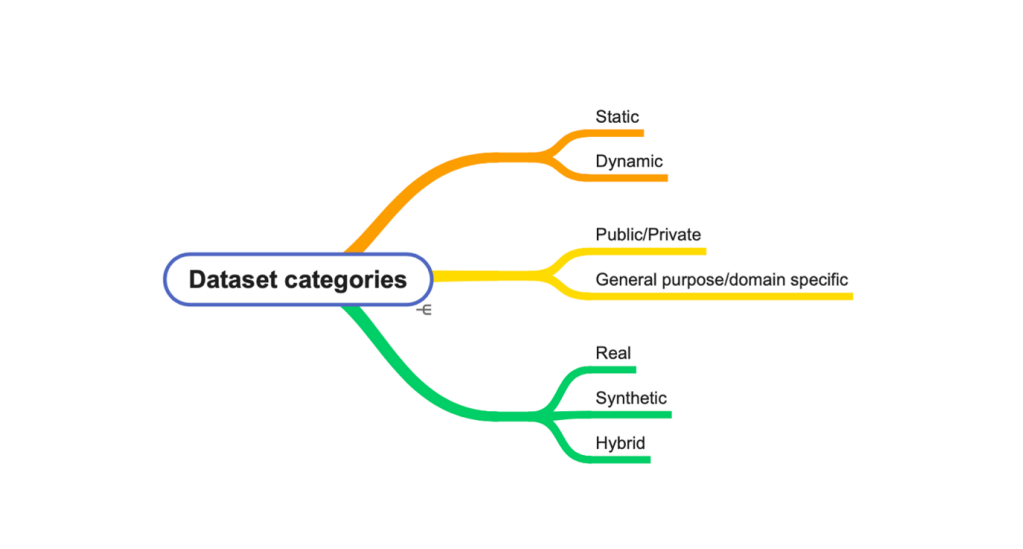


Figure 34: Dataset categories

The optimal dataset category (or combination of categories, considering that their natures are complementary) depends on several factors:

- Translation purpose: the nature of the translator and the standards to be addressed strongly influence the selection of datasets, as well as the use case.
- Domain specificity: in principle, the translator could be fully domain-independent, and at least the core translation algorithms are domain-independent. But depending on the nature of the translator the phases of model definition and training are strongly domain-dependent, therefore the engineering of the translator widely depends on the use case in which it will be used. The translator's effectiveness will greatly benefit from datasets originating directly from the domain in which they will be used, that is from the project use cases.
- Resource availability: available resources represent a constraint for the datasets' selection, e.g. budget, time, expertise for data collection and assessment, computing resources, etc.
- Model requirements: the model or, more widely, the technology adopted to develop the translator influences the selection of datasets, e.g. solutions based on machine learning have different necessities than model-based solutions.

By carefully considering these factors, we will be able to select the most appropriate dataset categories to train and set up a high-performing automatic translation system.

Table 4: Dataset Categories Description

Type	Description	Disadvantages	Advantages
Real datasets	Data collected from the real world, such as the internet, or from sensors, from databases, from logs, etc. They can be produced in real-time or collected in the past, and represent historical information. They can be raw or processed and can be generated by digital or physical twins.	May contain errors, or inconsistencies, or could be biased. Might require cleaning and pre-processing before use.	Captures the real usage of standards and reflects real-world use cases. Often available in real-time.
Synthetic	Data artificially generated using algorithms, heuristics, or methods based on artificial intelligence.	May not perfectly reflect real-world language usage. Requires expertise to generate high-quality synthetic data.	Can be tailored to specific requirements and domains. Allows for control over dataset size, details and quality.
Hybrid (real+synthetic)	Datasets resulting from the combination of real data and data artificially generated.	They share the disadvantages of the previous categories.	They share advantages over the previous categories. Allows to use of real incomplete or small datasets that are extended/complemented with artificial data.
Static	Data that remains relatively unchanged over time, e.g. historical datasets or synthetic datasets that are not generated in real-time.	Might not be able to capture all the nuances of standards that evolve with time or scenarios open to standards evolution. May not be suitable for domains with frequent standard updates.	Provide a solid foundation for understanding the core of the standard. Often well-structured and easy to process.
Dynamic	Data that is constantly changing and evolving, frequently in real-time or near real-time, e.g. dataset generated from sensors, time series, etc.	Can be noisy and difficult to clean. May require frequent updates to the model.	Captures the real use of a standard in the use cases. Useful for translators adopted in real-time applications.
Public, general purpose	Datasets freely available for anyone to download and use.	May have quality issues or copyright restrictions. Might not be domain-specific enough for specialized applications.	Cost-effective and readily accessible. Often large and diverse datasets are available. Sometimes, could be domain specific.
Use case specific, not public	Datasets adopted for a specific translator adopted in the use cases developed in WP6, WP7 and WP9.	Requires effort and resources to collect or create domain-specific data. May not be readily available for all the use cases in the project.	Improves translation accuracy and domain-specific standard usage. Ensures the model is familiar with the relevant concepts of standard.

5.2 Dataset selection and utilization process

The design and development of an autonomous translator make use of datasets, and their adoption can be streamlined with a structured process based on a set of common phases, circularly closing with a feedback loop that allows to adjust and refine the implementation of the algorithm, tune the model and/or the configuration parameters of the translator itself.

Figure 35 illustrates the preliminary high-level process that we have identified to select and utilize datasets for the design, development, testing, and assessment of the automatic translators.

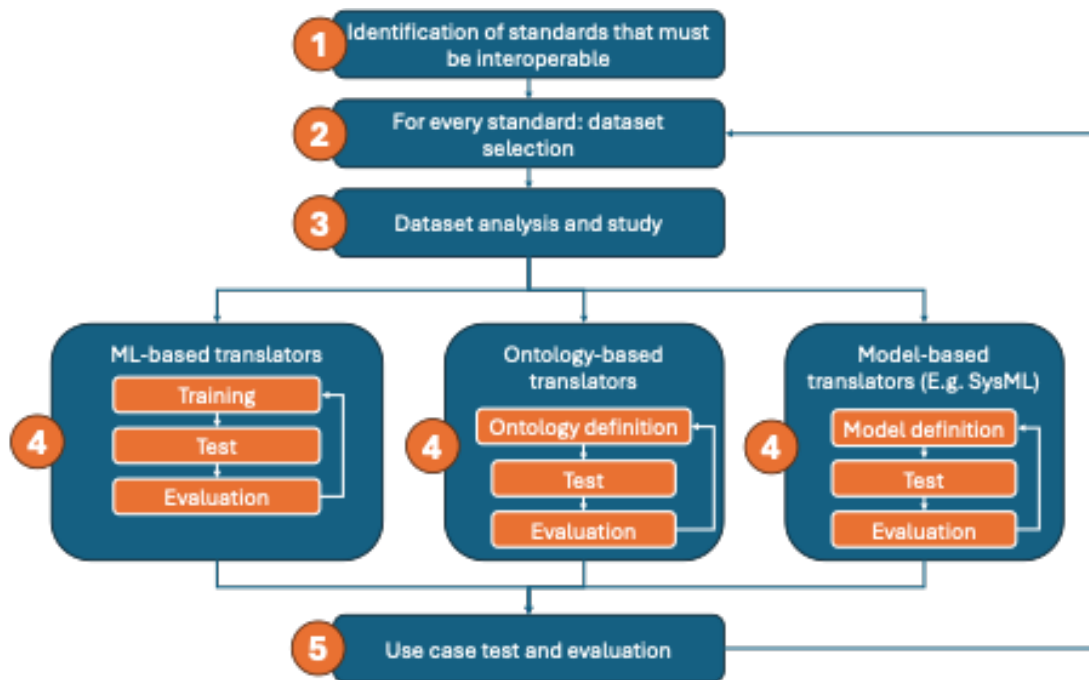


Figure 35: High-level identification and use process.

The first phase consists in the identification of the standards that must be made interoperable through the machine translation system. This phase might encompass factors like:

- Target audience: identifying the intended users (humans or machines) of the translation, whether it is for technical documentation, general communication, creative content, machine-to-machine interaction, etc.
- Domain specificity: specifying whether the translation needs to be tailored to a particular domain, such as automotive, aeronautics, electronics, legal, financial, etc. The requirements collected by the project use cases are fundamental in this phase of the project because they identify the domain in which the translator will operate.
- Quality metrics: defining the criteria for evaluating the quality of the translation is fundamental for the final step consisting of the assessment of the translation results in terms of the accuracy, quality, robustness, and adherence to the target standard.

The second step consists of the selection of one or more appropriate datasets for every standard that must be made automatically interoperable. Following the identification of standards, this stage involves selecting datasets to define and train the machine translation model. These datasets will likely consist of information that will be collected, processed and translated in the use cases and, depending on the nature of the use case and its objectives, will belong to one or more categories described in the previous section. The selection of the dataset is not static but depends on the final evaluation at the end of each development cycle: if the results of the translation do not satisfy the quality metrics, the motivation could be the model or the translation algorithm, but could also be the selection of the dataset that is not adequate for the specific standards to be translated. In this case, a new dataset must be selected, and a new development cycle starts.

Once the datasets are selected, the third phase consists of their analysis and study to ensure their quality and suitability for the subsequent phases. This may involve tasks like:

- Data cleaning, to eliminate errors, inconsistencies, and irrelevant information from the dataset.
- Data pre-processing, to format the dataset into a structure compatible with the machine translation model.
- Data augmentation, to artificially expand (when and if necessary) the dataset through techniques like e.g. synonym replacement or back-translation to improve model robustness.

Phase 4 of the process is focused on model definition, training, testing, and evaluation. This phase plays a crucial role in shaping the effectiveness of the translation system. While the core steps (model definition, training, and evaluation) remain similar across different translator categories, the specific details within this phase can vary. Here's a breakdown of how each category approaches model definition, training, and evaluation:

- **Machine Learning-based translators:** for this category of translators, this phase starts with the selection of the appropriate machine translation architecture, leveraging various machine learning algorithms, such as statistical machine translation (SMT) or neural machine translation (NMT), a recurrent neural network (RNN), a transformer-based model, etc. The chosen model is then fed with the training dataset to learn the intricacies of translating between the source and target standards. The model learns statistical patterns and relationships between language structures in both standards. Training focuses on optimizing the model's ability to predict the most likely target standard sequence for a given source standard sequence. The test and evaluation could rely on specific metrics, like BLEU score (measuring n-gram precision) or ROUGE score (assessing recall of n-grams), etc. and human evaluation might be used to assess the quality of translation, particularly for critical applications.
- **Ontology-based translators:** these translators leverage ontologies, which are structured representations of knowledge within a specific domain. The ontology definition is intended to generate a model that can learn the relationships between concepts and entities represented within the ontology. The dataset is used to generate and verify this knowledge, which enables the model to achieve a deeper understanding of the meaning and context beyond just "word-based" translations. In the test and evaluation phase, specific metrics are defined (BLEU solutions can still be adopted), but ontology-specific evaluation methods are adopted to assess how well the translation aligns with the conceptual structure and relationships within the domain ontology. This ensures the translation captures the intended meaning, ensures robustness in case of small deviations between domain-specific standards, and improves the interoperability level. The developer intervention is important in this case to assess the semantic accuracy and consistency of the translation within the specific domain in which the translator must operate.
- **Model-based translators:** this category of translators encompasses a very broad range of models, including potentially hybrid solutions that combine technology elements also from the other two categories. The model definition depends largely on the adopted model, and we cannot provide more details in this phase of the project. Just as an example, rule-based models might involve defining linguistic rules and patterns for translation. In the test and evaluation phase, metrics like the ones adopted in the previous categories are commonly used. Human evaluation plays a more prominent role for this category of translators, specifically when domain-specific standards are considered.

These three sub-phases of the process really depend on the nature of the translator and can also vary depending on the computational resources available for the training and model definition, on the desired level of control required in the translation, and on the specificities of the domain in which the translator will be adopted (for example, domain-specific evaluation metrics might be employed for specialized translations). In this phase, development, testing, and evaluation of results are performed in a laboratory environment and can rely on simulations, synthetic information, and automated testing and evaluation procedures.

The final phase consists in the test and evaluation of the translator in the real context of a real-world use case, potentially involving human evaluation to assess the translator's performance in a concrete scenario. Overall, phase 5 plays a vital role in bridging the gap between the controlled laboratory environment of phase 4 and the concrete application of an automatic translation system in the project use cases. This phase allows to test the translator's performance on the actual dataset in the use case and evaluation factors like the translation accuracy, quality, robustness, and adherence to the target standard. It also allows the identification of areas

for improvement for both the translator and the dataset selection that are addressed in phases 2 and 3 with a feedback loop, which might involve:

- Selecting new datasets.
- Fine-tuning the model, by further training the model on these additional datasets.
- Addressing specific translation errors.
- Enhancing domain-specific knowledge, for example when domain-specific accuracy is an issue.

5.3 Use-case support activities

Summary of the activities carried out in conjunction with some of the UC providers to support the alignment among translators and UC.

UC 7.1 Support The case study of 7.1 requires translators for the migration process of mechatronic projects designed with specific tools, extending from the planning phase to production launch. This migration necessitates the adaptation of data models according to the tools essential for the process.

During the specific step of migrating the digital design of electronic boards/controls from the electronic design tool to the PLM tool, specific data models have already been defined for both the output of the electronic design tool and the input required by the PLM. Now, the translation from one data model to the other needs to be executed.

DITAG tool has been selected to perform this translation, and we are currently assessing how to leverage this tool within the Arrowhead platform. This analysis aims to showcase and incentivize the management of data model interoperability.

In addition to preparing the data models, we have explored potential standards to facilitate this process. The IPC-2581 standard has emerged as a promising candidate, and further examination is proposed to integrate it into the digital reference and advocate for its adoption as the preferred data model.

Concurrently, an initial collaboration with Task 4.2 has been established to explore the application of AI translation in the UC. This initiative seeks to test various solutions, leveraging diverse approaches to offer interoperability solutions with translators based on the Arrowhead platform.

The case study of 7.1 contains two main subsystems: EC (Electronic Components) and PLM (Product Lifecycle Management). Each component applies its own, custom XML format. The EC's model is a rarely changing environment, but the PLM's model is under development, and the XML schema changes often. To grant interoperability between the subsystems, and to support the automatic mapping between the XML contents, Task 4.2 started to investigate the possible AI translation solutions.

As a first step, Task 4.2 investigated the shared example dataset that contained 170 EC - PLM XML pairs, and concluded the following:

- Each XML value regarding the PLM's content could be derived from the input, EC-related XML contents.
- Based on the initial dataset, a clear one-on-one association could be recognized between the EC – PLM XML contents.
- To support the automatic mapping procedure, a text-based definition could be added to each tag. This definition could support context recognition tasks.
- Since the PLM model's schema changes periodically, the automatic mapping solution could not work on training examples, because there will be no data pairs. The problem should be reduced to automatic mapping without training examples, using only text-based content definitions.

As a second step, Task 4.2 applied two GPT-based models:

- Microsoft Chat Copilot-based model, and
- Fine-tuned OpenAI GPT model.

The translation accuracy of the Chat Copilot-based model was unsatisfactory. The model generated more XML contents than expected, containing new, previously unseen XML tags. In contrast, the fine-tuned GPT model could reach 100% accuracy regarding the shared example dataset. It should be noted, that the example data pairs could also be easily translated with legacy algorithms.

As a next step, Task 4.2 will create new examples to try the translation mechanism without having training data, and applying the text-based definitions for context recognition and mapping.

UC 9.1 support A super-ontology-based data model translation approach tailored to the process industry (WP9) requirements has been investigated. The objective was to formulate an approach that discerns mapping necessities utilizing super ontology concepts, ultimately facilitating translations between disparate data modeling languages.

Throughout the initial year, emphasis has been placed on delineating mapping requisites for the use cases, albeit without active development efforts. Thus far, rule-based, metamodel, and data-based (ML) methodologies for identifying ontology-based mappings have surfaced as viable options. Moreover, data models utilized have been identified and earmarked as potential candidates for translation endeavors spanning years 2 and 3. However, a detailed analysis of these data models for this purpose is pending.

UC 9.2 Support Task 4.3 has been investigating workflows based on the SES use case related to sharing the LCA data across the value chain that includes suppliers and customers of SES. The workflow, illustrated in Figure 36, assumes the establishment of a DataSpace enabled by the Metadata Database.

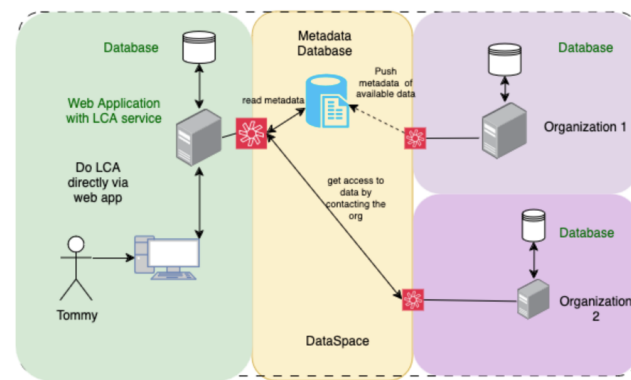


Figure 36: Translation Workflow

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. Dataspace connectors facilitate the transformation of data to the standard data format for the exchange. And they respond to requests from other organizations.

In this workflow model, the place for applying model-based translations is mostly allocated to the connectors. VTT connectors will be investigated as a possible implementation. The survey is planned to identify other technologies. Another candidate technology is Data Mesh.

UC 9.3 Support Preliminary tests have been conducted to serialize a process flow diagram, specifically a Balas simulator model, into various standard formats. The test utilized a simple example model, as depicted in Figure 37. For demonstration purposes, the serialization of Pump 1 and Stream S102 was executed in accordance with the DEXPI process format and IDO/PLM ontology. The parameters of Pump 1 and Stream S102 are elaborated upon in Figure 38. Selected portions of Pump 1 and Stream S102 are depicted in Figure 39, adhering to the DEXPI Process experimental XML schema [37], and in Figure 40, conforming to the IDO/PLM ontology in RDF Turtle format [38]. There are plans to expand this example case in the future.

Although a comprehensive mapping between Balas and DEXPI process information models has not yet been finalized, the demonstration was conducted to glean insights into the effectiveness of these standards. One challenge encountered in the mapping from Balas to DEXPI process, particularly to IDO/PLM, is the difficulty

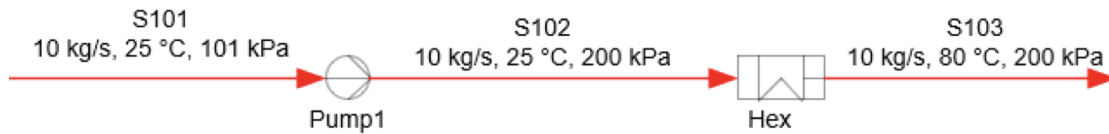
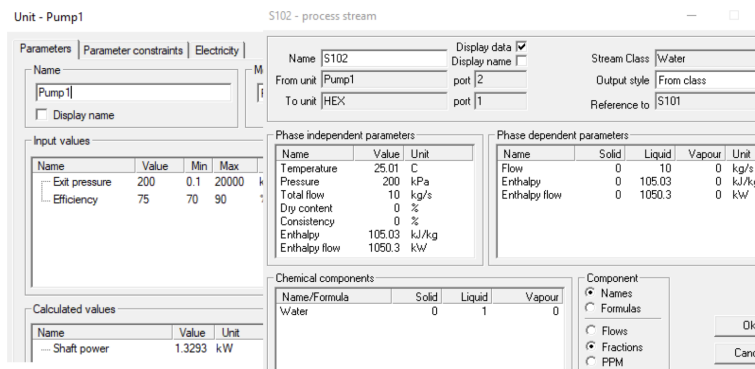


Figure 37: An example of Balas model consisting of a pump and a heater.

in identifying corresponding unit operation classifications, even with additional parameters, across different standards.

Moreover, serialization in accordance with the IDO/PLM ontology was undertaken due to the potential for an ontology to furnish a more expressive data format. It is envisaged that the integration of the DEXPI process into IDO will be pursued, augmenting it with pulp, board, and paper-specific concepts (addressing Requirements R1.6 and R1.7).



Name	Value	Min	Max	Unit
Exit pressure	200	0.1	20000	kPa
Efficiency	75	70	90	%

Name	Value	Unit
Temperature	25.01	C
Pressure	200	kPa
Total flow	10	kg/s
Dry content	0	%
Consistency	0	%
Enthalpy	105.03	kJ/kg
Enthalpy flow	1050.3	kJ/s

Name	Solid	Liquid	Vapour	Unit
Flow	0	10	0	kg/s
Enthalpy	0	105.03	0	kJ/kg
Enthalpy flow	0	1050.3	0	kJ/s

Name/Formula	Solid	Liquid	Vapour
Water	0	1	0

Name	Value	Unit
Shaft power	1.3293	kW

Figure 38: Parameters of Pump1 and stream S102.

In the evaluation of standard data formats (Objectives O1.2 and O1.5), a particular case-study requirement is to ensure that, alongside exporting Balas models in a standard format, models imported in the same standard format can be unequivocally re-imported into the Balas simulator. This necessity emphasizes the need for seamless interoperability, ensuring that data exported from the Balas simulator retains its integrity and can be accurately re-imported into the system without any ambiguity. This capability is crucial for maintaining consistency and efficiency in simulation processes.

```
<ProcessStep xsi:type="Pumping">
<Identifier>=Pump1</Identifier>
<MaterialPort>
  <Identifier>XL1</Identifier> <PortDirection>Inlet</PortDirection>
  <ConnectorIdentifier="S101"/>...
</MaterialPort>
<MaterialPort>
  <Identifier>XL2</Identifier> <PortDirection>Outlet</PortDirection>
  <ConnectorIdentifier="S102"/>
  <MaterialTemplateReference TemplateIdentifier="WaterTemplate"/>
</MaterialPort>...
</ProcessStep>
<ProcessConnection xsi:type="Stream">
<Identifier>S102</Identifier>
<Source BlockIdentifier="Pump1" PortIdentifier="XL2"></Source>
<Target BlockIdentifier="Hex" PortIdentifier="XL1"></Target>
<MaterialTemplateReference TemplateIdentifier="WaterTemplate"/>
<Temperature Mode="Design" Provenance="Specified" Range="Nominal">
  <Value>25.01</Value>
  <Unit>degC</Unit>...
</ProcessConnection>
<MaterialTemplate>
  <Identifier>WaterTemplate</Identifier>
  <NumberOfMaterialComponents>1</NumberOfMaterialComponents>
  <NumberOfPhases>1</NumberOfPhases>
  <ListOfMaterialComponents>
    <MaterialComponentIdentifier Identifier="H2O">
</ListOfMaterialComponents>
  <ListOfPhases>
    <PhaseIdentifier Identifier="Liquid"/>
</ListOfPhases>
</MaterialTemplate>
<MaterialComponent xsi:type="PureMaterialComponent">
  <Identifier>H2O</Identifier>...
</MaterialComponent>
```

Figure 39: DEXPI Process serialization.

```
@prefix ido: <http://rds.posccaesar.org/ontology/lis14/rdl/> .
@prefix rdl: <http://rds.posccaesar.org/ontology/plm/rdl/> .
@prefix balas: <http://example.org/balas/AH-demo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

balas:Pump1 a owl:NamedIndividual, rdl:PCA_10000011 , ... ;
ido:hasPart balas:Inlet_1_Pump1 , balas:Outlet_1_Pump1 ;
ido:hasParticipant balas:S102 , balas:S101 ;
ido:hasPhysicalQuantity balas:Pump1_Exit_Pressure .

balas:Outlet_1_Pump1 a owl:NamedIndividual, rdl:PCA_100005961 ;
ido:contains balas:S102 .

balas:S102 a owl:NamedIndividual, ido:Stream ;
ido:hasPart balas:H2O_Liquid ;
ido:hasPhysicalQuantity balas:S102_Temperature , ... ;
ido:containedBy balas:Outlet_1_Pump1 , balas:Inlet_1_Hex .

balas:S102_Temperature a owl:NamedIndividual, rdl:PCA_100003601 ;
ido:qualityQuantifiedAs balas:S102_Temp_Datum .

balas:S102_Temp_Datum a owl:NamedIndividual, ido:QualityDatum ;
ido:datumUOM rdl:PCA_100003672 ;
ido:datumValue "25.01"^^xsd:double .

balas:H2O_Liquid a owl:NamedIndividual, ido:Compound, ido:Phase .
```

Figure 40: Serialization according to IDO/PLM ontology.

6 Collaboration With Other WPs

WP4 has initiated 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. Several key actions have been undertaken:

- Analysis of the T3.1 survey results: This involved pre-selecting the data models per UC.
- Conducting individual meetings with each use case WP: These meetings aimed to confirm and discuss the data models employed per use case. Guiding questions were utilized to steer the discussions:
 - What standards are employed in each use case?
 - Where is interoperability between standards needed?
 - What datasets are available for each standard?
- Analyzing discussions with the use case providers and collaborating on addressing the various standards identified.
- Enhancing understanding of the standards and data models with support and guidance from WP3 experts.

This ongoing effort will persist throughout the project duration, ensuring close alignment between the technical packages and the use cases and demonstrators. Some preliminary results are summarized as follows.

Table 5: Relation between the use cases and the translation approaches.

WP	UC	Standards	Approach(es)
WP6	6.1	MDF4, ISO 15926, ISO 15118, DBC	Ontology-based translation (DITAG) and AI-based translation
WP7	7.1	ISO 10303-239/4000, IPC-2581	Ontology-based translation (DITAG) and AI-based translation
WP7	7.2	S5000F	Ontology-based translation
WP9	9.1	DEXPI, CFIHOS, ISO/TS 15926-4, ISO 10303-239/4000, ISO/CD 23726-3	Ontology-based translation (IDO) and Model-based translation
WP9	9.2	DEXPI, DEXPI+, ISO 10303-239/4000, IEC 61355	Ontology-based translation (IDO)
WP9	9.3	DEXPI, CFIHOS, ISO/TS 15926-4, ISO 10303-239/4000, ISO/CD 23726-3	Ontology-based translation (IDO) and Model-based translation

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. Such collaboration underscores the project's commitment to leveraging synergies across work packages to achieve overarching objectives effectively. Furthermore, Task 4.3 has started a collaboration with WP2 to advance the development of the Arrowhead profile [33] 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 Platform is under analysis. In fact, the Papyrus4Manufacturing [39] a variant of Papyrus is integrated with a similar concept **semanticID** inspired by the AAS standard [40] to identify referable elements.

7 Appendixes

- **Translation Solution Inventory** Initial version of the inventory.
- **OpenNMT Tutorial** presents a short tutorial about building a model.
- **ChatGPT API Tutorial** presents a short tutorial about using the ChatGPT API.
- **OpenAI fine-tuning Tutorial** presents a short tutorial about tuning OpenAI tool.

8 Conclusions

The document provides a report about the first generation of translation solutions. The deliverable includes a description of how the objectives are been reached by the end of the first year, the initial prototypes and proposals for the translation solutions, and any other support activity that helps to advance the state of the art and integrate the solutions into the use cases. It provides an overview of the translation divided into three categories: Ontology-based translation, AI-based translation and Model-based translation.

The document also describes the structure and process planned for the acquisition of datasets and examples of specific work done in collaboration with other WP. The Collaboration especially with WP3 and the UC has been identified as crucial for the success of the project and WP4 has focused on it.

The document answers the following objectives of WP4:

- 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

And provides insights into the the initial work to address the following objectives:

- 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

In conclusion, the work carried out by WP4 in the first year of the AfPVN Project aligned effectively with its objectives and plans.

9 References

- [1] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "SawSDL: Semantic annotations for WSDL and XML schema," *IEEE Internet Computing*, vol. 11, no. 6, pp. 60–67, 2007.
- [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] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-DL reasoner," *Journal of Web Semantics*, vol. 5, no. 2, pp. 51–53, 2007.
- [8] N. Guarino and C. Welty, "Identity and subsumption," in *The semantics of relationships: An interdisciplinary perspective*. Springer, 2002, pp. 111–126.
- [9] C. A. Hurtado, A. Poulouvasilis, and P. T. Wood, "Query relaxation in RDF," in *Journal on Data Semantics X*. Springer, 2008, pp. 31–61.
- [10] S. M. Rashid, J. P. McCusker, P. Pinheiro, M. P. Bax, H. O. Santos, J. A. Stingone, A. K. Das, and D. L. McGuinness, "The Semantic Data Dictionary – An Approach for Describing and Annotating Data," *Data Intelligence*, vol. 2, no. 4, pp. 443–486, 10 2020.
- [11] P. Pinheiro, H. Santos, Z. Liang, Y. Liu, S. M. Rashid, D. L. McGuinness, and M. P. Bax, "Hadatac: A framework for scientific data integration using ontologies," in *Proceedings of the ISWC*, 2018, p. 49.
- [12] 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, pp. 80–91.
- [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, 2009.
- [14] M. Dumontier, C. J. Baker, J. Baran, A. Callahan, L. Chepelev, J. Cruz-Toledo, N. R. Del Rio, G. Duck, L. I. Furlong, N. Keath *et al.*, "The SemanticScience Integrated Ontology (SIO) for Biomedical Research and Knowledge Discovery," *Journal of Biomedical Semantics*, vol. 5, no. 1, p. 14, 2014.
- [15] W. Fleming, "Overview of automotive sensors," *IEEE Sensors Journal*, vol. 1, no. 4, pp. 296–308, 2001.
- [16] X. Chen and G. Zhang, "Design of automotive gateway based on FlexRay," in *2011 International Conference on Electric Information and Control Engineering*, 2011, pp. 4897–4900.
- [17] F. Luo and P. Wei, "A design of CAN, LIN bus test board," in *2019 IEEE 19th International Conference on Communication Technology (ICCT)*, 2019, pp. 1142–1146.

- [18] S. Woo, H. J. Jo, I. S. Kim, and D. H. Lee, "A practical security architecture for in-vehicle can-fd," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2248–2261, 2016.
- [19] G. Cena and A. Valenzano, "A protocol for automatic node discovery in canopen networks," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 3, pp. 419–430, 2003.
- [20] F.-L. Lian, J. Moyne, and D. Tilbury, "Performance evaluation of control networks: Ethernet, controlnet, and devicenet," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 66–83, 2001.
- [21] H. Kopetz, W. Elmenreich, and C. Mack, "A comparison of lin and ttp/a," in *2000 IEEE International Workshop on Factory Communication Systems. Proceedings (Cat. No.00TH8531)*, 2000, pp. 99–107.
- [22] D. Soni and A. Makwana, "A survey on mqtt: a protocol of internet of things (iot)," in *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, vol. 20, 2017, pp. 173–177.
- [23] (Accessed 2024) References (aitia). [Online]. Available: <https://en.wikipedia.org/wiki/Temperature>
- [24] (Accessed 2024) Humidity - wikipedia. [Online]. Available: <https://en.wikipedia.org/wiki/Humidity>
- [25] T. Tothfalusi, E. Varga, Z. Csiszar, and P. Varga, "MI-based translation methods for protocols and data formats," in *3rd International Workshop on Analytics for Service and Application Management, Canada*, 2023.
- [26] [Online]. Available: <https://eclipse.dev/papyrus/>
- [27] OMG, "OMG Systems Modeling Language (OMG SysML™)," OMG Doc Nb formal/19-11-01,(<https://www.omg.org/spec/SysML/1.6>), 2019.
- [28] F. Rekik, S. Dhouib, and Q.-D. Nguyen, "Bridging the Gap between SysML and OPC UA Information Models for Industry 4.0," *The Journal of Object Technology*, vol. 22, no. 2, pp. 1–15, 2023. [Online]. Available: <https://cea.hal.science/cea-04169475>
- [29] [Online]. Available: <https://dexpi.org/wp-content/uploads/2020/09/DEXPI-PID-Specification-1.3.pdf>
- [30] [Online]. Available: <https://www.jip36-cfihos.org/cfihos-standards/>
- [31] OMG, "OMG Systems Modeling Language (OMG SysML™)," OMG Doc Nb ptc/2023-06-02,(<https://www.omg.org/spec/SysML/2.0/Language/>), 2023.
- [32] [Online]. Available: <https://www.omg.org/spec/SysML/2.0/Beta1/Transformation/PDF>
- [33] [Online]. Available: <https://github.com/asmaasmaoui/profile-library-sysml>
- [34] [Online]. Available: <https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation/>
- [35] [Online]. Available: <https://plantuml.com/en/>
- [36] [Online]. Available: <https://mbse-syson.org/>
- [37] D. Cameron, W. Otten, H. Temmen, and G. Tolksdorf, *DEXPI Process Specification, Release 1.0*, 1st ed., 2023.
- [38] World Wide Web Consortium (W3C), "Rdf 1.1 turtle - terse rdf triple language," WWW Document, 2014, accessed 4.4.24. [Online]. Available: <https://www.w3.org/TR/2014/REC-turtle-20140225/>
- [39] [Online]. Available: <https://youtu.be/p4gTzzc95hw?si=OhCZp061WdM0cTLg>
- [40] "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

10 Revision History

10.1 Contributing and reviewing partners

Contributions	Reviews	Participants	Representing Partner
X	X	Cristina Paniagua	LTU
X	X	Filipe Moutinho	UNINOVA
X	X	Asma Smaoui	CEA
X	X	Paolo Azzoni	ETH
X		Paulo Pinheiro	PIAGET
X		Oskar Wintercorn	LTU
X		Tamas Tothfalusi	AITIA
X		David Rutqvist	SIN
X		Géza Kulcsár	IQL
X		Vahid Tavakkoli	KLU
X		Gerardo Santillan	SEM
X		Miren Illarramendi	SEA
X		Teemu Mätäsniemi	VTT
X		Luis Lino Ferreira	ISEP
X		Valeriy Vyatkin	AALT
X		David Hästbacka	TAU
X		Luca Martorano	LEO
X		João Rosas	UNINOVA

10.2 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2024-03-20	0.1	Creation of the draft and structure of the document	Cristina Paniagua
2	2024-04-05	0.2	Introduction and Objectives Sections	Cristina Paniagua
3	2024-04-16	0.3	Darft 1st generation of translation solutions	WP4 partners
4	2024-04-17	0.4	Collaboration with other WP Section	Cristina Paniagua
5	2024-04-24	0.5	Refinement and update of the text	Cristina Paniagua
6	2024-05-03	0.6	Final update of the text	WP4 partners
7	2024-05-07	0.7	Dataset acquisition Section	Paolo Azzoni
8	2024-05-13	0.8	Review and correction	Filipe Moutinho
9	2024-05-14	1.0	Final version	Cristina Paniagua

10.3 Quality Assurance

No.	Date	Version	Approved by
1	2024-05-21	1.2	Jerker Delsing

WP 4 – Automated data model translations	
Deliverable 4.1	M12
Deliverable 4.2	M18
Deliverable 4.3	M30

Tool/System Name	Description	Tasks	Leader Partner	Other partners	Initial/Preliminary Working Prototype at /M12	Open Source (Y/N)
DITAG - Data Interoperability Translators Automatic Generator tool	This tool receives as input a reference ontology and the schemas (XML and/or JSON schemas) of the provider and consumer systems. The tool then checks whether these systems can interoperate, and if so, it automatically generates a translator.	4.1	UNINOVA		M12	Y
SEMVAR.CC	SEMVAR.CC system is derived from the open-source Human-Aware Data Acquisition Framework (HADatAc). HADatAc is a schema-free, evolutionary, scalable and provenance-aware infrastructure for managing data and metadata content from multiple scientific studies.	4.1	PIAGET	UNINOVA	M18	Y
The Industrial Data Ontology (IDO)	Novel approach to modeling industrial installations, utilizing ontologies aligned with the semantic web and the Industrial Data Ontology (IDO) to promote semantic interoperability.	4.1	LTU	WP3	M30	Y
Broker Translation	Solution capable of translating and redirecting messages between brokers, allowing for multiple devices of any protocol to communicate in a high-performance environment with advanced configurability, while guaranteeing the delivery QoS requirements.	4.1	ISEP		M18	Y
Service Inventory	The “Service Inventory” module is a new Arrowhead Core System module to extract information and to create labels based on text-related information (e.g., text-based data, XML, JSON).	4.2	AITIA		M18	Y

NLP Translation	AI translator based on Word2Vec models using text files.	4.2	AITIA		M12	Y
Papyrus	Papyrus is an environment for editing any kind of EMF model, particularly supporting UML 2 (Unified Modeling Language) and related modeling languages such as SysML (System Modeling Language) and MARTE (Modeling and Analysis of Real-Time and Embedded systems). Papyrus also offers very advanced support for UML profiles that enables users to define editors for DSLs (Domain Specific Languages) based on the UML 2 standard. Papyrus is a collection of plug-ins and features on top of the Eclipse Modeling Framework	4.3	CEA		M12	Y
SysML v1.6-v2 Translation	Translation from models using SysML1.6 to SysML2.0.	4.3	CEA	IQL, AITIA		Y
P&Ids Translation	Model Broker is a commercial background software product of Semantum. There are several different variations of the product, but most relevant variations for the Arrowhead fPVN project are Model Broker for Diagrams and Model Broker for Dynamic Simulation. Using Model Broker for Diagrams user can translate legacy technical drawings like P&IDs, logic diagrams, function block diagrams and electrical diagrams into intelligent machine understandable formats. Model Broker for Dynamic Simulation uses these intelligent data formats to translate the engineering models into dynamic process simulation models.	4.3	SEM		M12	N
Deep-Learning Based Meta-Modeling for Data Translation	Through the utilization of ontology learning and Graph Neural Networks (GNNs), this conceptual approach holds the promise of revolutionizing data translation practices. Enhancing interoperability and communication across diverse data models.	4.3	KLU		M30	Y

OpenNMT tutorial

1. Open command prompt and navigate to working directory
2. Use the following commands for installation:
 - pip install OpenNMT-py
 - git clone <https://github.com/yamoslem/MT-Preparation.git>
 - pip install -r MT-Preparation/requirements.txt
3. Paste the source and target files to the working directory
4. -python MT-Preparation/train_dev_split/train_dev_test_split.py <number of validation samples> <number of test samples> source.txt target.txt
5. Create config file with running the following script(documentation for parameters: <https://opennmt.net/OpenNMT-py/options/train.html>):

```
# Create the YAML configuration file
# On a regular machine, you can create it manually or with nano
# Note here we are using some smaller values because the dataset is small
# For larger datasets, consider increasing: train_steps, valid_steps, warmup_steps,
save_checkpoint_steps, keep_checkpoint
```

```
config = ""# config.yaml
```

```
## Where the samples will be written
save_data: run
```

```
# Training files
data:
  corpus_1:
    path_src: XmlSource.xml.subword.train
    path_tgt: JSONSource.json.subword.train
    transforms: [filtertoolong]
  valid:
    path_src: UN.en-fr.fr-filtered.fr.subword.dev
    path_tgt: JSONSource.json.subword.dev
    transforms: [filtertoolong]
```

```
# Vocabulary files, generated by onmt_build_vocab
src_vocab: run/source.vocab
tgt_vocab: run/target.vocab
```

```
# Vocabulary size - should be the same as in sentence piece
src_vocab_size: 50000
tgt_vocab_size: 50000
```

```
# Filter out source/target longer than n if [filtertoolong] enabled
src_seq_length: 150
tgt_seq_length: 150
```



```
# Tokenization options
src_subword_model: source.model
tgt_subword_model: target.model

# Where to save the log file and the output models/checkpoints
log_file: train.log
save_model: models/model.fren

# Stop training if it does not improve after n validations
early_stopping: 4

# Default: 5000 - Save a model checkpoint for each n
save_checkpoint_steps: 1000

# To save space, limit checkpoints to last n
# keep_checkpoint: 3

seed: 3435

# Default: 100000 - Train the model to max n steps
# Increase to 200000 or more for large datasets
# For fine-tuning, add up the required steps to the original steps
train_steps: 3000

# Default: 10000 - Run validation after n steps
valid_steps: 1000

# Default: 4000 - for large datasets, try up to 8000
warmup_steps: 1000
report_every: 100

# Number of GPUs, and IDs of GPUs
world_size: 1
gpu_ranks: []

# Batching
bucket_size: 262144
num_workers: 0 # Default: 2, set to 0 when RAM out of memory
batch_type: "tokens"
batch_size: 4096 # Tokens per batch, change when CUDA out of memory
valid_batch_size: 2048
max_generator_batches: 2
accum_count: [4]
accum_steps: [0]

# Optimization
model_dtype: "fp16"
```

```
optim: "adam"
learning_rate: 2
# warmup_steps: 8000
decay_method: "noam"
adam_beta2: 0.998
max_grad_norm: 0
label_smoothing: 0.1
param_init: 0
param_init_glorot: true
normalization: "tokens"

# Model
encoder_type: transformer
decoder_type: transformer
position_encoding: true
enc_layers: 6
dec_layers: 6
heads: 8
hidden_size: 512
word_vec_size: 512
transformer_ff: 2048
dropout_steps: [0]
dropout: [0.1]
attention_dropout: [0.1]
'''
```

```
with open("config.yaml", "w+") as config_yaml:
    config_yaml.write(config)
```

6. Get number of available threads with `nproc -all`
7. `onmt_build_vocab -config config.yaml -n_sample -1 -num_threads <number of available threads>`
8. `-onmt_train -config config.yaml`

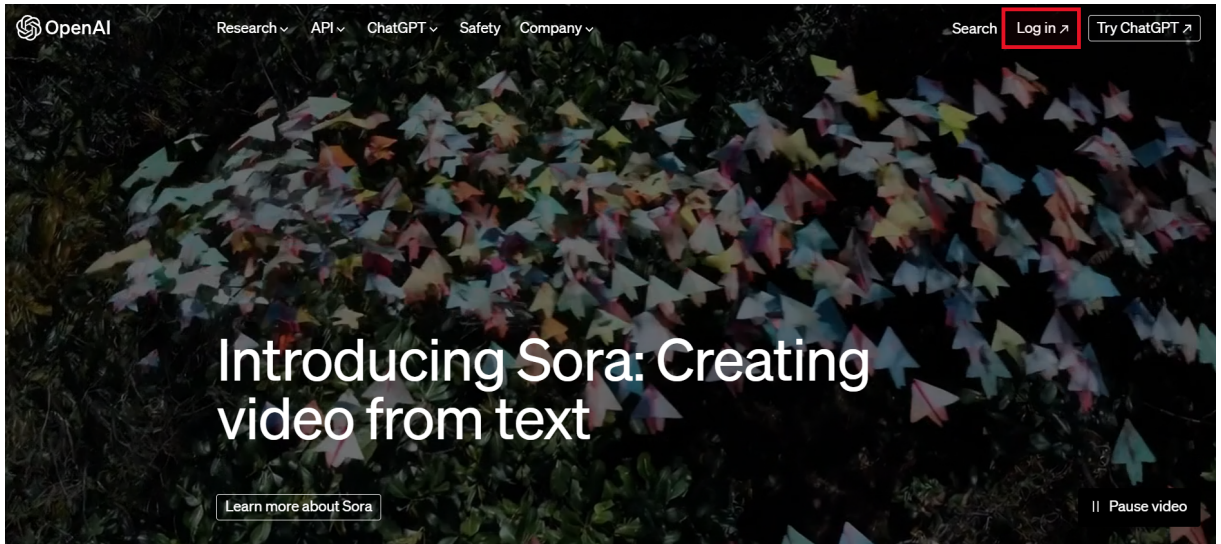
Example for training with example dataset provided:

<https://github.com/yamoslem/OpenNMT-Tutorial>

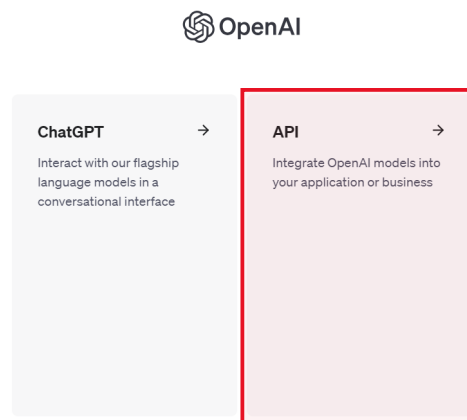
A short tutorial regarding the ChatGPT API

ChatGPT API setup:

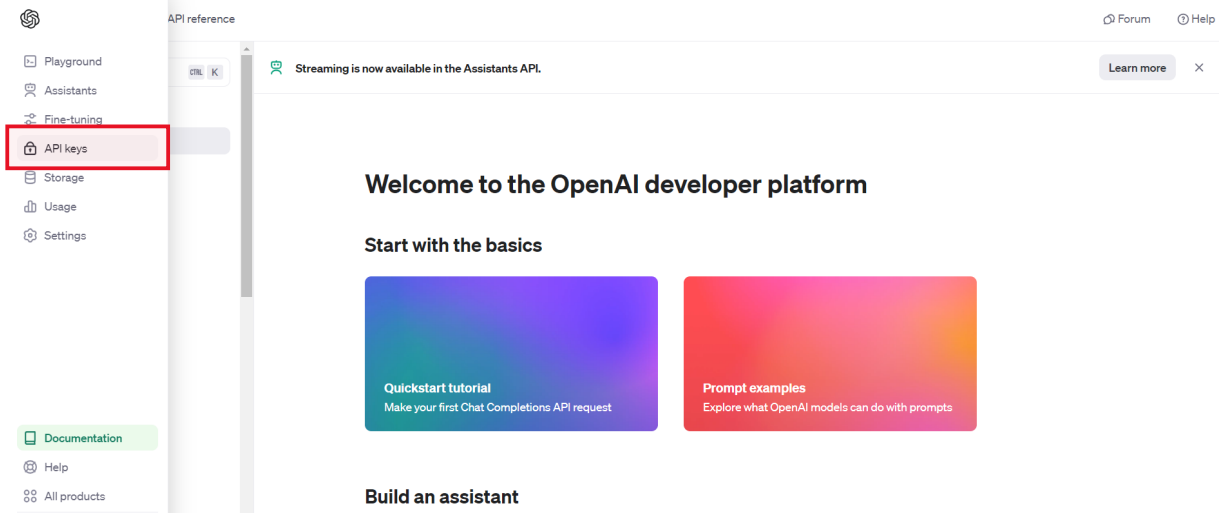
1. Go to OpenAI.com and select Log in



2. After logging in select API



3. Go to API keys







4. Create new secret key

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically disable any API key that we've found has leaked publicly.

Enable tracking to see usage per API key on the [Usage page](#).

NAME	SECRET KEY	TRACKING	CREATED	LAST USED	PERMISSIONS	
Secret key	sk-...aWjB	+ Enable	2023. jún. 26.	2023. jún. 26.	All	 
python	sk-...1F2L	+ Enable	2023. szept. 11.	2024. márc. 18.	All	 

[+ Create new secret key](#)

Default organization

If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.

Personal

Note: You can also specify which organization to use for each API request. See [Authentication](#) to learn more.

5. Copy your secret key and paste in your python code for the value of `openai.api_key`
6. In python, install the openai library (`pip install openai`)

Further setup information:

<https://platform.openai.com/docs/quickstart?context=python>

Billing

To be able to use the API you have to have enough balance on your OpenAI account. It will subtract the money from that account after each API call. Information on pricing: <https://openai.com/pricing>

To add money to your account:

1. Go to <https://platform.openai.com/account/billing/overview>
2. Add a payment method, then click „Add credit to balance”

OpenAI chat completion

Example request with all the possible parameters set to some value:

```
from openai import OpenAI
client = OpenAI(
    api_key=os.environ.get("CUSTOM_ENV_NAME")
)
completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Hello!"}
    ]
    #Number between -2.0 and 2.0.
    #Positive values penalize new tokens based on their existing
    #frequency in the text so far, decreasing the model's likelihood
    #to repeat the same line verbatim.

    frequency_penalty=0.5,

    #Modify the likelihood of specified tokens appearing in the
    #completion. Accepts a JSON object that maps tokens (specified
    #by their token ID in the tokenizer) to an associated bias value
    #from -100 to 100. Mathematically, the bias is added to the logits
    #generated by the model prior to sampling. The exact effect will
    #vary per model, but values between -1 and 1 should decrease or
    #increase likelihood of selection; values like -100 or 100 should
    #result in a ban or exclusive selection of the relevant token.

    #logit_bias,

    #Whether to return log probabilities of the output tokens or not.
    #If true, returns the log probabilities of each output token returned
    #in the content of message. This option is currently not available on
    #the gpt-4-vision-preview model.

    logprobs=False,

    #An integer between 0 and 20 specifying the number of most likely tokens
    #to return at each token position, each with an associated log
    probability.
    #logprobs must be set to true if this parameter is used.

    #top_logprobs=5,

    #Maximum number of tokens generated

    max_tokens=500,

    #How many chat completion choices to generate for each input message.
    #Note that you will be charged based on the number of generated tokens
    #across all of the choices. Keep n as 1 to minimize costs.
```

n=1,

#Number between -2.0 and 2.0. Positive values penalize new tokens based
#on whether they appear in the text so far, increasing the model's
#likelihood to talk about new topics.

presence_penalty = 0,

#An object specifying the format that the model must output.
#Compatible with GPT-4 Turbo and all GPT-3.5 Turbo models newer
#than gpt-3.5-turbo-1106. Setting to { "type": "json_object" }
#enables JSON mode, which guarantees the message the model generates
#is valid JSON.

#response_format = { "type": "json_object" },

#If specified, our system will make a best effort to sample
#deterministically, such that repeated requests with the same
#seed and parameters should return the same result.
#Determinism is not guaranteed, and you should refer to the
#system_fingerprint response parameter to monitor changes in the backend.

seed = 1,

#Up to 4 sequences where the API will stop generating further tokens.

stop = ["Best wishes", "Yours faithfully"],

#If set, partial message deltas will be sent, like in ChatGPT.
#Tokens will be sent as data-only server-sent events as they become
#available, with the stream terminated by a data: [DONE] message.

stream = False,

#What sampling temperature to use, between 0 and 2. Higher values
#like 0.8 will make the output more random, while lower values like
#0.2 will make it more focused and deterministic.

temperature = 0.7,

#An alternative to sampling with temperature, called nucleus sampling,
#where the model considers the results of the tokens with top_p
#probability mass. So 0.1 means only the tokens comprising the top
#10% probability mass are considered.

top_p = 1,

#A list of tools the model may call. Currently, only functions are

```

#supported as a tool. Use this to provide a list of functions the
#model may generate JSON inputs for. A max of 128 functions are
supported.

#tools

#Controls which (if any) function is called by the model. none means the
#model will not call a function and instead generates a message. auto
means
#the model can pick between generating a message or calling a function.
#Specifying a particular function via
#{ "type": "function", "function": { "name": "my_function" } } forces the
model
#to call that function.

#tool_choice

#A unique identifier representing your end-user, which can help OpenAI
toú
#monitor and detect abuse.

#user=user1

)

print(completion.choices[0].message)

```

Further information and examples: <https://platform.openai.com/docs/api-reference/chat/create>

Example response object:

```

{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "created": 1677652288,
  "model": "gpt-3.5-turbo-0125",
  "system_fingerprint": "fp_44709d6fcb",
  "choices": [{
    "index": 0,
    "message": {
      "role": "assistant",
      "content": "\n\nHello there, how may I assist you today?",
    },
    "logprobs": null,
    "finish_reason": "stop"
  }],
  "usage": {

```



```
"prompt_tokens": 9,  
"completion_tokens": 12,  
"total_tokens": 21  
}  
}
```

Further information and examples: <https://platform.openai.com/docs/api-reference/chat/object>

Function used for translation:

```
def openAIResponse(messageToTranslate, prompt, temp):  
    openai.api_key = ""  
    try:  
        openai.ChatCompletion.timeout = 1  
        response = openai.ChatCompletion.create(  
            model="gpt-3.5-turbo",  
            messages=[  
                {  
                    "role": "system",  
                    "content": prompt  
                },  
                {  
                    "role": "user",  
                    "content": messageToTranslate  
                }  
            ],  
            temperature=temp,  
            max_tokens=256,  
            top_p=1,  
            frequency_penalty=0,  
            presence_penalty=0  
        )  
    except:  
        print("Error")  
        return "Error"  
    return response
```

Example usage:

```
messageToTranslate = "XML message"  
prompt = "Translate the following XML message to JSON. Only give the  
translation, nothing else."  
temp = 0.5  
response = openAIResponse(messageToTranslate, prompt, temp)
```

ChatGPT Fine tuning tutorial

The example files that are used in this example (finetune.jsonl and finetunetest.jsonl) are available here:

<https://github.com/Aitia-IIOT/ah-ai-translation-poc/tree/master/OpenAIFineTuningExamples>

Install the openai API:

```
!pip install openai
```

Create jsonl file (finetune.jsonl in our example) with structure of:

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already."}]}
```

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"}, {"role": "assistant", "content": "Oh, just some guy named William Shakespeare. Ever heard of him?"}]}
```

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "How far is the Moon from Earth?"}, {"role": "assistant", "content": "Around 384,400 kilometers. Give or take a few, like that really matters."}]}
```

Create an openai file object. For this you need to set your API key, that you can generate at <https://platform.openai.com/api-keys> after logging in.

```
import openai

openai.api_key = "your-api-key"

openai.files.create(
    file=open("/content/finetune.jsonl", "rb"),
    purpose="fine-tune"
)
```

Code to check if your files are formatted correctly, and to estimate token number (code from openai finetuning documentation):

```
!pip install tiktoken
```

```
import json
import tiktoken # for token counting
import numpy as np
from collections import defaultdict

data_path = "/content/finetune.jsonl"

# Load the dataset
with open(data_path, 'r', encoding='utf-8') as f:
    dataset = [json.loads(line) for line in f]

# Initial dataset stats
print("Num examples:", len(dataset))
print("First example:")
for message in dataset[0]["messages"]:
    print(message)

# Format error checks
format_errors = defaultdict(int)

for ex in dataset:
    if not isinstance(ex, dict):
        format_errors["data_type"] += 1
        continue

    messages = ex.get("messages", None)
    if not messages:
        format_errors["missing_messages_list"] += 1
        continue

    for message in messages:
        if "role" not in message or "content" not in message:
            format_errors["message_missing_key"] += 1

        if any(k not in ("role", "content", "name", "function_call",
"weight") for k in message):
            format_errors["message_unrecognized_key"] += 1

        if message.get("role", None) not in ("system", "user",
"assistant", "function"):
            format_errors["unrecognized_role"] += 1

    content = message.get("content", None)
    function_call = message.get("function_call", None)
```

```

        if (not content and not function_call) or not
isinstance(content, str):
            format_errors["missing_content"] += 1

        if not any(message.get("role", None) == "assistant" for message in
messages):
            format_errors["example_missing_assistant_message"] += 1

if format_errors:
    print("Found errors:")
    for k, v in format_errors.items():
        print(f"{k}: {v}")
else:
    print("No errors found")

encoding = tiktoken.get_encoding("cl100k_base")

# not exact!
# simplified from https://github.com/openai/openai-
cookbook/blob/main/examples/How_to_count_tokens_with_tiktoken.ipynb
def num_tokens_from_messages(messages, tokens_per_message=3,
tokens_per_name=1):
    num_tokens = 0
    for message in messages:
        num_tokens += tokens_per_message
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":
                num_tokens += tokens_per_name
    num_tokens += 3
    return num_tokens

def num_assistant_tokens_from_messages(messages):
    num_tokens = 0
    for message in messages:
        if message["role"] == "assistant":
            num_tokens += len(encoding.encode(message["content"]))
    return num_tokens

def print_distribution(values, name):
    print(f"\n#### Distribution of {name}:")
    print(f"min / max: {min(values)}, {max(values)}")
    print(f"mean / median: {np.mean(values)}, {np.median(values)}")
    print(f"p5 / p95: {np.quantile(values, 0.1)}, {np.quantile(values,
0.9)}")

# Warnings and tokens counts
n_missing_system = 0
n_missing_user = 0

```

```

n_messages = []
convo_lens = []
assistant_message_lens = []

for ex in dataset:
    messages = ex["messages"]
    if not any(message["role"] == "system" for message in messages):
        n_missing_system += 1
    if not any(message["role"] == "user" for message in messages):
        n_missing_user += 1
    n_messages.append(len(messages))
    convo_lens.append(num_tokens_from_messages(messages))
    assistant_message_lens.append(num_assistant_tokens_from_messages(messages))

print("Num examples missing system message:", n_missing_system)
print("Num examples missing user message:", n_missing_user)
print_distribution(n_messages, "num_messages_per_example")
print_distribution(convo_lens, "num_total_tokens_per_example")
print_distribution(assistant_message_lens,
"num_assistant_tokens_per_example")
n_too_long = sum(1 > 4096 for l in convo_lens)
print(f"\n{n_too_long} examples may be over the 4096 token limit, they
will be truncated during fine-tuning")

# Pricing and default n_epochs estimate
MAX_TOKENS_PER_EXAMPLE = 4096

TARGET_EPOCHS = 3
MIN_TARGET_EXAMPLES = 100
MAX_TARGET_EXAMPLES = 100
MIN_DEFAULT_EPOCHS = 1
MAX_DEFAULT_EPOCHS = 25

n_epochs = TARGET_EPOCHS
n_train_examples = len(dataset)
if n_train_examples * TARGET_EPOCHS < MIN_TARGET_EXAMPLES:
    n_epochs = min(MAX_DEFAULT_EPOCHS, MIN_TARGET_EXAMPLES //
n_train_examples)
elif n_train_examples * TARGET_EPOCHS > MAX_TARGET_EXAMPLES:
    n_epochs = max(MIN_DEFAULT_EPOCHS, MAX_TARGET_EXAMPLES //
n_train_examples)

n_billing_tokens_in_dataset = sum(min(MAX_TOKENS_PER_EXAMPLE, length)
for length in convo_lens)
print(f"Dataset has ~{n_billing_tokens_in_dataset} tokens that will be
charged for during training")
print(f"By default, you'll train for {n_epochs} epochs on this
dataset")

```

```
print(f"By default, you'll be charged for ~{n_epochs *  
n_billing_tokens_in_dataset} tokens")
```

You can estimate the cost of fine-tuning based of the number of tokens and the number of epochs with the help of the [openai pricing page](#). With the `finetune.jsonl` file that is provided, the training cost under 4 dollars with ~150.000 tokens trained for 3 epochs.

To create a finetuning job the id of the previously created fileobject is needed. Code to create finetuning job:

```
openai.fine_tuning.jobs.create(  
    training_file="file-id",  
    model="gpt-3.5-turbo"  
)
```

The state of the finetuning process can be retrived using the finetuning job's id:

```
openai.fine_tuning.jobs.retrieve("ftjob-CQylw8kuISu0iKVNGQbf9NoQ")
```

After the finetuning job is finished the retrieval will have the name of the fine tuned model in the `fine_tuned_model` attribute. Using this we can use the model to generate a chat completion. For temperature I used 0, to make the translation as precise and as close to the original translation as possible.

```
from openai import OpenAI  
  
client = OpenAI(api_key = "api-key")  
  
completion = client.chat.completions.create(  
    model="fine_tuned_model name",  
    messages= [  
        {"role": "system", "content": "You are a translator, that  
translates the output XML files of a program to input XML messages of  
another one."},  
        {"role": "user", "content": "<row>\n    <EPPL Family>11  
THERMISTORS</EPPL Family>\n    <EPPL Group>03 TEMPERATURE SENSOR</EPPL  
Group>\n    <Quality type>F</Quality type>\n    <Description>Thermistor  
4KOhm 4K3A354 (-55C TO +115C) ESCC  
4006/13</Description>\n    <SPN>E11001002.F01</SPN>\n    <Code Eng  
COM>E1140S0000001</Code Eng COM>\n    <Code Old  
COM>E1140S0000001MSB</Code Old COM>\n    <Code Old  
SEM>\n    <ASA></ASA>\n    </Code Old SEM>\n    <Code Old  
SENET></Code Old SENET>\n    <Code generic>No</Code
```

```

generic>\n      <Component Number>400601304B</Component
Number>\n      <Commercial Part Number>4K3A354</Commercial Part
Number>\n      <Generic Part Type></Generic Part Type>\n      <Engineering
Package></Engineering
Package>\n      <Package>WIRED</Package>\n      <Finish></Finish>\n      <Gen
eric specification>ESCC 4006</Generic specification>\n      <Detail
specification>ESCC 4006/013 ISS 8</Detail specification>\n      <Quality
Level>ESCC</Quality Level>\n      <Quality status>ESCC QPL</Quality
status>\n      <ESD Sensitivity level></ESD Sensitivity
level>\n      <Moisture sensitivity level></Moisture sensitivity
level>\n      <Comment></Comment>\n      <Weight></Weight>\n      <Manufactur
er>Measurement Specialities (IR)</Manufacturer>\n      <PPL
EBOM>Preferred</PPL EBOM>\n      <PPL MBOM>Preferred</PPL
MBOM>\n      <Link files></Link files>\n      <Library Ref>11-THER-
THERMISTOR</Library Ref>\n      <Sim Model Name></Sim Model
Name>\n      <Sim File></Sim File>\n      <Sim Kind></Sim Kind>\n      <Sim
SubKind></Sim SubKind>\n      <Sim Netlist></Sim Netlist>\n      <Sim Spice
Prefix></Sim Spice Prefix>\n      <Sim Port Map></Sim Port
Map>\n      <Assembly qual status></Assembly qual status>\n      <Value>4
K</Value>\n      <Tol></Tol>\n      <TCR>0</TCR>\n      <Volt>0</Volt>\n      <
Footprint1></Footprint1>\n      <Footprint2></Footprint2>\n      <Footprint
3></Footprint3>\n      <Application>COM-SPACE-
SMT</Application>\n      </row>"}

],
temperature = 0
)
print(completion.choices[0].message)

```

Results on test data:

1.example:

Output:

```

<row>\n      <Type></Type>\n      <Name></Name>\n      <Revision></Revision>\n      <Ti
tle></Title>\n      <Description>Thermistor 4K0hm 4K3A354 (-
55C TO +115C) ESCC 4006/13</Description>\n      <Owner></Owner>\n      <MaturitySta
te></MaturityState>\n      <Family>11 THERMISTORS</Family>\n      <Subfamily>03 TEM
PERATURE SENSOR</Subfamily>\n      <Quality>E</Quality>\n      <Manufacturer>Measur
ement Specialities (IR)</Manufacturer>\n      <EIN>E11001002.E01</EIN>\n      <Symb
ol>11-THER-
THERMISTOR</Symbol>\n      <Package>\n      <Footprint></Footprint>\n      <Geome
tricPackage>WIRED</GeometricPackage>\n      </Package>\n      <Code Eng COM>E1140S0
000001</Code Eng COM>\n      <Code Old COM>E1140S0000001MSP</Code Old COM>\n      <
Component Number>400601304P</Component Number>\n      <Commercial Part Number>4K3
A354</Commercial Part Number>\n      <Value>4 K</Value>\n      <Tol></Tol>\n      <TC
R>0</TCR>\n      <Volt>0</Volt>\n      <Creation Date></Creation Date>\n      <Modifi
cation Date></Modification Date>\n      </row>

```

Expected output:


```

<row>\n    <Type></Type>\n    <Name></Name>\n    <Revision></Revision>\n    <Title></Title>\n    <Description>Thermistor 4KOhm 4K3A354 (-55C TO +115C) ESCC 4006/13</Description>\n    <Owner></Owner>\n    <MaturityState></MaturityState>\n    <Family>11 THERMISTORS</Family>\n    <Subfamily>03 TEMPERATURE SENSOR</Subfamily>\n    <Quality>E</Quality>\n    <Manufacturer>Measurement Specialities (IR)</Manufacturer>\n    <EIN>E11001002.E01</EIN>\n    <Symbol>11-THER-THERMISTOR</Symbol>\n    <Package>\n    <Footprint></Footprint>\n    <GeometricPackage>WIRED</GeometricPackage>\n    </Package>\n    <Code Eng COM>E1140S000001</Code Eng COM>\n    <Code Old COM>E1140S000001MSP</Code Old COM>\n    <Component Number>400601304P</Component Number>\n    <Commercial Part Number>4K3A354</Commercial Part Number>\n    <Value>4 K</Value>\n    <Tol></Tol>\n    <TCR>0</TCR>\n    <Volt>0</Volt>\n    <Creation Date></Creation Date>\n    <Modification Date></Modification Date>\n </row>

```

2.example:

Output:

```

<row>\n    <Type></Type>\n    <Name></Name>\n    <Revision></Revision>\n    <Title></Title>\n    <Description>Thermistor 4KOhm 4K3A354 (-55C TO +115C) ESCC 4006/13</Description>\n    <Owner></Owner>\n    <MaturityState></MaturityState>\n    <Family>11 THERMISTORS</Family>\n    <Subfamily>03 TEMPERATURE SENSOR</Subfamily>\n    <Quality>F</Quality>\n    <Manufacturer>Measurement Specialities (IR)</Manufacturer>\n    <EIN>E11001002.F01</EIN>\n    <Symbol>11-THER-THERMISTOR</Symbol>\n    <Package>\n    <Footprint></Footprint>\n    <GeometricPackage>WIRED</GeometricPackage>\n    </Package>\n    <Code Eng COM>E1140S000001</Code Eng COM>\n    <Code Old COM>E1140S000001MSB</Code Old COM>\n    <Component Number>400601304B</Component Number>\n    <Commercial Part Number>4K3A354</Commercial Part Number>\n    <Value>4 K</Value>\n    <Tol></Tol>\n    <TCR>0</TCR>\n    <Volt>0</Volt>\n    <Creation Date></Creation Date>\n    <Modification Date></Modification Date>\n </row>

```

Expected output:

```

<row>\n    <Type></Type>\n    <Name></Name>\n    <Revision></Revision>\n    <Title></Title>\n    <Description>Thermistor 4KOhm 4K3A354 (-55C TO +115C) ESCC 4006/13</Description>\n    <Owner></Owner>\n    <MaturityState></MaturityState>\n    <Family>11 THERMISTORS</Family>\n    <Subfamily>03 TEMPERATURE SENSOR</Subfamily>\n    <Quality>F</Quality>\n    <Manufacturer>Measurement Specialities (IR)</Manufacturer>\n    <EIN>E11001002.F01</EIN>\n    <Symbol>11-THER-THERMISTOR</Symbol>\n    <Package>\n    <Footprint></Footprint>\n    <GeometricPackage>WIRED</GeometricPackage>\n    </Package>\n    <Code Eng COM>E1140S000001</Code Eng COM>\n    <Code Old COM>E1140S000001MSB</Code Old COM>\n    <Component Number>400601304B</Component Number>\n    <Commercial Part Number>4K3A354</Commercial Part Number>\n    <Value>4 K</Value>\n    <Tol></Tol>\n    <TCR>0</TCR>\n    <Volt>0</Volt>\n    <Creation Date></Creation Date>\n    <Modification Date></Modification Date>\n </row>

```