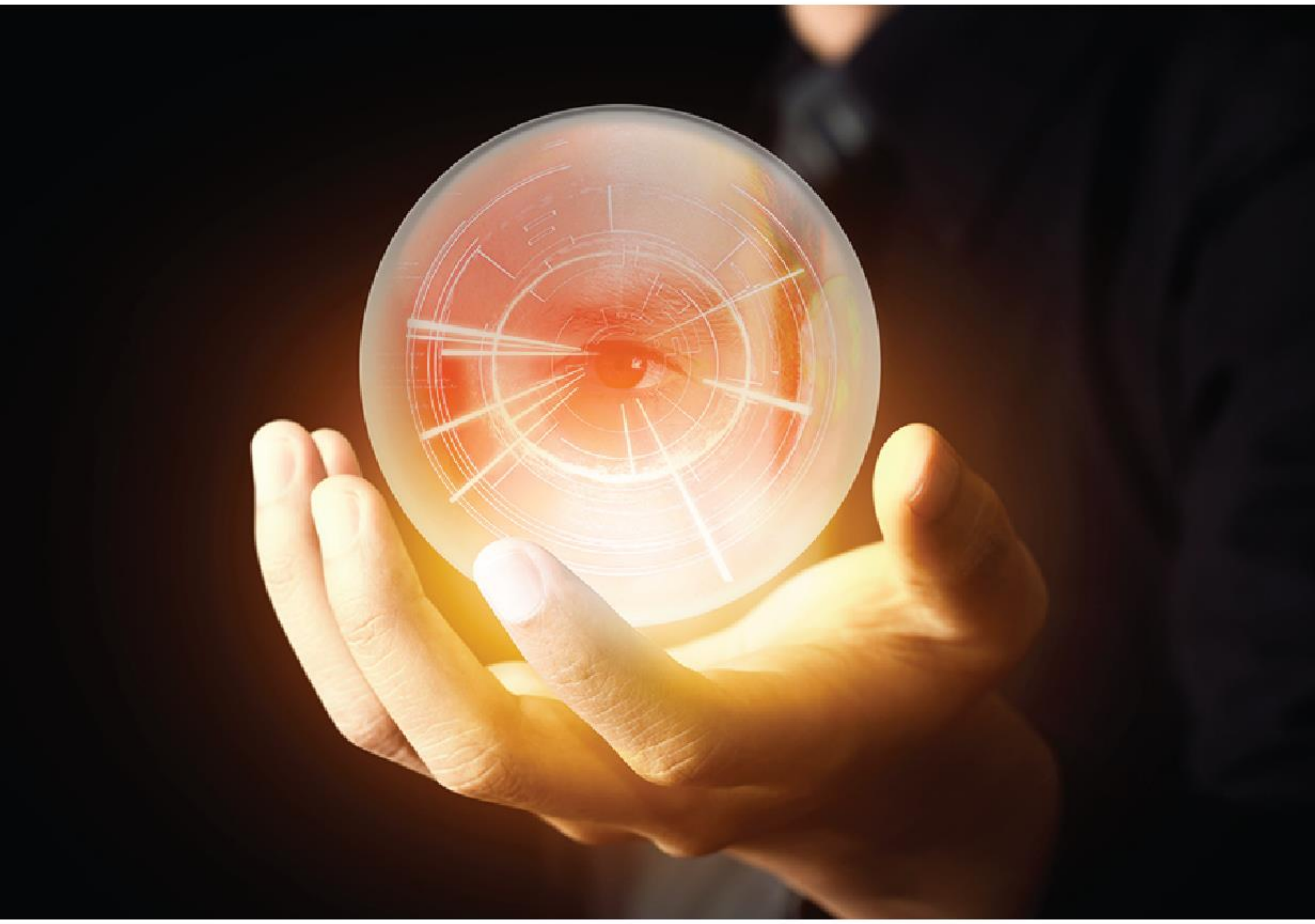
PROPHESY

Platform for rapid deployment of self-configuring and optimized predictive maintenance services



DELIVERABLE

D3.7 – PROPHESY-CPS Middleware Infrastructure v1

Horizon 2020
European Union funding
for Research & Innovation

The PROPHESY initiative has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 766994.

Project Acronym:	PROPHESY
Grant Agreement number:	766994 (H2020-IND-CE-2016-17/H2020-FOF-2017)
Project Full Title:	Platform for rapid deployment of self-configuring and optimized predictive maintenance services
Project Coordinator:	INTRASOFT International SA

This project is co-funded by the European Union

DELIVERABLE

D3.7 – PROPHESY-CPS Middleware Infrastructure v1

Dissemination level	PU – Public
Type of Document	(R) Report / (DEM) Demonstrator
Contractual date of delivery	M14, 30/11/2018
Deliverable Leader	SENSAP
Status - version, date	Final, v1.0, 30/11/2018
WP / Task responsible	MODRAGON/SENSAP
Keywords:	Data Streaming, Streaming Analytics, Middleware Infrastructure, Asset Administration Shell

Executive Summary

This report is part of deliverable D3.7 "PROPHESY-CPS Middleware Infrastructure v1" and illustrates the prototype implementation of the CPS infrastructure which supports data streaming and analytics in PROPHESY-CPS. It also accompanies a set of software packages that realize this implementation. Deliverable D3.7 is therefore devoted to the detailed specification and prototype implementation of the CPS middleware infrastructure, which follows principles of the PROPHESY-CPS architecture (D3.1) and adheres to the specification of the deliverables of WP2.

In particular, starting from the PROPHESY-CPS Detailed Architecture, a more detailed specification of the middleware infrastructure is provided in terms of the following functionalities: streaming analytics in local (CPS) and cloud (PdM) level and communication between them. These functionalities are provided by two subsystems: Asset Administration Shell (AAS) and Data Streaming subsystem. These two components are designed in a way that are capable to provide analytics in both CPS and platform (PdM) level.

The AAS links to the communication of CPS with the upper layer such as PROPHESYH-PdM platform, by providing a standard mechanism for exposing the CPS functionality. Furthermore, AAS is based on the I4.0 component concept. The I4.0 component can be defined as a Cyber Physical System (CPS) in the context of the RAMI4.0. AAS is the virtual representation of the asset. The AAS is composed by a *Manifest* and a *Component Manager*. The *Manifest* aggregates information about identification, data and functionalities exposed while the *Component Manager* is responsible to administer the models within the Manifest as well as to link the information in the AAS to both physical and cyber worlds. In such a way, the *Component Manager* includes two interfaces, namely a *northbound interface* (NBI) and a *southbound interface* (SBI). The former provides a uniform Application Programming Interface (API) that receives and send data in a strict and coherent format.

On the other hand, the Data Streaming subsystem links to the data streaming analytics both in PROPHESY-CPS and PROPHESY-PdM level. Data streaming consists of three components: "*Streaming Gateway*", "*Message Translating & Routing Service*" and finally "*CPS/Asset Registry*". *Asset/CPS Registry* provides a discoverability mechanism internal the CPS for the available streams and analytics. Furthermore, in platform level is enhanced with CPS discoverability functionalities. *Streaming Gateway* provides a communication Interface with AAS in order to receive external requests for data streaming or commands. Finally, *Message Translating and Routing Service,* is responsible for creating data flows from the data sources to analytics processors, in order to support data analytics in CPS level. The Message Translating and Routing Service provides also cloud analytics in PdM level. The only difference in platform level is that data streams are provided by AAS.

Note that the present version of the deliverable represents the first release of the CPS middleware infrastructure, while a second and final release will be provided as part of D3.8. The final release will provide a complete implementation AAS and Data Streaming

subsystems, including relevant tools. It will also incorporate feedback received during the integration of the prototypes with other components, as well as feedback from the use of the middleware infrastructure in the PROPHESY use cases implementation.

Deliverable Leader:	SENSAP
Contributors:	ART, NOVA FCT, NOVA ID, UNPARALLEL
Reviewers:	AIT, INTRA
Approved by:	INTRA

Document History			
Version	**Date**	**Contributor(s)**	**Description**
V0.1	12/10/2018	SENSAP	Initial structure of the document
V0.2	25/10/2018	NOVA ID	Chapter 2
V0.3	1/11/2018	UNPARALLEL	Chapter 4
V0.4	8/11/2018	SENSAP	Chapter 3
V0.5	10/11/2018	NOVA ID	Chapter 6
V0.6	15/11/2018	SENSAP	Chapter 5
V0.7	24/11/2018	NOVA ID	Contribution in chapter 3,4
V0.75	26/11/2018	UNPARALLEL	Contribution in chapter 2, 6
V0.8	26/11/2018	SENSAP	Contribution in chapter 3, 6
V0.9	27/11/2018	ART, UNPARALLEL, NOVA ID, SENSAP	Comments
V0.95	28/11/2018	AIT, INTRA	Review
V1.0	30/11/2018	SENSAP	Final version

Table of Contents

Table of Figures

List of Tables

Definitions, Acronyms and Abbreviations

Acronym/ Abbreviation	Title
AAS	Asset Administration Shell
AM	Analytics orchestrator Manifest
API	Application Programming Interface
AR	Augmented Reality
CAEX	Computer Aided Engineering eXchange
CPS	Cyber Physical System
CRUD	Create, Read, Update and Delete
DDA	Distributed Data Analytics
DoA	Description of Action
DSM	Data Source Manifest
EAE	Edge Analytics Engine
ECI	Edge Computing Infrastructure
HF-ML	High Frequency Machine Learning
HTTP	Hyper Text Transfer Protocol
KPI	Key Performance Indicators
LF-ML	Low Frequency Machine Learning
MOM	Message-Oriented Middleware
NBI	North-Bound Interface
OEE	Overall Equipment Effectiveness
OOP	Object Oriented Programming
OPC	Open Platform Communications
OPC-UA	OPC Unified Architecture
PM	Processor Manifest
PO	Processor Orchestrator
RAMI	Reference Architectural Model Industrie
REST	Representational State Transfer
RPC	Remote Procedure Call
RUL	Remaining Useful Life
SBI	South-Bound Interface
SODA	Service-Oriented Device Architecture
URI	Unique Resource Identifiers
VoIP	Voice over IP

1 Introduction

1.1 PROPHESY

1.1.1 The PROPHESY Vision

Despite the proclaimed benefits of predictive maintenance (PdM), the majority of manufacturers are still disposing with preventive and condition-based maintenance approaches, which result in suboptimal OEE (Overall Equipment Effectiveness). This is mainly due to the challenges of predictive maintenance deployments, including the fragmentation of the various maintenance related datasets (i.e. data "silos"), the lack of solutions that combine multiple sensing modalities for maintenance based on advanced predictive analytics, the fact that early predictive maintenance solutions do not close the loop to the production as part of an integrated approach, the limited exploitation of advanced training and visualization modalities for predictive maintenance (such as the use of Augmented Reality (AR) technologies), as well as the lack of validated business models for the deployment of predictive maintenance solutions to the benefit of all stakeholders. The main goal of PROPHESY is to lower the deployment barriers for advanced and intelligence predictive maintenance solutions, through developing and validating (in factories) novel technologies that address the above-listed challenges.

In order *to alleviate the fragmentation of datasets and to close the loop to the field,* PROPHESY will specify a novel CPS (Cyber Physical System) platform for predictive maintenance, which shall provide the means for *diverse data collection, consolidation and interoperability*, while at the same time supporting digital automation functions that will *close the loop to the field and will enable "autonomous" maintenance functionalities*. The project's CPS platform is conveniently called PROPHESY-CPS and is developed in the scope of WP3 of the project.

In order *to exploit multiple sensing modalities for timely and accurate predictions of maintenance parameters (e.g., RUL (Remaining Useful Life))*, PROPHESY will employ advanced predictive analytics which shall operate over data collected from multiple sensors, machines, devices, enterprise systems and maintenance-related databases (e.g., asset management databases). Moreover, PROPHESY will provide tools that will facilitate the development and deployment of its library of advanced analytics algorithms. The analytics tools and techniques of the project will be bundled together in a toolbox that is coined *PROPHESY-ML* and is developed in WP4 of the project.

In order *to leverage the benefits of advanced training and visualization for maintenance,* including *increased efficiency and safety of human-in-the-loop processes* the project will take advantage of an Augmented Reality (AR) platform. The AR platform will be customized for use in maintenance scenarios with particular emphasis on remote maintenance. It will be also combined with a number of visualization technologies such as ergonomic dashboards, as a means of enhancing worker's support and safety. The project's AR platform is conveniently called *PROPHESY-AR.*

In order *to develop and validate viable business models for predictive maintenance deployments,* the project will explore optimal deployment of configurations of turn-key solutions, notably solutions that comprise multiple components and technologies of the PROPHESY project (e.g., data collection, data analytics, data visualization and AR components in an integrated solution). The project will *provide the means for evaluating such configurations against various business and maintenance criteria,* based on corresponding, relevant KPIs (Key Performance Indicators). PROPHESY's tools for developing and evaluating alternative deployment configurations form the project service optimization engine, which we call *PROPHESY-SOE.*

1.1.2 PROPHESY WP3 Overview

PROPHESY's WP3 is devoted to the implementation and delivery of the PROPHESY-CPS platform, which will be based on the MANTIS[1] reference architecture and be optimized for Predictive Maintenance (PdM) services. The implementation is driven by the relevant specifications produced in WP2. The main objectives of WP3 are:

- to adapt and instantiate the MANTIS architecture, based on PROPHESY requirements and specifications,
- to specify digital models and techniques for sharing and interoperability of diverse maintenance datasets, which reside in different systems and data stores,
- to specify and implement sensors for data collection, including mechanisms for flexibly accommodating different types of PdM sensing modalities (such as vibration, oil analysis, imaging, acoustics etc.),
- to provide an implementation of the PROPHESY-CPS platform, including edge devices, data streaming analytics infrastructures and integration with the cloud,
- to ensure that the PROPHESY-CPS guarantees security and data protection across all levels of a PdM solution,
- to provide techniques for maintenance-driven processes, through closing the loop from the maintenance insights to the shop floor, including capabilities for self-configuring and self-adaptive processes.

1.2 Purpose

Task 3.4 is devoted to the implementation and delivery of the middleware infrastructure that will support data streaming and analytics in PROPHESY-CPS. Furthermore, this infrastructure should enable analytics both at the edge and at the cloud layers of the PROPHESY-CPS solutions.

This document is a living document, which means it will continuously evolve during the two iterations of the task. It also covers currently known open issues, which might be elaborated in the next version of this document.

[1] MANTIS. Cyber Physical System based Proactive Collaborative Maintenance. A project funded by the ECSEL Joint Undertaking under grant agreement No 662189. 2015-2018. http://www.mantis-project.eu/

1.3 Document structure

The document structure is as follows:

- Chapter 1 provides an **Introduction** of the deliverable scope and structure. Also defines the role of this task to WP3.
- Chapter 2 provides the **Background Information** which are taken into account to the design and implementation of CPS Middleware infrastructure. This chapter includes the connection with specification in D2.1 and available technologies and standards that are relevant to middleware infrastructure.
- Chapter 3 is an **Overview of PROPHESY-CPS Middleware Infrastructure Elements.** It provides an architecture of the middleware infrastructure which is responsible for data streaming in CPS and cloud level.
- Chapter 4 focuses on describing the structure and functionality of **Asset Administration Shell** sub-system. A detailed view of components is presented.
- Chapter 5 analyses the **Data Streaming** sub-system which is responsible for the data streaming and analytics.
- Chapter 6 presents a set of validating **Use Cases** for the middleware infrastructure i.e. proof-of-concept use cases that can be supported by the middleware infrastructure.

2 Background Information

2.1 PROPHESY-CPS Logical Architecture

The Figure 1, provides an overview of the PROPHESY-CPS logical architecture, i.e. the core architectural components and the interactions between them.

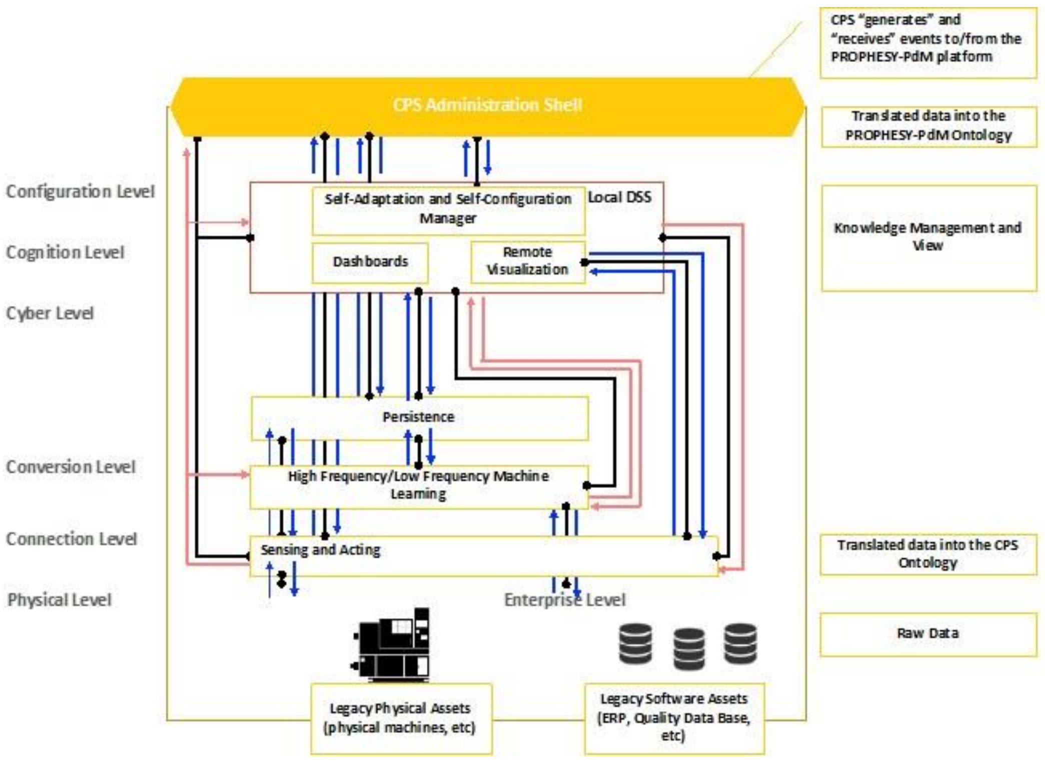

CPS Administration Shell
Configuration Level
Cognition Level
Cyber Level
Conversion Level
Connection Level
Physical Level
Self-Adaptation and Self-Configuration Manager
Local DSS
Dashboards
Remote Visualization
Persistence
High Frequency/Low Frequency Machine Learning
Sensing and Acting
Enterprise Level
Legacy Physical Assets (physical machines, etc)
Legacy Software Assets (ERP, Quality Data Base, etc)
CPS "generates" and "receives" events to/from the PROPHESY-PdM platform
Translated data into the PROPHESY-PdM Ontology
Knowledge Management and View
Translated data into the CPS Ontology
Raw Data

Figure 1: PROPHESY-CPS Logical Architecture [/ref deliverable d2.1]

The PROPHESY-CPS – as described in deliverable D2.1 PROPHESY-CPS Specifications v1 – is the necessary component to enable the implementation of Predictive Maintenance solutions within the manufacturing company. It facilitates the data extraction, collection, transforming, loading and analysis – from the physical world – to identify behavioural patterns of the physical asset in order to predict relevant maintenance events.

The PROPHESY-CPS logical architecture only identify the main functional components and their relationships, however further details need to be added and discussed in order to move from a functional description to a detailed description that can be used by developers during the implementation task.

2.2 Supporting Technologies, frameworks, standards and implementations

2.2.1 REST-Web Services

Representational State Transfer (REST) is an architectural style and model that relies on the web standards such using Hyper Text Transfer Protocol (HTTP) verbs and Unique Resource Identifiers (URIs). The following principles are the foundation of any REST implementation:

- All the resources are identified by URIs;
- All the resources can have multiple representations;
- All the resources can be accessed/modified/created/deleted by standard HTTP methods;
- The connections between client and server are stateless.

The design and implementation of the components based on the REST architectural style follows the following necessary step:

1. Identification of the resource URIs (see Table 1);
2. Identification of the HTTP methods for Create, Read, Update and Delete (CRUD) operations supported by the resource (see Table 2); and
3. Identify the different representations supported by the resource.

Table 1: Sample URI and related description

URI	URI Description
/xxx/yyy/zzz	*Describe the resource represented by the URI*

Table 2: HTTP methods and action description on the resource

HTTP Method	URI	Action Description
GET **POST** **DELETE** **PUT**	/xxx/yyy/zzz	*Describe the action taken on the resource represented by the URI*

2.2.2 Integration Middleware

2.2.2.1 Node Red

Node-RED is a programming tool that is used “wiring” together hardware devices, APIs and online services in graphical and, thus, very intuitive way. It provides a browser-based editor that delivers all the necessary mechanisms for “wiring” together components and services (also called nodes) while creating execution flows. These flows can be – then – deployed and executed on the top of node.js.

The usage of node.js as execution runtime delivers to node-red event-driven, non-blocking capabilities (inherited from the node.js model) while assuring its execution on low-cost single-board platforms. Furthermore, node-red relies on a very active and vibrant community where nodes and flows are constantly uploaded and shared by node-red developers allowing to extend more and more the node-red capabilities and functionalities.

2.2.2.2 Apache Kafka

The usage of message-oriented middleware (MOM) has become more and more crucial for designing, building and implementing Service Oriented Architectures. In this landscape, several technologies and solutions have been developed that support sending/receiving messages between distributed systems, i.e. between distributed application modules that typically run on heterogenous hardware and software platforms such as ActiveMQ[2], RabbitMQ[3], Apache Kafka[4], etc.

In particular Apache Kafka is a distributed messaging system written and used by LinkedIn for web page processing it provides all the necessary mechanisms for designing and developing distributed streaming platforms and delivers higher performances (in terms of speed and memory footprint) when compared with Active and RabbitMQ based-solutions (see Figure 2 and Figure 3). By taking into account the previous experience of the research and industrial partners – involved in the development of the PROHESY-PdM system – as well as the higher performances, apache Kafka technology will be adopted for delivering streaming capabilities in both PROPHESY-CPS and PROPHESY-PdM platform.

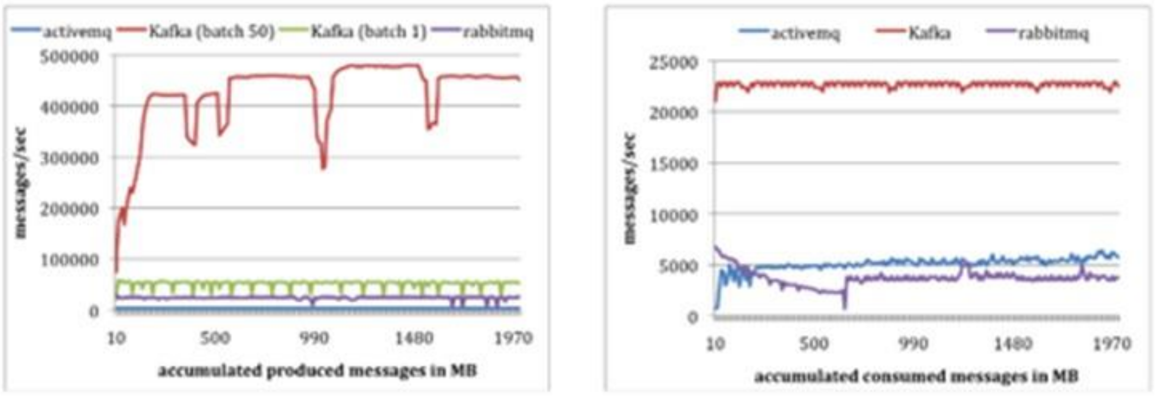


Figure 2: Producer/Consumer performance comparison [1]

[2] http://activemq.apache.org

[3] http://www.rabbitmq.com

[4] https://kafka.apache.org

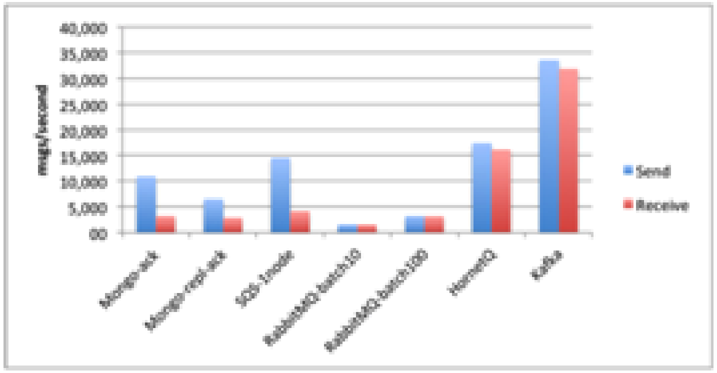


Figure 3: Throughput performance comparison [2]

2.2.3 Data Representation and Exchange

2.2.3.1 IEC 62714 AutomationML[5]

AutomationML (Automation Markup Language) is another XML-based, neutral and open data format that enables storage and exchange of plant engineering information. AutomationML alleviates the heterogeneity of the various state-of-the-art engineering tools, which are used in different areas of industrial automation.

AutomationML is based on a series of other standards and formats, including: (i) the previously described CAEX for plant topology representation in an hierarchical manner; (ii) the COLLADA format for representing geometrical information, graphical attributes and kinematics in the 3D space; and (iii) industrial automation logic implemented based on the PLCopen XML format, which provides the means for modelling the dynamic behaviour (e.g., sequences of actions) of the automation system.

AutomationML subsumes some of the above listed formats for plant information representation (e.g., CAEX) and as such it is listed among the RAMI4.0 and Industry 4.0 standards. In PROPHESY it can provide a methodology (including reference and use of other schemata) for representing plant information and industrial automation logic.

2.2.3.1.1 IEC 62424 CAEX

Similar to XMpLant, CAEX (Computer Aided Engineering Exchange) is a neutral data format, which provides the means for describing a plant and its elements, but also for exchanging data across different process engineering and process control tools. CAEX assumes a hierarchical plant architecture and enables storage of hierarchical object information. It enables an object-oriented approach to plan modelling, as it uses objects to represent the various modules of a plant. Note that CAEX is also an XML format and hence it is defined as XML schema. Elements of the latter schema could be used to structure plant descriptions in PROPHESY.

[5] https://www.automationml.org/o.red.c/home.html

2.2.3.1.2 IEC 61360

As stated in [3], properties classify a system and can be expressed by means of simple values. In the Industry 4.0 context, information needs to be easily shared between all the actors located in the three-dimensional model of the RAMI 4.0. Although it is not feasible to define a single and universal information model, the application of property models can help to achieve interoperability between systems and software components [4]. In particular, properties can be used to define and communicate the actual state, the actual configuration and/or setting variables, as well as, the capabilities of the system or physical asset. To do that, properties need to be defined in a very generic and unambiguous way, the standard IEC-61360 defines the structure and the necessary elements to be used to organize the property itself.

2.2.4 Far-Edge

The main goal of H2020 FAR-EDGE is to provide and validate a novel industrial automation platform that leverages the edge computing paradigm, along with distributed ledger technologies for secure state synchronization of edge industrial systems in a plant or across the supply chain.

FAR-EDGE is designing, developing and validating an edge computing platform for factory automation, which offers functionalities in three distinct, yet complementary domains, namely Automation, Analytics and Simulation. The Analytics domain provides the means for collecting, filtering and processing large volumes of data from the manufacturing shopfloor towards calculating indicators associated with manufacturing performance and automation. Analytics functions are offered by the Distributed Data Analytics (DDA) infrastructure, which is enabling analytics functions within FAR-EDGE compliant deployments i.e. deployments that comply with the structuring principles of the FAR-EDGE Platform architecture. Another part of FAR-EDGE platform design and implementation is also the development of a range of digital models for the three main functional domains of its platform, namely automation, simulation and Distributed Data Analytics (DDA).

A relevel part of FAREDGE to the PROPHESY scope is the Distributed Data Analytics (DDA). DDA is the specification and prototype implementation of the EAE (Edge Analytics Engine) enabler of the FAR-EDGE architecture, which is a runtime environment that provides the means for executing edge scoped analytics functions within edge gateways. The EAE interfaces to the Edge Computing Infrastructure (ECI) of the project in order to access field data. Its core operation is enabled by three types of processor functions, which enable the pre-processing of data streams, their data analysis and ultimately the storage of the analytics results. The EAE enable the concurrent execution of multiple instances of the above types of processing operations at the same time.

One of the main innovations of the EAE is that it is flexibly configurable and programmable. In particular, the operation of the EAE can be driven by an Analytics Orchestrator Manifest (APM) document, which specify how available processor functions can be combined to yield a specific analytics calculation. This specification leverages devices and processing functions that are available in the data streams registry of the ECI. Hence, solution integrators and manufacturers can flexibly configure their analytics operations through defining APM.

The EAE is also extensible in terms of processing functions. Providers and integrators of factory automation solutions can define new processing capabilities, as soon as they adhere to the three main types of EAE processors. For example, the EAE can be extended with processors that implement machine learning schemes such as logistic regression or Bayesian processing techniques, in addition to processors that provide support for simple statistical calculation over data streams such as averages and standard deviations. By registering these processing capabilities in the device registry of the ECI, they can enable their use in the scope of the definition of edge analytics functions via APM. Overall, EAE is not only configurable and programmable, but also flexibly extensible, in terms of processing functions.

2.2.5 Human-Machine Interface

Telegram[6]

Telegram is a cloud-based instant messaging and voice over IP (VoIP) service developed by Telegram Messenger LLP. Telegram client apps are available for the most common platforms such as Android, iOS, macOS, Windows, Linux, etc. By using these apps, users can exchange messages and files of any type.

The usage of telegram is extremely interesting especially for the design and implementation of the so-called "bots". Bots are telegram accounts that are operated by programs allowing software programs to communicate and exchange messages with other programs (bots) and/or human user.

2.2.6 Messaging Integration patterns

Now days, rarely exists monolithic applications which can meet consumers demand. The key concept is the system design and implementation to be more in a service-oriented approach, in which, every service is exposed as a micro-service which "produces" a specific task. The full functionality of the designed system can be achieved by integrating these services. Of course, integration is not just a simple procedure, because we must face different implementations, came from various vendors, for each simple component. Integration patterns come to solve this issue by providing a guidance. In fact, integration patterns are not invented: they are best practices originated from previous integration projects. For the scope of data streaming we investigate the usage of the messaging integration patterns which are a subset of the integration patterns (Figure 4).

[6] https://telegram.org

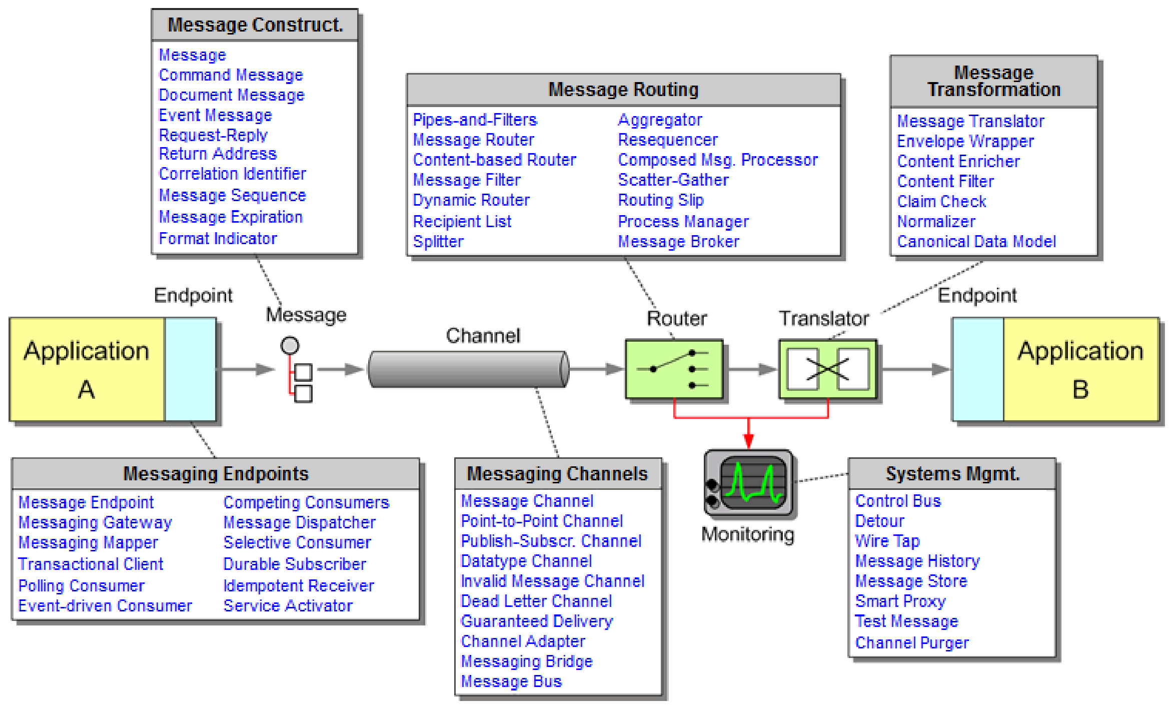


Figure 4: Messaging integration patterns. source: [5]

2.2.6.1 Why using messaging

The adoption of messaging approach to an integration solution has many advantages[5]:

- Remote Communication: Messaging enables separate applications to communicate and transfer data.
- Platform/Language Integration: Systems are connecting via remote communication, usually use different languages, technologies and protocols. A messaging system can be a translator between these systems, allowing them to all communicate through a common way.
- Asynchronous Communication: Messaging enables "a send and forget" approach to communication. The sender does not have to wait for the receiver to receive and process the message. If the receiver wants to send an acknowledgement back to the sender, this message will be detected by a call-back mechanism on the sender.
- Mediation: A messaging system acts as a mediator between all the systems that can send and receive messages. A system can use it as a directory of other applications or services available to integrate with.
- Reliable Communication: Messaging provides reliable delivery, that a remote procedure call (RPC) cannot, because uses a "store and forward approach" to transmitting messages. The data is packaged as messages, which are atomic, independent units. When the sender sends a message, the messaging system stores the message. It then delivers the message by forwarding it to the receiver.

2.2.6.2 Basic messaging concept

- Channels — Messaging applications transmit data through a Message Channel, a virtual pipe that connects a sender to a receiver.
- Messages — A Message is an atomic packet of data that can be transmitted on a channel. Thus, to transmit data, an application must break the data into one or more packets, wrap each packet as a message, and then send the message on a channel. Likewise, a receiver application receives a message and must extract the data from the message to process it. The message system will try repeatedly to deliver the message (e.g., transmit it from the sender to the receiver) until it succeeds.
- Multi-step delivery — In the simplest case, the message system delivers a message directly from the sender's computer to the receiver's computer. However, actions often need to be performed on the message after it is sent by its original sender but before it is received by its final receiver. For example, the message may have to be validated or transformed because the receiver expects a different message format than the sender. A Pipes and Filters architecture describes how multiple processing steps can be chained together using channels.
- Routing — In a large enterprise with numerous applications and channels to connect them, a message may have to go through several channels to reach its destination. The route a message must follow may be so complex that the original sender does not know what channel will get the message to the final receiver. Instead, the original sender sends the message to a Message Router, an application component and filter in the pipes-and-filters architecture, which will determine how to navigate the channel topology and direct the message to the final receiver, or at least to the next router.
- Transformation — Various applications may not agree on the format for the same conceptual data; the sender formats the message one way, yet the receiver expects it to be formatted another way. To reconcile this, the message must go through an intermediate filter, a Message Translator, that converts the message from one format to another.
- Endpoints — An application does not have some built-in capability to interface with a messaging system. Rather, it must contain a layer of code that knows both how the application works and how the messaging system works, bridging the two so that they work together. This bridge code is a set of coordinated Message Endpoints that enable the application to send and receive messages.

3 Overview of PROPHESY-CPS Middleware Infrastructure Elements

3.1 Middleware Infrastructure architecture

Based on Description of Action (DOA) the middleware infrastructure has two key scopes: a) the first one is to support data collections and analytics in the CPS level and b) the second one is to support data streaming analytics to the cloud layer. This section provides an overview of middleware infrastructure aligned with the PROPHESY-CPS architecture defined in Task 3.1 and depicted in Figure 5.

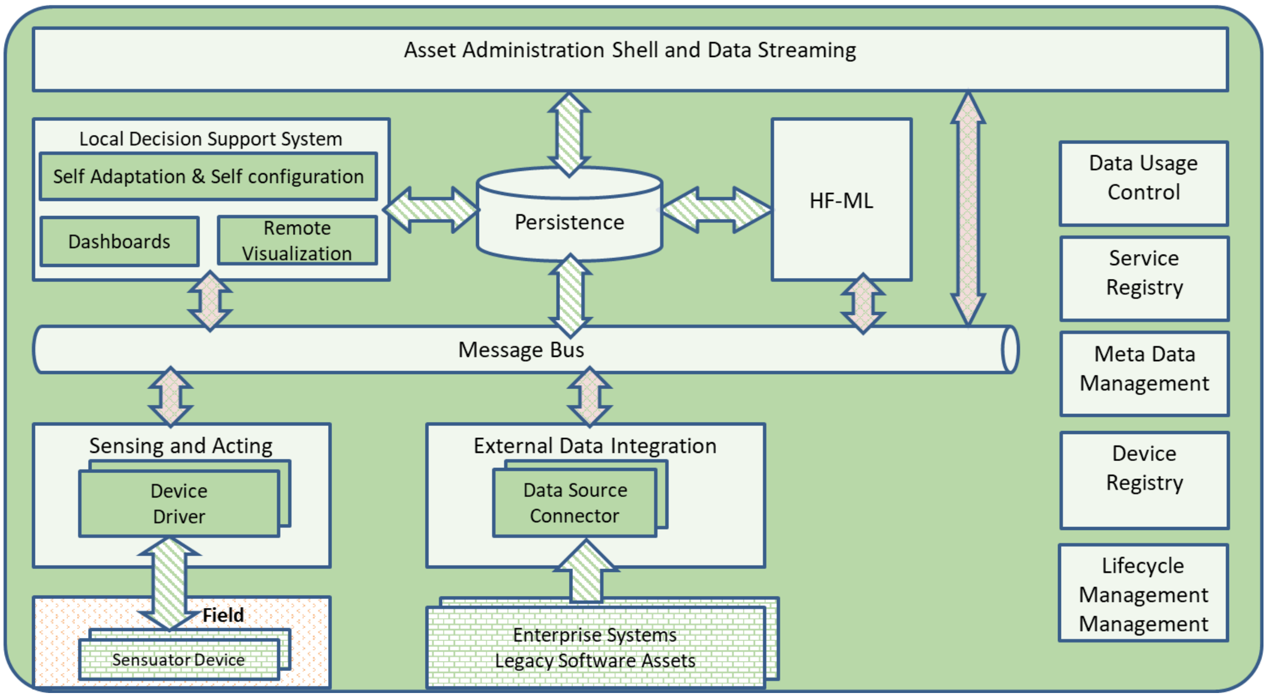


Figure 5: PROPHESY-CPS Logical Architecture based on D3.1

3.1.1 Key functionality of middleware infrastructure.

The functionality of Middleware Infrastructure is acquired by the combination of specifications defined in Description of Action (DoA), in deliverable D2.1 “PROPHESY-CPS Specifications”, in D2.5 “Platform Architectures and Ecosystem Specifications”, in D3.1 “CPS Detailed Architecture v1” and in D3.3 “Digital Modelling and Interoperability”.

In summary, Middleware Infrastructure should follow the specification described in Table 3:

Table 3: Summary of specifications of middleware infrastructure

	Source of spec	Description	Category
1	DoA	support data collection and analytics in PROPHESY-CPS	Enable Analytics
2	DoA	provide data steaming analytics	Enable Analytics
3	DoA	enable analytics both at the edge and at the cloud layers of the PROPHESY-CPS	Enable Analytics and cloud integration
4	D2.1	AAS is responsible for the communication with systems outside the CPS for exposing the functionality	Communication
5	D2.1	PROPHESY-CPS should expose the following functionality to external systems: • Provide access in the streams produced by the internal assets (sensors, machines, analytics results) • Mechanism which enables the configuration of High-Frequency Machine Learning • Mechanism which enables the configuration sensing and acting	Communication
6	D2.5	PdM middleware infrastructure is similar to the CPS's one.	cloud integration
7	D3.1	The final detailed architecture of PROPHESY-CPS is provided (Figure 5) and is aligned with MANTIS.	Communication and cloud integration
8	D3.3	Use a common model which describes elements and data in CPS and platform level	All
9	D2.1 & D2.5	In PROPHESY we have two different machine learning levels. On CPS level we have HF-ML and in platform level we have LF-ML. HF-ML uses data provided by CPS itself and LF-ML processes data streams from every CPS belongs to the PROPHESY platforms	Enable Analytics and cloud integration

All these specifications can be grouped in 3 main categories: the first category refers to the **communication** of PROPHESY-CPS with external systems (and for the PROPHESY scope we can focus on data and functionality exchange between PROPHESY-CPS and PROPHESY-PdM), the second one refers to the mechanism which **enables analytics**, and finally, the last one refers to the **integration in cloud** level. The next paragraph "3.1.2: Middleware Infrastructure in CPS level" is focused on functionality acquired from "communication" and "enable analytics" specifications which is the local scope of middleware infrastructure, while the paragraph "3.1.3: Cloud Integration" is focused on integration between CPS and PdM.

3.1.2 Middleware Infrastructure in CPS level

The adoption of AAS, satisfies the "communication" group of specifications and partly, the "cloud integration". In more details, the AAS can provide access to the internal data stream or functions of CPS, from other systems. Of course, for the scope of RPOPHESY the only "other" system, is the PROPHESY-PdM platform and its elements.

Internally, middleware infrastructure should provide access in every component that consumes data produced by internal assets. As consumers are the local based analytics (HF-ML). Middleware infrastructure should provide a solid mechanism for supporting and enabling analytics running in the CPS and consuming data streams produced by CPS assets. For this reason, we introduce as part of middleware infrastructure, the Data Streaming subsystem which can satisfy the "Enable Analytics" group of specifications. In summary, Data Streaming subsystem should be capable of carrying data streams created by any type of data sources and enabling analytics.

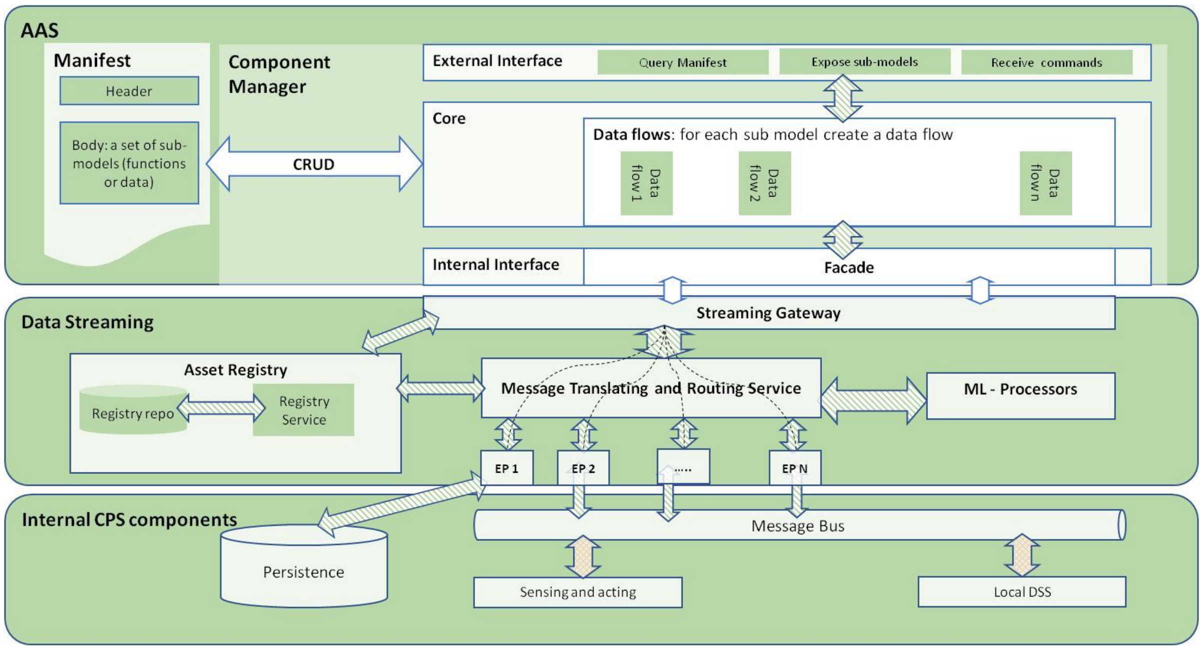


Figure 6: AAS and Data Streaming subsystems of middleware infrastructure

3.1.3 Cloud Integration

The streaming analytics in PROPHESY-PdM platform is similar with the PROPHESY-CPS one. The scope of the platform is to support cloud analytics. Since we adopted the CPS architecture to the PdM platform, Data Streaming sub system in PdM level is a mirroring implementation of Data Streaming in CPS level (Figure 7). It contains the same components and is responsible for registering CPSs into PdM platform and requesting streams for them. Also, it acts as infrastructure for enabling analytics in platform level. In summary, the difference between CPS level and PdM level is that in CPS level, the data sources are topics on a message bus produced by assets. On PdM level, data sources are data streams provided by each CPS.

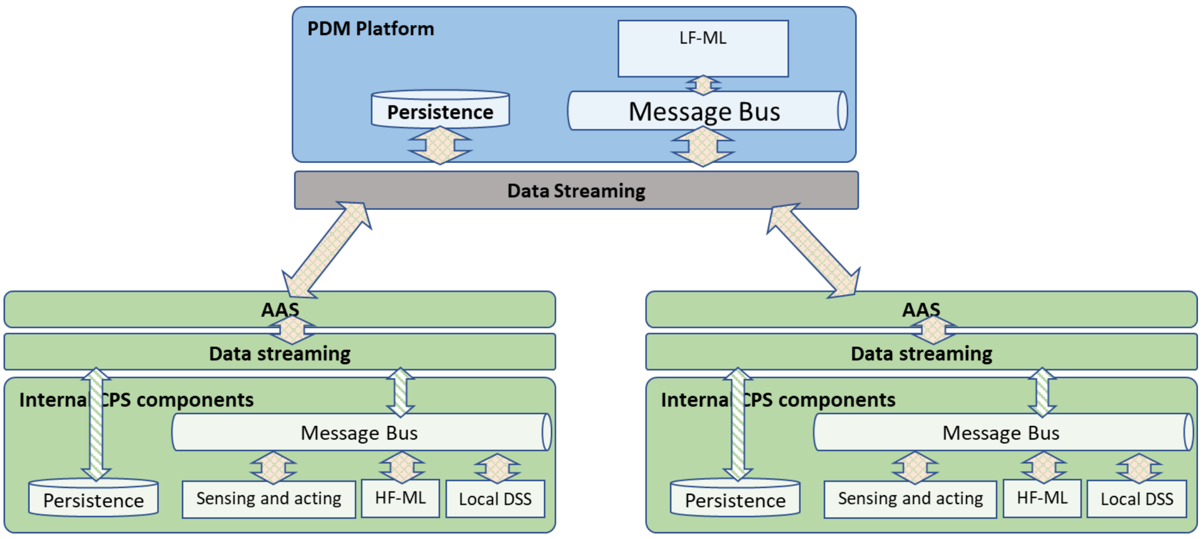


Figure 7: Cloud Integration of data streaming analytics

3.2 Asset Administration Shell

The asset administration shell is tending to support the communication of CPS with external systems, such as PROPHESY PdM platform (Figure 6 and Figure 7). The AAS consists of two components:

- **Manifest**: is responsible to expose to external systems the description of the functionalities and resources are provided by the CPS. It consists of two main parts: a) the Header and b) the Body. The Header contains information about CPS itself and the Body is a collection of sub-models. Each sub-model describes a set of properties are related with the physical asset capabilities. In PROPHESY CPS, each property represents a stream of data produced by assets (sensors, machines or analytics processors) or a command.
- **Component Manager**: provides the necessary services – from one side – for maintaining and managing the AAS, i.e. its own sub- models, and – from the other side – for establishing a communication with the physical asset for gathering the required live data. To do that, the Component Manager comprises the following main sub-components/modules, namely: i) External Interface; ii) Core; and iii) Internal Interface.
    - **External Interface:** provides the instruments to enable the physical asset virtualization through a common interface to the lower levels. In particular, it handles all requests for consuming properties by external users.

- **Core:** provides all the necessary mechanisms and the logic to allow the correct functioning of the AAS.
- **Internal Interface:** acts as the connection layer between the rest CPS and specially the Data Streaming component and the AAS.

The Table 4 provides in summary the components and sub components of AAS. A more detailed description of Asset Administration Shell is provided in the next chapter of this document (Chapter 4: Asset Administration Shel).

Table 4: AAS components in a summary view

Component	Description
Manifest	Expose to external systems the description of the functionalities and resources are provided by the CPS
Header	Information about CPS itself
Body	Collection of sub-models related to live data and functionalities of CPSs assets
Component Manager	Management of AAS Streaming live data to external systems (PdM platform) Receiving commands from external systems (PdM platform)
External Interface	• Communication with external systems • Interface for managing – configuring AAS itself
Core	Business logic for the information flows between CPS and external systems
Internal Interface	The connection layer between data streaming sub-system and AAS

3.3 Data Streaming

Data Streaming sub-system is responsible for streaming data from logical data sources inside the CPS, to the AAS. Also, data streaming component provides the necessary infrastructure for analytics processing (Figure 6 and Figure 7). Data Streaming consists of the following components:

- **Streaming Gateway**: The main role of this component is to provide a connection Interface with AAS in order to enable data streaming outside the CPS. By receiving a request from AAS, the Streaming Gateway discovers the appropriate data source which satisfies this request. Once the data source is discovered, the Streaming Gateway routes the request to the specific data source and creates a data stream between AAS and data source.
- **Asset/CPS Registry**: This component provides flexibility and data source discoverability inside the CPS (aligned with the local scope of CPS). All assets inside the CPS are registered into this component. Once they registered, they can be discovered by other components (either processors either AAS) and finally their data can be streamed. In platform level the registry satisfies a more global scope. For this reason,

the registry in platform level is enhanced with CPS registration functionality and not only by registering asset.

- **Message Translating and Routing Service**: This component is responsible for creating data flows from the data sources to analytics processors, in order to support data analytics in CPS level. The Message Translating and Routing Service provides also cloud analytics in PdM level. The only difference in platform level is that the data sources are streams provided by AAS.

The Table 5 provides in summary the components and sub components of Data Streaming sub system. A more detailed description of Data Streaming is provided in the Chapter 5: Data streaming in CPS and cloud.

Table 5: Data Streaming Components in a summary view

Component	Description
Streaming Gateway	Receives requests from AAS for consuming live data and routes to the specific data source.
Asset/CPS Registry	It is the registry in which all assets of PROPHESY CPS are declared in order to be accessible. Provides asset discoverability. In cloud level registry is extended in order to include CPSs.
Message Translating and Routing Service	It is responsible for activating analytics processors and creating data flows between data producers and data consumers.

4 Asset Administration Shell

In PROPHESY, the AAS concept is introduced to allow the integration of any existing asset in the solution developed within the scope of the project. The main purpose of the AAS is to equip CPSs with the necessary capabilities to talk and share information within the cyber world, i.e. to deliver to the application developers an abstraction layer that is capable to handle the typical heterogeneity of industrial processes. In such a way, developers can easily use the AAS interface to talk with the asset regardless the specific network technology deployed, and ontology used.

4.1 Conceptual overview

The asset administration shell concept (AAS) is strictly connected to the I4.0 component concept. The I4.0 component can be defined as a Cyber Physical System (CPS) in the context of the RAMI4.0 and connects physical and cyber worlds with the objective of creating a unique virtual representation of the physical asset within RAMI4.0 tri-dimensional model. The I4.0 Component can be defined as:

"*globally unique identifiable participant with communication capabilities consisting of an Administration Shell and an asset within an I4.0 system*" based on [6].

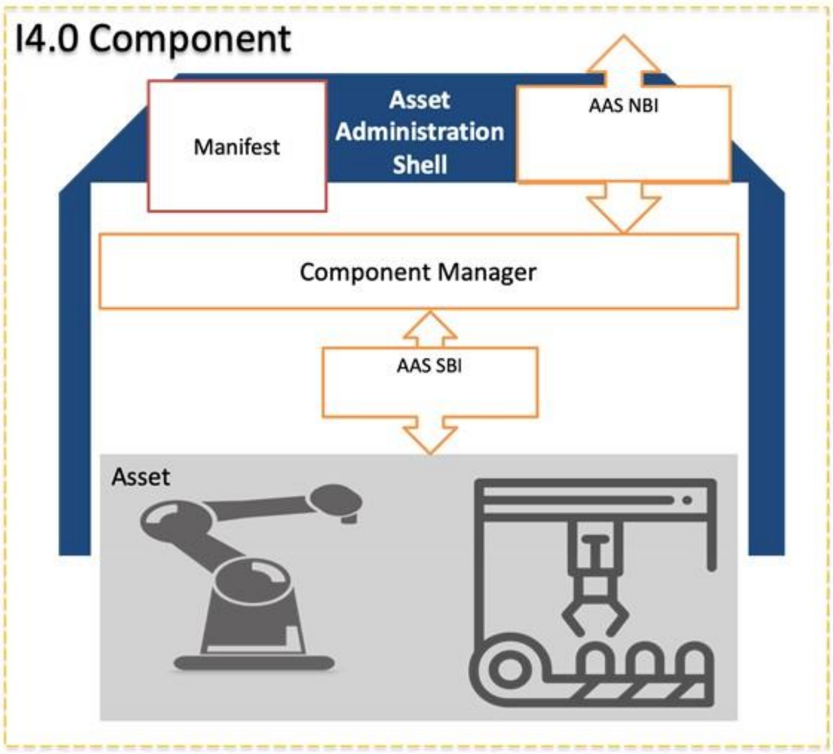


Figure 8: The I4.0 Component logical representation [7]

Therefore, an I4.0 Component is composed by two mandatory elements, namely: the AAS and the asset. In particular, the AAS is the virtual representation of the asset, i.e. the "cyberization" of the physical world. The AAS is – in turn – composed by a *Manifest* and a *Component Manager*. The *Manifest* aggregates information about identification, data and functionalities exposed while the *Component Manager* is responsible to administer the models within the Manifest as well as to link the information in the AAS to both physical and cyber worlds. In such a way, the *Component Manager* includes two interfaces, namely a *northbound interface* (NBI) and a *southbound interface* (SBI). The former provides a uniform Application Programming Interface (API) that receives and send data in a strict and coherent format. The latter is very specific API (strictly dependent on the technology deployed) and provides access to the asset and – thus – to the operational and run-time data. In this scenario, the *Component Manager* is responsible to guarantee the translation of the data to/from a strict and coherent format from/to very specific domain ontologies or – in other words – within the AAS all the information acquired from physical asset is translated and presented in a "understandable" form in order to be easily used and shared.

Conceptual View of a Service-Oriented Device Architecture (SODA)

The envisioned PROPHESY AAS relies on the SOA paradigm in order to enable the design and development of distributed networked systems. In particular, the AAS uses the NBI interface to connect to the Manufacturing SODA backbone while enabling the AAS to be accessible by any user that is interested to the information the AAS includes. The manufacturing SODA backbone is where data is shared between the AASs (see Figure 9). In the context of the PROPHESY project, the focus is not on the communication between AASs but on the communication between AAS and PROPHESY-PdM platform. Since PROPHESY-PdM platform can be considered on a higher level of abstraction than the PROPHESY-CPS thus a hierarchical organization – and communication pattern – between the CPS and PdM layers is being investigated (see Figure 10) in line with what it has been specified in deliverable D2.1 – PROPHESY-CPS Specification.

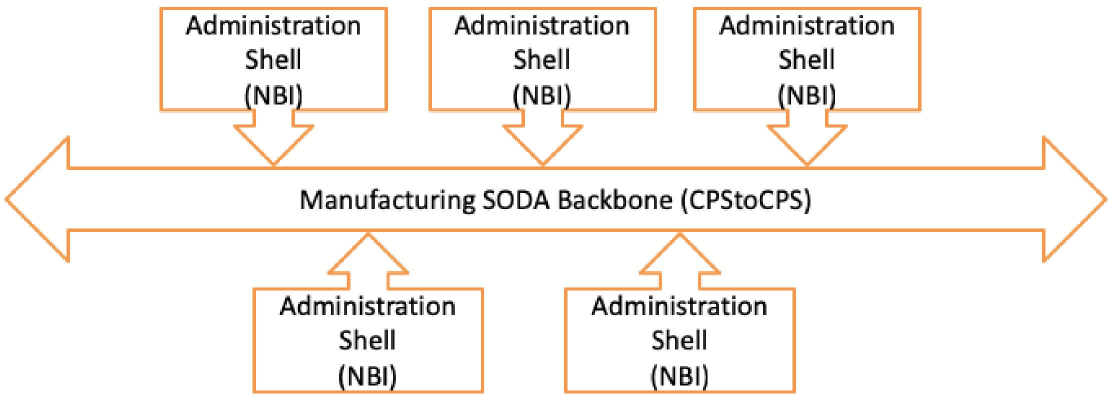

Administration Shell (NBI)
Administration Shell (NBI)
Administration Shell (NBI)
Manufacturing SODA Backbone (CPStoCPS)
Administration Shell (NBI)
Administration Shell (NBI)

Figure 9: AAS Example Network

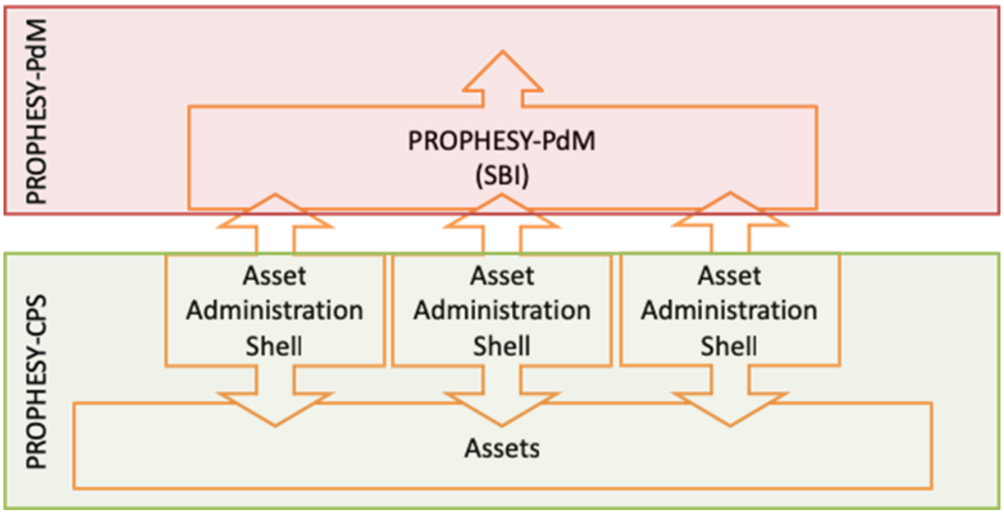

PROPHESY-PdM
PROPHESY-PdM (SBI)
PROPHESY-CPS
Asset Administration Shell
Asset Administration Shell
Asset Administration Shell
Assets

Figure 10: AAS to PROPHESY-PdM Platform Example Network

4.2 Software Modules/Components

4.2.1 Layered Organization

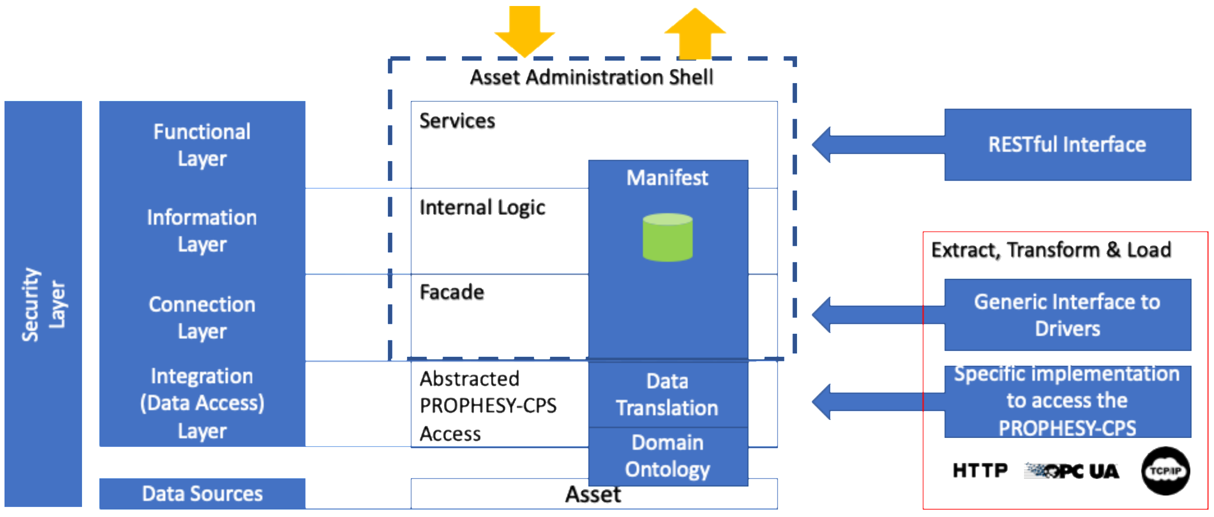

Asset Administration Shell
Security Layer
Functional Layer
Information Layer
Connection Layer
Integration (Data Access) Layer
Data Sources
Services
Internal Logic
Facade
Abstracted PROPHESY-CPS Access
Manifest
Data Translation
Domain Ontology
Asset
RESTful Interface
Extract, Transform & Load
Generic Interface to Drivers
Specific implementation to access the PROPHESY-CPS
HTTP
OPC UA
TCP/IP

Figure 11: AAS Layered Organization and Technical Environment

The AAS relies on a layered architecture pattern that promotes the concept of separation of concerns where software components/modules with the same responsibilities are enclosed within the same layer (see Figure 11). In particular, the *Functional Layer* is where the functionality of the system (i.e. of the PROPHESY-CPS) are exposed as services to the external users. RESTful web services have been used as the reference technology to build the necessary interfaces to be used by service consumers. The *Information Layer* is the core of the AAS, i.e. where the internal logic of the AAS resides. It contains all the processing logic and necessary workflows that makes the “things happen”. Finally, the *Connection* and *Integration Layers* are together responsible for sending and receiving data from the asset. The latter handles the communication with the asset (communication driver) while translating all the requests from the upper layer into the domain-specific ontology and *vice-versa*. The former provides a unique and harmonized communication interface (generic interface to the communication drivers) to access the asset while detaching the AAS from the specific communication technology.

4.2.2 Manifest

4.2.2.1 Core Structuring Model

The Figure 12 shows the core structuring model of the AAS. This model uses *sub-models* as the primary concept to structure the content of the AAS. They are used to expose physical asset capabilities – in a generic and standardized way – within an I4.0 production system. *Sub-models* provide containers for *Properties. Properties* – in turn – are collection of data elements and are used to structure the content of the *sub-models*.

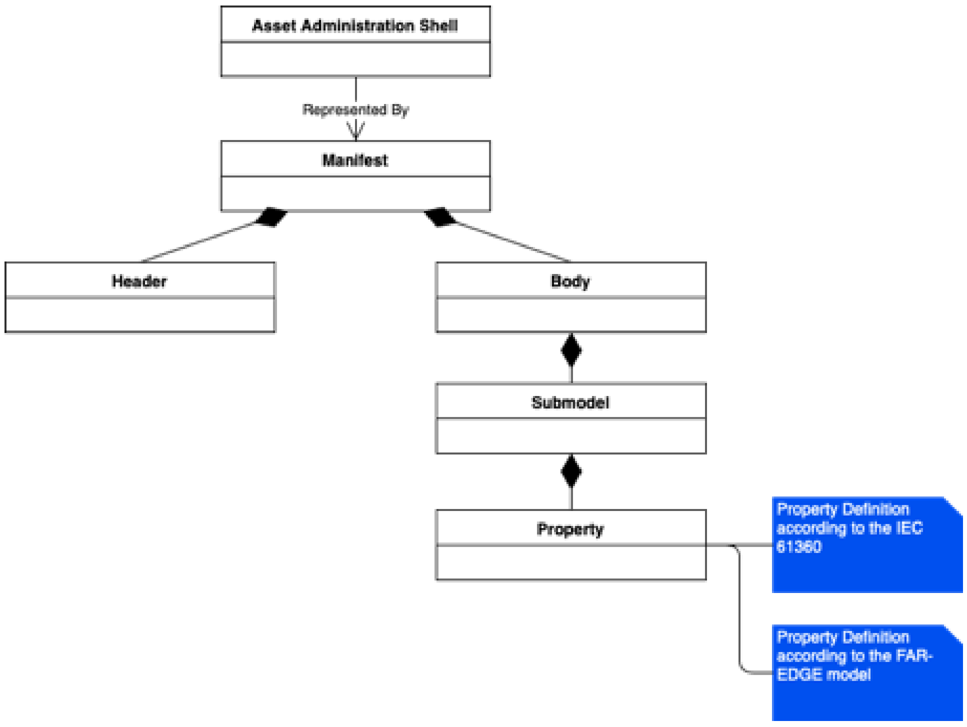

Asset Administration Shell
Represented By
Manifest
Header
Body
Submodel
Property
Property Definition according to the IEC 61360
Property Definition according to the FAR-EDGE model

Figure 12: Asset Administration Shell Core Structuring Model based on [8]

Property Definition and Characterization

The definition and characterization of the *properties* that populate the *sub-models* within the Manifest follows the IEC-61360 standard. However, only a subset of the whole data fields of the standard have been used for defining and characterizing the *properties*. The subset of data fields is in accordance with [7] (see Table 6).

Table 6: Considered Data Fields for Properties Definition and Characterization based on [7]

Property Definition								Property Characterization			
Hierarchy	ID	Name	Definition	Unit of Measure	Data Format	Data Type	Value List	Value	Expression Semantic	Expression Logic	Views

Field	Description	Values
Expression Semantic	Specifies which role the property plays in an interaction	Requirement, Confirmation, Measurement, …
Expression Logic	Specifies the function to be used if different values need to be compared	Equal, Greater Than, Less Than, Not Equal, Less Equal, Greater Equal

For more information about the data fields and the related meeting please refer to [7].

The Figure 13 shows the translation of the meta-model presented in Figure 12 into the Computer Aided Engineering Exchange (CAEX) data format. In particular, the Automation ML tool has been used to create the CAEX representation. The different colours in Figure 13 have the following meaning:

- Pink: standard elements related with the CAEX file;
- Yellow: standard element elements related with the Automation ML standard data format; and
- Green: custom elements introduced to describe the meta-model in Figure 13.

In order to test the AAS a *Manifest* with several sub-models have been created by taking into account the examples and descriptions in [7]. In particular two sub-models have been included within the *Manifest,* namely:

1. Energy Efficiency: this hypothetical sub-model aims to show how the AAS can provide energy-related properties to external users about current consumption of the attached physical asset;
2. Drilling: this hypothetical sub-model aims to show how the AAS can provide confirmation/setting properties to external users about a drilling process executed by the attached physical asset.

The final result is presented in Figure 14.

4.2.2.2 Implementation

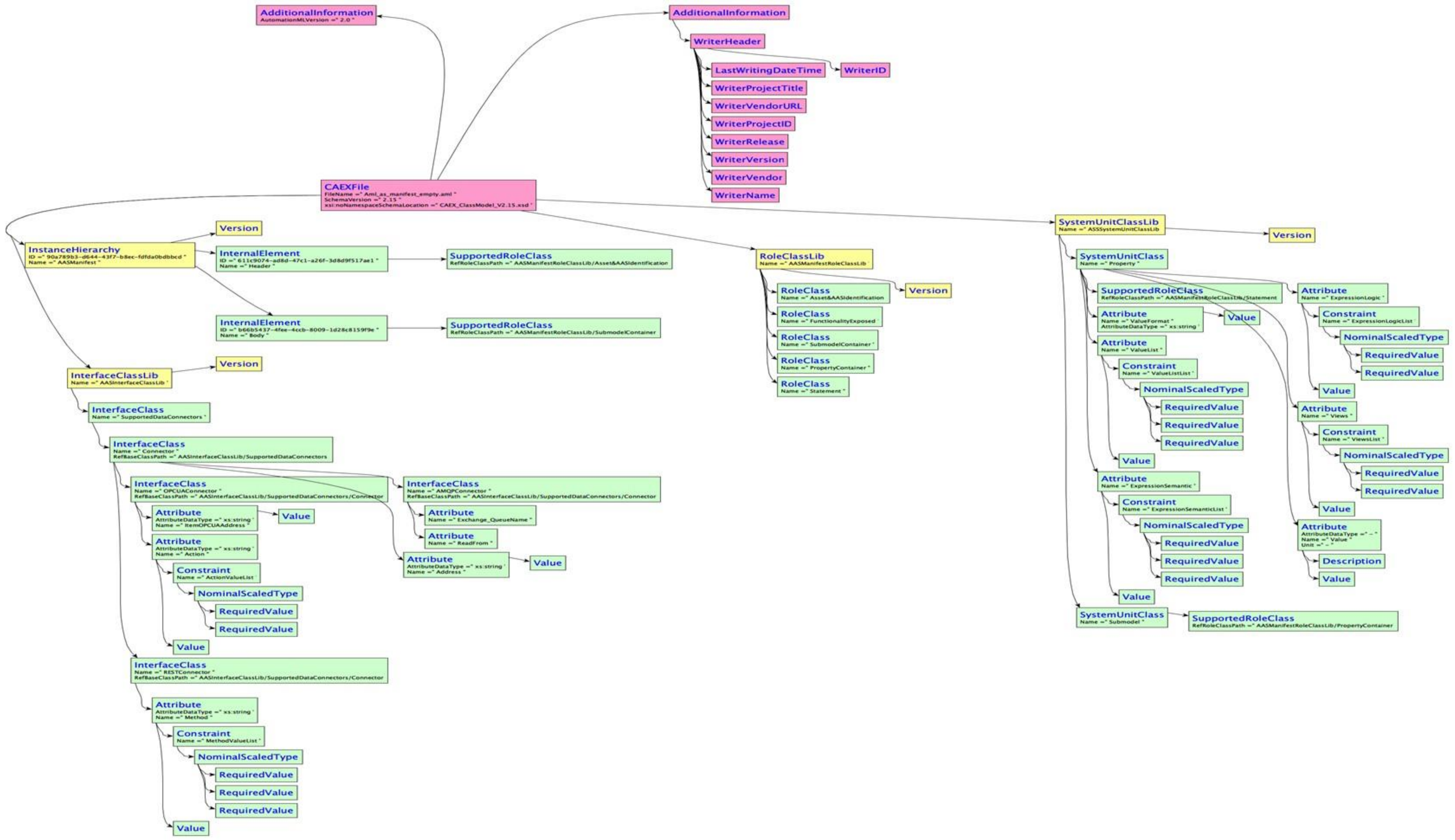


Figure 13:Overview of the Manifest structure (CAEX File)

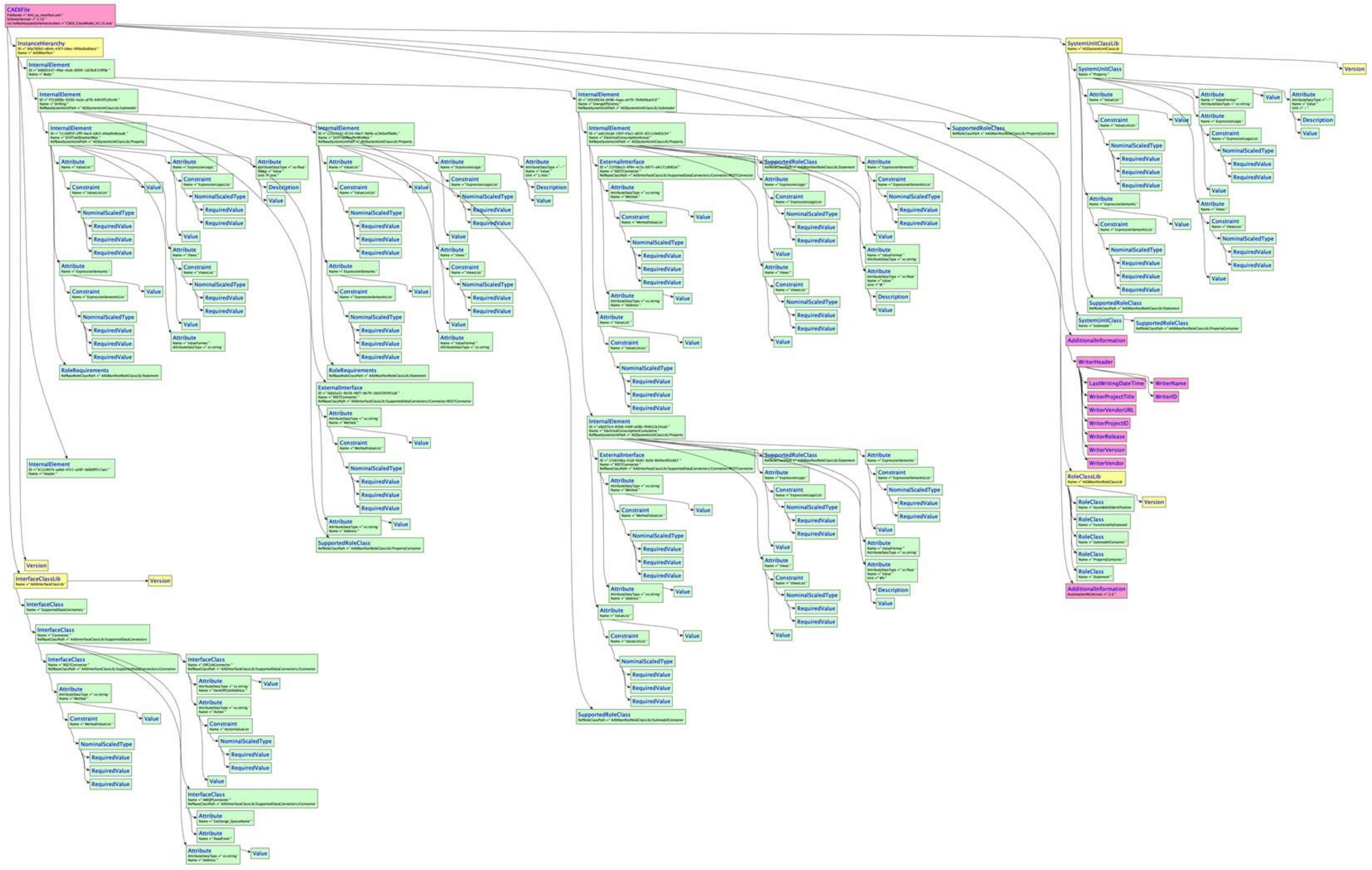


Figure 14: Overview of the Manifest structure populated with two sub-models (CAEX file)

4.2.3 Component manager

The *Component Manager* is aimed to provide – from one side – the necessary services for maintaining and managing the AAS, i.e. its own sub- models, and – from the other side – to establish a communication with the physical asset for gathering the required live data. To do that, the *Component Manager* comprises the following main sub-components/modules, namely: i) External Interface; ii) Core; and iii) Internal Interface.

4.2.3.1 External Interface

The *External Interface* is the sub-component of the *Component Manager* that provides the instruments to enable the physical asset virtualization through a common interface and technology-agnostic access to the lower levels. In particular, it handles the creation (Create), selection (Read), modification (Update) and deletion (Delete) of elements from the *Manifest* requested by external users. It relies on RESTful services and Telegram commands for communication. At the current stage of the PROPHESY project, the *External Interface* exposes the capabilities/functionalities showed in Table 7. The provided API represents the capabilities/functionalities of the AAS and is linked to the *sub-models* specified and described within the *Manifest*, that – in turn – are strictly related to physical asset.

Table 7: External Interface exposed capabilities/functionalities (via RESTful)

External Interface Exposed Functionalities			
Resource	**Operation**	**HTTP Method**	**URL**
AAS	Initialize the AAS	PUT	<aas_URL>/aas/initializtion
	Load a manifest automation ML file	POST	<aas_URL>/aas/loadManifest
Manifest	List the AAS identifiers that are part of the header	GET	<aas_URL>/aas/manifest/headers
	List the AAS submodels that are part of the body	GET	<aas_URL>/aas/manifest/submodels
	Get a specific submodel by id	GET	<aas_URL>/aas/manifest/submodel/:id
	Store a new submodel within the Manifest	POST	<aas_URL>/aas/manifest/addSubmodel
	Delete an existing submodel	DELETE	<aas_URL>/aas/manifest/removeSubmodel/:id
	Update an existing submodel	PUT	<aas_URL>/aas/manifest/updateSubmodel
	Subscribe to a submodel, i.e. generates periodically	PUT	<aas_URL>/aas/manifest/subscribeSubmodel/:id

	events about the status of the submodel		
	Unsubscribe a subscribe submodel	PUT	<aas_URL>/aas/manifest/unsubscribeSubmodel/:id

Next to the RESTful interface, a message-based interaction (implemented by using Telegram) is also possible. This interface allows the interaction with the AAS by using all kind of devices while allowing the contextualization of the functionalities and capabilities, i.e. to organize functionalities and capabilities according to specific rules and/or groups.

AAS Bot Manager

The *AAS Bot Manager* is the sub-component of the *External Interface* that allows the interaction with the AAS by using the Telegram technology. The following commands are available for the early prototype of the AAS.

Table 8: External interface exposed capabilities/functionalities (via Telegram)

Telegram Exposed Functionalities		
Resource	**Operation**	**Command**
AAS	Show the list of commands	/help
	Initialize the AAS	/intialize
	Load a manifest automation ML file	/loadManifest
	Get the current Manifest	/getManifest

4.2.3.2 Core

The *Core* is the sub-component of the *Component Manager* that provides all the necessary mechanisms and the logic to allow the correct functioning of the AAS. In particular, it enables the information flow from the physical world to the cyber world and manage all the necessary transformation and/or routing processes.

Data Flows

The *Data Flows is the* sub-component of the *Core* that manages the creation of flows of data to external applications and/or more in general to any external user. It enables the creation of a communication channel from the physical asset (by using the *Internal Interface*) to the cyber world by connecting the specific communication driver to the generic requested sub-model through the combination of 2 possible mechanisms, namely:

- Services provided by the AAS; and
- Events generated by the AAS (Kafka producer).

4.2.3.3 Internal Interface

The *Internal Interface* is the sub-component of the *Component Manager* is acting as the integration and connection layer between the physical world and the AAS and is designed to provide a generic and standardized operations to access all the elements of the PROPHESY-CPS that are not part of the AAS. Specifically, it is implemented by using a Facade software design pattern (see Figure 15) in order to provide modelling abstractions of physical assets while ensuring their seamlessly integration and usage within the AAS.

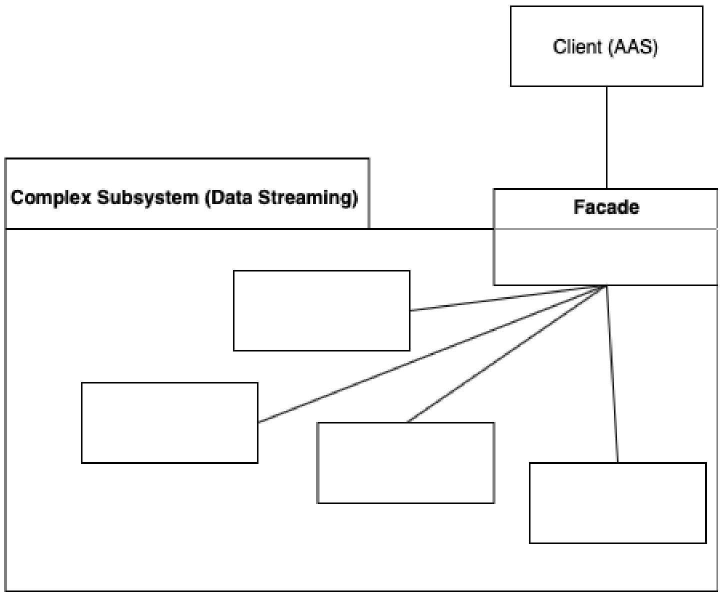


Figure 15: Facade Software Design Pattern

4.2.3.4 Development Environment

The success in preserving digital objects, components, modules or more in general implementations strictly depends on the technology chosen and used [9]. Since technology is a driving force in preserving and continuously change the characteristics of the digital objects, then the process of selecting a technology needs to guarantee that:

1) The technology should be capable to treat new objects created with the latest technologies;
2) The technology should be supported by manufacturers and eventually provides migration path to the next technological generation;
3) The technology should be easy-to-use by any user;
4) The technology should be capable to cope with the heterogeneity of products and digital implementations,

In order to identify a suitable environment for developing the core digital components to be used as the basis for the implementation of the logic associated to the AAS, the following evaluation factors have been extracted from [9] and – thus – considered (see Table 9).

Table 9: Evaluation Factors

Group	Factor	Remark
General	Maturity	Is the technology fully developed and are there already systems in productive use?
	Experience	Are there already verifiable experiences in applying the technology?
	Community	How many developers are using the framework? How active is the development community?
	Standardization	Is the technology based on standards?
	Modularity and Flexibility	Is it easily possible to add new components at low cost, to change or update them?
	Support	Is the technology widespread enough to guarantee that it will be supported by the manufacturers during the desired lifespan of the preservation system?
Technical	Heterogeneity/ Compatibility	Does the technology allow the connection and integration of hardware and software components based on distinct technologies (e.g. REST, OPC-UA, Modbus, etc)?
	Preservation Period (Migration)	Does the technology allow to export implemented digital objects together with their context data in standards format in order to be imported into a new system/device/asset?
	Modularity and Flexibility	Is the technology flexible enough to allow the easy integration of new digital objects?
	Platform	Can the technology be deployed in Linux and Windows based operating systems? Does the technology allow the deployment of digital objects in single-board computers?
	Adaptability	Does the technology allow the easy adaptation of the implementations to cope with changes in logic and/or system evolution
	Licensing	Is the technology fully available to be able to fix bugs or implement extensions?
	Co-creation	Does the technology allow the collaborative creation of digital objects?
	Preservation Period (internet technologies)	Is the technology based on internet technologies in order to guarantee a longer time span?
People	Skills	Does the selected technology need specific skills which must be available in-house? Are these skills already available or can they easily be acquired? Is the technology easy-to-use?
	Experience	Is there sufficient experience with the technology for support in case of difficulties available?
Proce dures	Flexibility	Does the technology allow flexibly implementations? Does it allow changes in the preservation procedures?

The evaluation factors represent criteria that have been considered for selecting the AAS core components development technology. By applying the assessment criteria (see Table 9), it was decided to use Node-red as the core development technology. As a matter of fact, this technology recently hit 1-million npm- installs and is based on a very active community of developers that is present on Slack, Node-red forum, as well as, Stackoverflow. The data base of digital objects (things) and/or nodes is growing every day and is quickly reaching 3000 of objects that can be integrated within the application while enabling compatibility, integrability as well as to cope with the heterogeneity in terms of hardware and software that is typical of today. Furthermore, since node-red is based on the flow-programming paradigm than it provides a graphical and easy-to-use environment that allows to build complex applications by dragging and dropping nodes, wiring and configuring nodes. It can be used by people with no or few programming skills and it makes use of internet technologies (JavaScript) for building up applications as well as new nodes.

5 Data streaming in CPS and cloud

Since Asset Administration Shell subsystem is responsible for the communication with PROPHESY-PdM, Data streaming subsystem is responsible – from one side – for connecting assets to the Asset Administration Shell and vice versa and – from the other side – for supporting data analytics of HF-ML or LF-ML.

The general approach of Data Streaming is to provide an efficient mechanism which can receive data from various source points, process or transform these data and finally delivers these data to a set of destination points (Figure 16). Source points are mainly represented as topics on the message bus, which are the logical twins of the physical devices (such as a sensor or an asset to the field), or the results of the analytics processors. Destination points are either topics to the message bus – as an analytic result – or the Asset Administration Shell which implements external requests comes to the CPS from another CPS or from the PROPHESY–PdM platform. Finally, the data processing is implemented by the machine learning sub system (HF-ML toolbox in the CPS level and LF-ML in the platform level). In conclusion, Data Streaming subsystem is the background infrastructure which a) enables data processing and analytics and b) acts as a proxy for AAS.

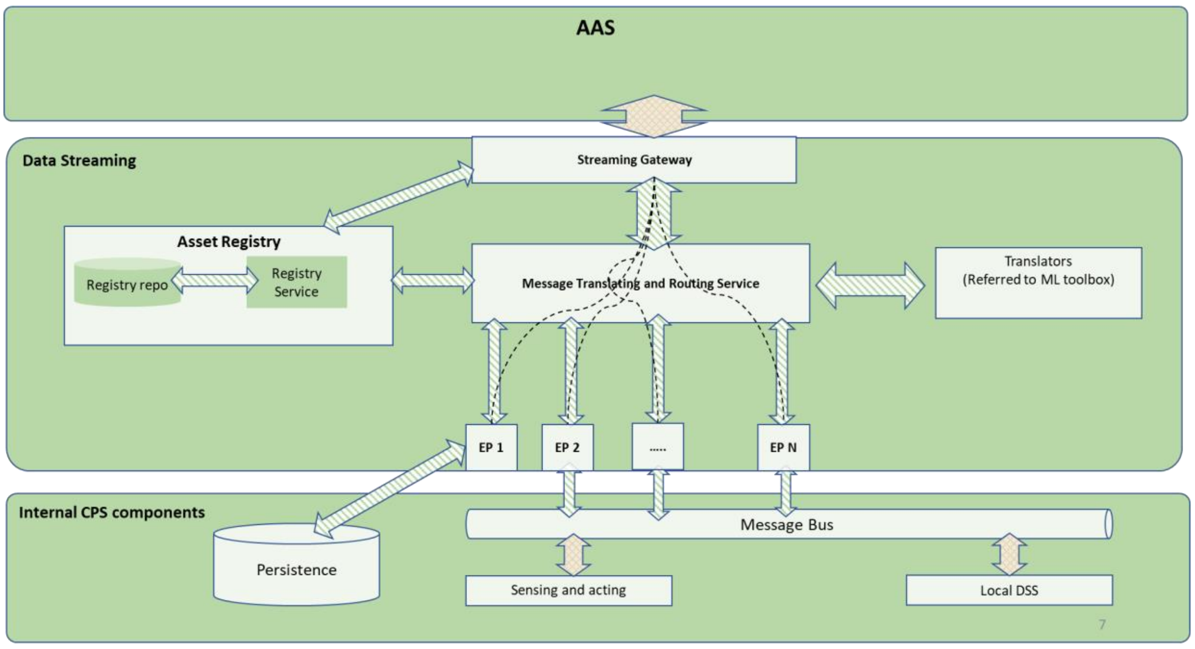


Figure 16: Data Streaming component architecture

5.1 Main concept

The adoption of a message bus in the RPOPHESY CPS architecture, drives us to the usage of a set of messaging integration patterns for the implementation of Data Streaming subsystem. The Data Streaming subsystem is responsible of receiving data from various source points and delivering these data to a set of destination points. In summary, Data Streaming subsystem is the key element of integration inside the PROPHESY CPS extending the Message Bus functionality by providing an abstraction level.

5.1.1 Adopted messaging patters

This paragraph analyses the messaging integration patterns which are adopted for the Data streaming component. The basic concept of messaging is that we have two systems that want to exchange data. The system which sends data is called producer (or sender) and the system which receives data is called consumer (or receiver). Both are called message end points. These data are packaged as messages and the producer sends them through a message channel (Figure 17).

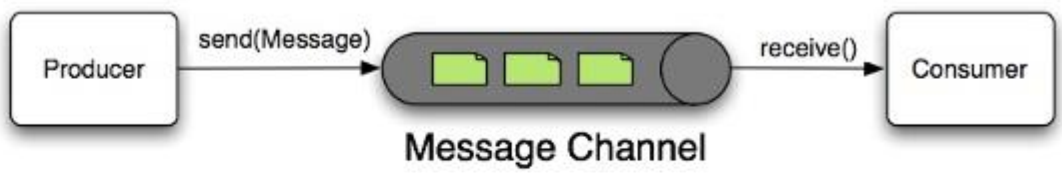


Figure 17: Basic concept of messaging source: [10]

Bellow we describe the basic terms are used in messaging and the proposed patterns for these terms.

5.1.1.1 Message channel

A Message channel is a virtual pipe that connects a sender to a receiver. The system/application sending the data may not know which application will receive the data, but by selecting a particular channel to send the data on, the sender knows that the receiver will be one that is looking for that sort of data by looking for it on that channel. In this way, the applications that produce shared data have a way to communicate with those that wish to consume it. Two messaging patterns for message channels are investigated to be used in data streaming component:

- Point to point channel: Ensures that only one receiver consumes any given message. If the channel has multiple receivers, only one of them can successfully consume a specific message (Figure 18).
- Publish subscribe channel: There is one input channel that splits into multiple output channels, one for each subscriber. When an event is published into the channel, the Publish-Subscribe Channel delivers a copy of the message to each of the output channels (Figure 18).

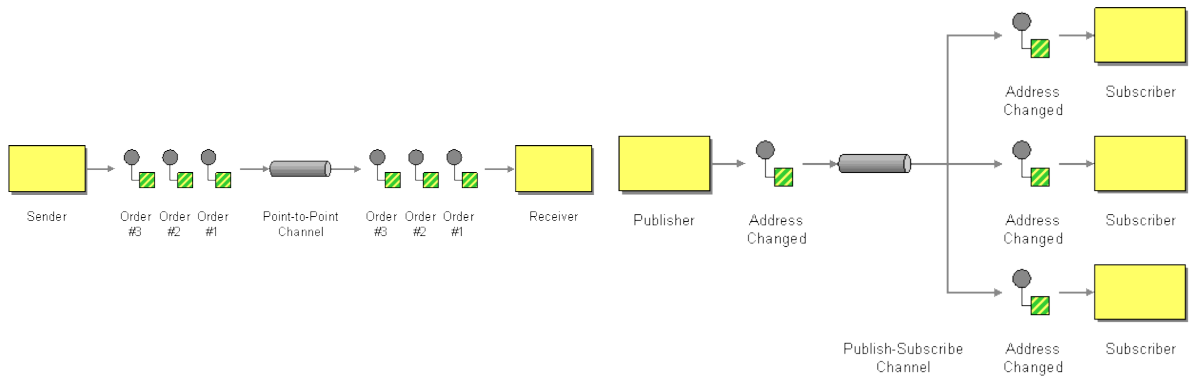


Figure 18: To the left, point to point channel and to the right, publish subscribe channel source: [11]

5.1.1.2 Message

Message is any data record which is packaged to an entity in order to be transmitted through a message channel. For example, based on PROPHESY digital models which is summarily described to the next paragraph, a message can be a liveData/ record which is produced by every data source, logical or not. The following messaging patterns are investigating for the PROPHESY purpose:

- Command message: reliably invokes a procedure in another application. This pattern can be applied by the data streaming infrastructure inside the CPS, in order to receive commands from external systems (PROPHESY -PdM platform) and pushes to appropriate asset. An example can be the customisation of analytics.
- Event message: enables the asynchronous event notification between applications.
- Request reply message: for two-way communication needs.

5.1.1.3 Messaging Endpoints

Messaging Endpoint is a client of the messaging system which enables applications to send or receive messages. Usually applications know nothing about messaging systems. Messaging endpoints receive data from sender application, transform them to message format and sends them to message channel. Also, another endpoint receives messages from message channel extracts the messages and delivers the data to the receiver application (see Figure 19). Generally, there are two main types of endpoints: sender and receiver. Message patterns are investigating for the PROPHESY purposes are:

- Messaging gateway: The Messaging Gateway encapsulates messaging-specific code (e.g., the code required to send or receive a message) and separates it from the rest of the application code which does not know anything about messaging system. This way, only the Messaging Gateway code knows about the messaging system. Messaging Gateway can be used both as sender and receiver.
- Channel adapter: The adapter acts as a messaging client to the messaging system and invokes applications functions via an application-supplied interface. This way, any application can connect to the messaging system and be integrated with other

applications if it has a proper Channel Adapter. It can be applied in both sender and receiver.

- Event-Driven Consumer: The application should use an Event-Driven Consumer, one that is automatically handed messages as they're delivered on the channel. This is also known as an asynchronous receiver, because the receiver does not have a running thread until a call back thread delivers a message. This pattern is referred only to the receivers.

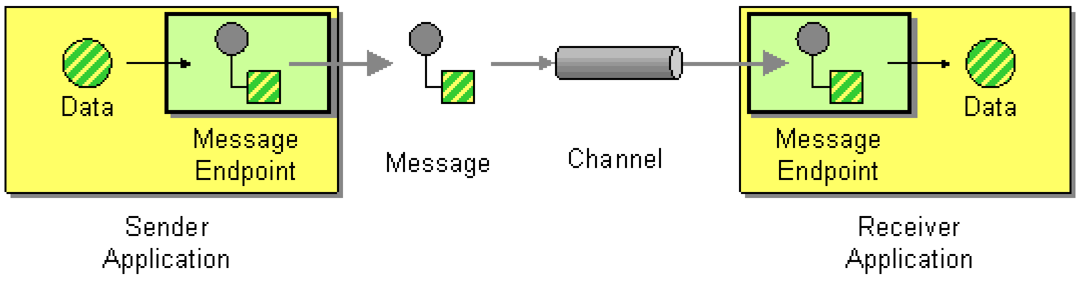


Figure 19: Messaging Endpoints, source: [12]

5.1.2 Definitions from digital models (according to deliverable D3.3)

As defined in deliverable D3.3 "Digital Modelling and Interoperability v1", for the data routing and data exchange, PROPHESY will use a customized version of FAR-EDGE Digital models capable to support the measurements streaming from the monitored machines along with a data routing and configuration methodology. Bellow we describe in summary the main entities of PROPHESY digital models.

5.1.2.1 Data definitions

- **Data Kind (DK)**: This entity specifies the semantics of the data of the data source, which provides flexibility in modelling different types of data. It can be used to define virtually any type of data in an open and extensible way.
- **Data Interface Specification (DI)**: The DI entity is associated with a data source and provides the information need to connect to it and access its data, including details like network protocol, port, network address and more.
- **Data Source Definition (DSD)**: This entity defines the properties of a data source in the shopfloor, such as a data stream from a sensor or an automation device.
- **Processor Definition (PD)**: This entity specifies a processing function to be applied on one or more data sources. It can be used to set up a data routing flow and to utilize analytics algorithms as well.

5.1.2.2 Data manipulation

- **Processor Manifest (PM)**: This entity represents an instance of a processors that is defined through the PD. The instance specifies the type of processors and its actual logic through linking to a programming function.

- **Processor orchestrator (PO)**: A PO entity represents an entire data routing workflow. It defines a combination of data processor instances (i.e. of PMs) that implements a distributed data routing task. The latter is likely to span multiple edge gateways (i.e., CPSs) and to operate over their data sources.

5.1.2.3 Edge Gateway

- **Data Source Manifest (DSM)**: The DSM entity specifies a specific instance of a data source in-line with its DSD, DI and DK specifications. Multiple manifests (i.e. DSMs) are therefore used to represent the data sources that are available in the factory in the scope of the PROPHESY predictive maintenance.

5.1.2.4 Live Data Set (LDS)

- **Live Data Set (LDS)**: The LDS entity models and represents the actual dataset that stem from an instance of a data source that is represented through a DSM. Hence, it references a DSM, which drives the specification of the types of the attributes of the LiveDataSet in-line with the DK. A LiveDataSet is associated with a timestamp and keeps track of the location of the data source in case it is associated with a mobile (rather than a stationary) data source. Hence, it has a location attribute as well. In principle the data source comprises a set of name-value pairs, which adhere to different data types in-line with the DK of the DSM.

5.2 Components of Data Streaming sub-system

The Data Streaming subsystem consists of the following components:

- Asset Registry
- Streaming Gateway
- Message Translating and Routing Service
- Translators

5.2.1 Asset Registry

In order to support a multi-protocol environment and provide a flexibility and data source discoverability in data streaming component, we have designed and implemented Data Source Registry. The Registry functionality consists of two sub-components: a) the Registry Repo and b) the Registry Service. The Registry Repo is a REST Web Service that provides a CRUD interface on the Registry database that follows the PROPHESY digital data model, whereas the Registry Service offers business services, like register device etc. The Registry supports the multi-protocol and multi-data model field environment by registering the devices including not only their connection endpoints, but also the data models and protocols they support. This way a client or a consumer of a data source, e.g. an analytics process, can discover these protocols and data models and use specific clients and data transformations. This way, through the Registry it is possible for PRTOPHESY-CPS processes to connect and acquire data from data sources (devices) following different protocols and/or data models.

5.2.1.1 Registry Service

Based on PROPHESY digital data models introduced in deliverable D3.3 and presented in summary on paragraph 5.1.2, there are three entities which are registered to the registry:

- Data Source Manifest (DSM)
- Processor Manifest (PM)
- Processor Orchestrator (PO)

For each entity, two operations are supported by Registry Service a) register and b) un-register. Figure 20 depicts the sequence diagram for each register and un-register operation. The functionality of Registry Service is exposed through a REST interface:

- **HTTP POST /registry/DeviceRegistration**: a data source is registered as a logical device by providing its Data Source Manifest. The sequence diagram of all actions is depicted in Figure 20.
- **HTTP DELETE /registry/DeviceRegistration/deviceId**: unregisters a logical device identified by deviceId. The sequence diagram is shown in Figure 20.
- **HTTP POST /registry/ProcessorManifest**: registers a processor by providing Processor Manifest. The sequence diagram is similar to Figure 20.
- **HTTP DELETE /registry/ProcessorManifest/processorId**: unregisters a processor identified by processorId. The sequence diagram is similar to Figure 20.
- **HTTP POST /registry/ProcessorOrchestrator**: registers a processor orchestrator. The sequence diagram is similar to Figure 20.
- **HTTP DELETE /registry/ProcessorOrchestrator/orchestratorId**: unregisters a processor orchestrator identified by orchestratorId. The sequence diagram is similar to Figure 20.

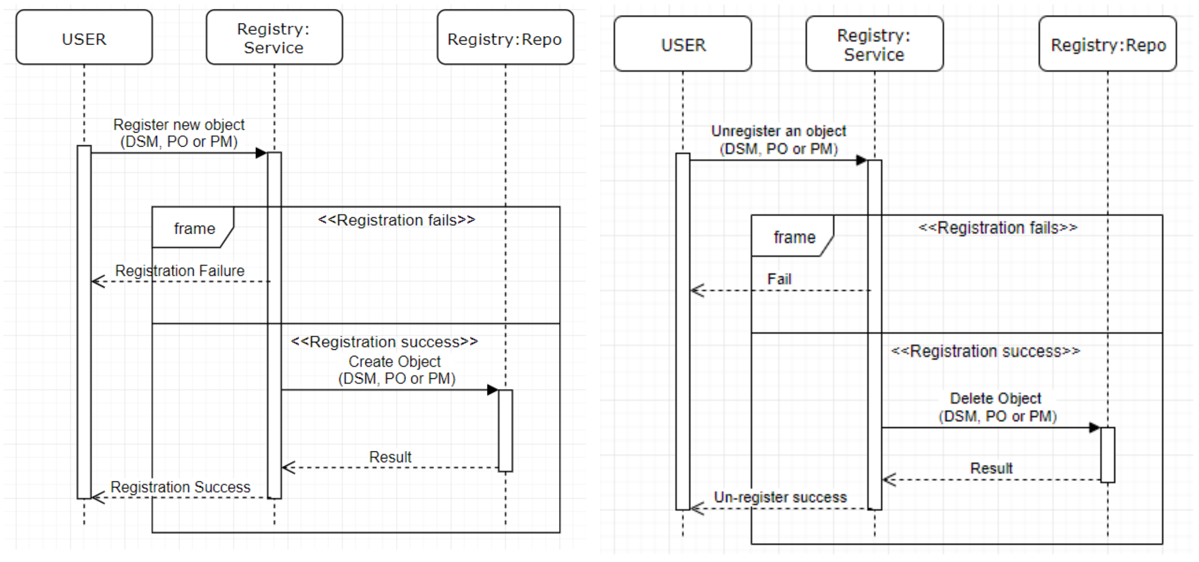


Figure 20: Device registration Sequence diagrams: a) register and b) unregister

5.2.1.2 Registry repo

Device Registry Repo is the persistence layer of the Registry and exposes a CRUD (Create, Read, Update, Delete) RESTful Web Interface.

Exposing its functionality as a REST Web Service, it defines the model entities as Resources (represented as URLs) and allows CRUD actions through the HTTP's GET, POST, PUT and DELETE. The service invocation's request and response payloads are represented as JSON documents. These documents are actually the models described to the deliverable D3.3. The resources that have been identified are:

- /registry-repo/ProcessorOrchestrator
- /registry-repo/ProcessorManifest
- /registry-repo/DataSourceDefinition
- /registry-repo/DataSourceManifest
- /registry-repo/DataKind

All these resources are kept in the Registry Repo Database.

The current implementation of the Asset Registry is based on Java Spring boot framework. The database is H2. Each component (Device Registry service and Device Registry repo) is packaged as a microservice by using Docker images.

5.2.2 Streaming Gateway

The scope of Streaming gateway is to receive requests from AAS and create streams from message bus to AAS and vice versa. When a request, for consuming a property, arrives from AAS to the Streaming Gateway component, the component executes the following tasks:

- Discovers the appropriate data source for the specific property. Every property is presented as a data source definition and the data source discovery is achieved by using the registry Interface. The registry interface returns the specific data source manifest that produces the specific property.
- Creates a channel between the desired data source and AAS. In order to create a channel, we need the start point of the channel which is represented by a DSM, the channel type and the end point. Streaming gateway acts as the end point for AAS. Finally, Streaming Gateway is proposed to support two types of channel: point to point channel or publish subscribe channel.
- Streams messages from data source to the AAS internal interface.

5.2.3 Message Translating and routing Service

This component is responsible for supporting data steaming analytics. It is the core component of the infrastructure which will enable the analytics will be defined in WP4. The basic concept of the "Message Translating and Routing Service" is described in D3.3 and depicted in the Figure 21.

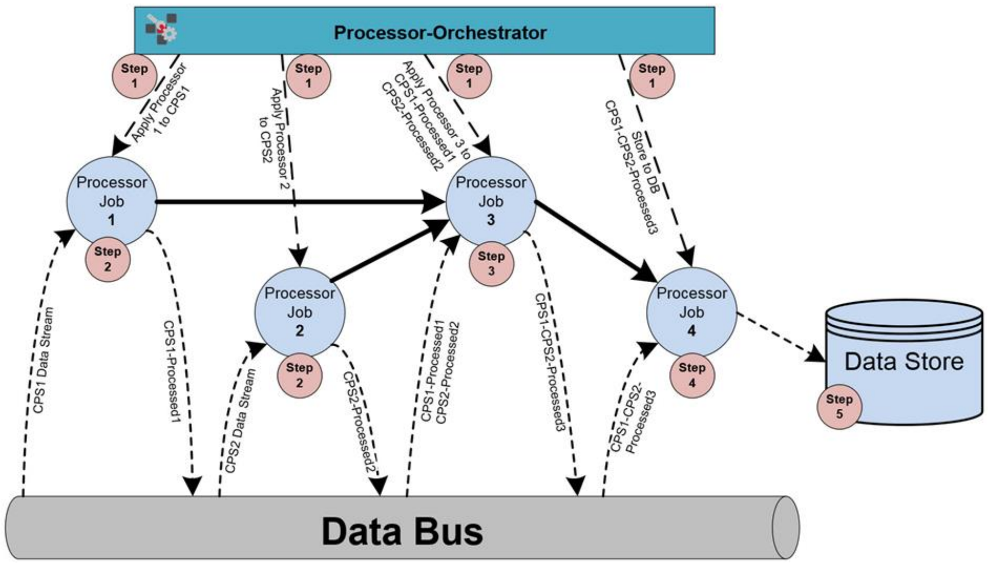


Figure 21: Analytics data routing example

Message translating and Routing Service provides the following functionality:

- Configure and instantiate a data Processor based on a processor manifest. The data processor is used to set up a data routing flow and to utilize analytics algorithms as well.
- Configure and instantiate a Processor Orchestrator. In practise Processor orchestrator is a combination of data processors which can execute more complex data flows such this presented to the Figure 21.

Message Translating and Routing Service adopts many of messaging integration patterns and extends the functionality supported by a message bus. It consists of a) a set of source endpoints b) a set of destination endpoints c) a set of data processors and d) a management interface (Figure 22).

Source endpoint

Data sources are registered in the asset registry and represented as topics to the message bus. As mentioned above data sources are the logical representation of any asset on the CPS which produces data (sensor, machine, analytics processor etc). Data source manifest (DSM) is the instantiation of a data source. As source endpoints are defined the clients are capable to connect the data sources with the data flows and described by the DI. For every new DI a new client needs to be defined. To the first version of middleware infrastructure, a generic "*kafka channel adapter*" is defined as a source endpoint.

Data processors

Data processors are the toolbox which will provide data analytics. The definition of processors is part of the WP4. The scope of the middleware infrastructure is to provide a mechanism to instantiate these processors. Each processor is defined by a processor definition and executes a single data transformation/processing. As input receives a data stream and as a result produces another data stream, which will be stored in a topic. Message Translating and Routing Service creates the message channel which connect data sources with the processors by providing their processor manifest.

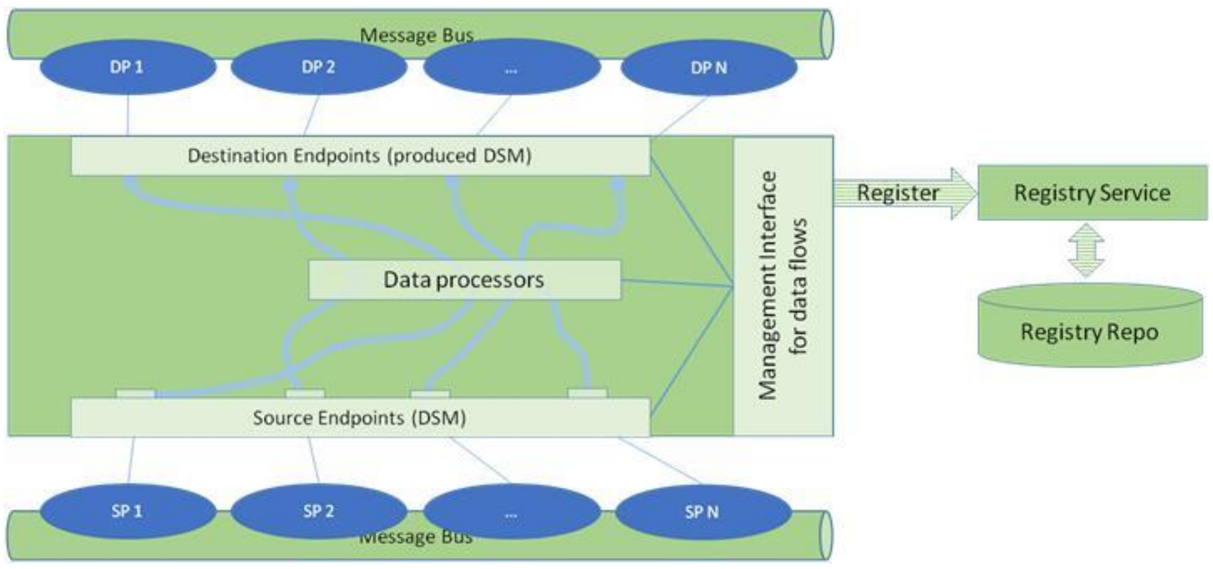


Figure 22: Message Translating and Routing Service

Destination endpoint

The destination of the messages provided by the data flow can be either a topic on the message bus which represents the result of analytics processors (and described by a DSM), either the Streaming Gateway which acts as an endpoint for the AAS. As destination endpoints are defined the clients are capable to connect the data flows with the destination of messages and described by a DI. To this version of middleware infrastructure, two types of destination endpoint are defined, a generic Kafka consumer as destination endpoint for analytics and the Streaming Gateway component.

Management Interface

Finally, the management interface is the core component of the Message Translating and Routing Service and is responsible for configuring and executing the data flows (Figure 24) from one source endpoint to another destination endpoint through a data processor (Figure 25). Furthermore, provides the necessary functionality to create a complex scheme of data flows based on a processor orchestrator. The functionality of Management Interface is exposed through the following REST interface:

- **HTTP POST /routing/processor**: Creates and executes a data flow based on a processor manifest (PM). The source endpoint of the data flow is chosen by the data Source DSM, while the destination endpoint is defined by sink DSM. The sequence diagram of all actions is depicted in Figure 23.
- **HTTP DELETE /routing/ processor/processorId**: Stops the data flow for the PM with id= processorId.

- **HTTP POST /routing/orchestrator**: creates and executes an entire scheme of data flows based on a processor orchestrator (PO).
- **HTTP DELETE /routing/ orchestrator/orchestratorId**: Stops all the data flows of a specific scheme provided by the orchestrator with id= orchestratorId.

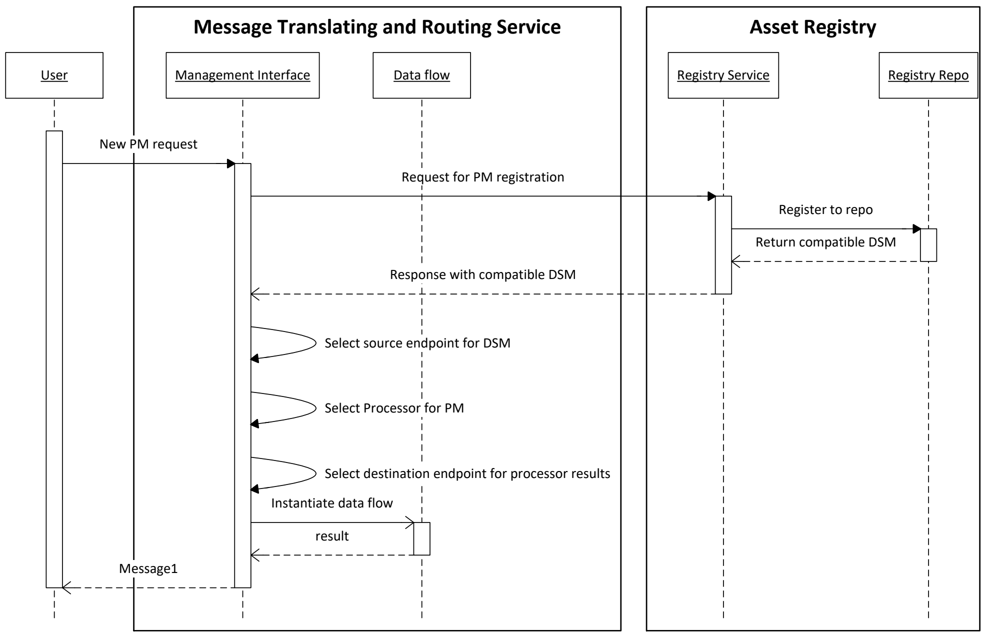


Figure 23: Create a new data flow for analytics: sequence diagram

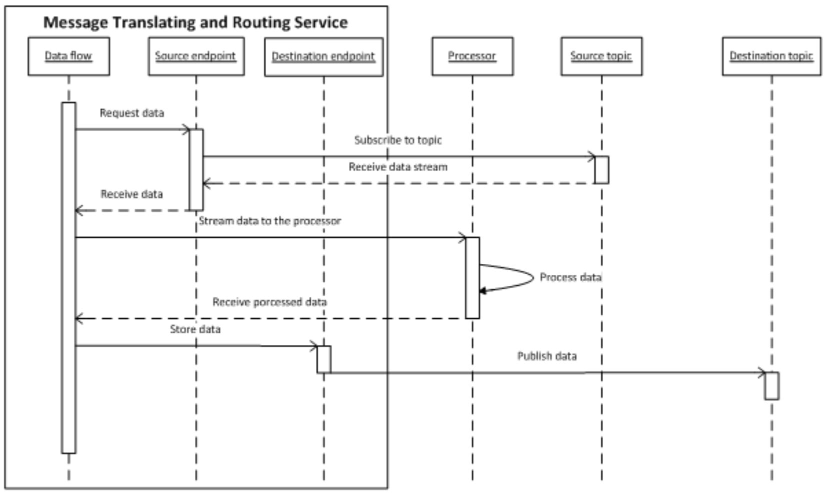


Figure 24: Execute a data flow

5.3 Prototype Implementation

To the current version of middleware infrastructure, are available the following components of Data Streaming subsystem:

- Asset registry (registry service and registry repo)
- Message Translating and Routing Service
  - Management Interface: Implemented the functionality of configuring and instantiating an analytic processor.
  - Source - destination endpoints: Implementation of Kafka producer - consumer
- Streaming Gateway
  - Implemented a REST end point for receiving request- reply calls

5.3.1 Source Code Availability

The source code for Data Streaming subsystem, is placed at PROPHESY private GitHub repository. The repository is named "DataStreaming" which is available at: https://gitlab.com/prophesyEuH2020/DataStreaming. The latest version of the components is under the "develop" branch.

5.3.2 Docker Availability

The spring boot applications "Registry - Service" and "Registry – Repo" are available as docker images in:

- https://gitlab.com/prophesyEuH2020/DataStreaming/docker/registry-repo
- https://gitlab.com/prophesyEuH2020/DataStreaming/docker/registry-service

6 Use cases

The prototype implementation of the CPS middleware Infrastructure presented in this deliverable will be validated in the work package WP6 (Complex Demonstrators Integration and Validation), where all the PROPHESY components will be deployed in the two complex pilot demonstrators.

In the current section, we provide a description of an example in the form of use cases that could lead to the creation of test cases that will validate the operation of the middleware infrastructure components. Each of these examples could be the basis of a single validation scenario or be a part of a more complex validation scenarios.

6.1 Use Case#1: Request/Subscribe Sub-model

Actors	**1. Asset Administration Shell** **2. Data Streaming** **3. Message Bus**
Trigger	An external request arrives to the AAS in order to consume a specific sub-model.
Normal flow	1. The Management Interface of Message Translating and Routing Service receives a request for instantiating a new processor (PM) 2. The Management Interface 3. The Manifest sends back the details for the specific sub-model. 4. The component Manager configures the data flow between EI and Internal Interface (II). 5. Internal Interface pushes the request to Streaming Getaway. 6. Streaming Getaway routes the request to the specific data source through Message Translating & Routing Service 7. Message Translating & Routing Service gets the data and translated to the specific data format
Precondition	The external system is authorized
KPI	N.A.
Frequency	Whenever a request arrives
Exception	Not specified
Alternative flow	Not specified
PROPHESY components	AAS (Manifest, Component manager), Data Streaming (Streaming Gateway – Asset Registry – Message Translating & Routing Service) and Kafka Message Bus

The use cased is aimed to provide a generic and harmonized way to access physical assets. This could improve the enterprise visibility as well as the usage of the data by external applications and/or services. Two types of access are defined: request – response and publish subscribe.

6.1.1 Narrative of Use Case: Data Flow Diagrams

6.1.1.1 Publish Subscribe Sub-model

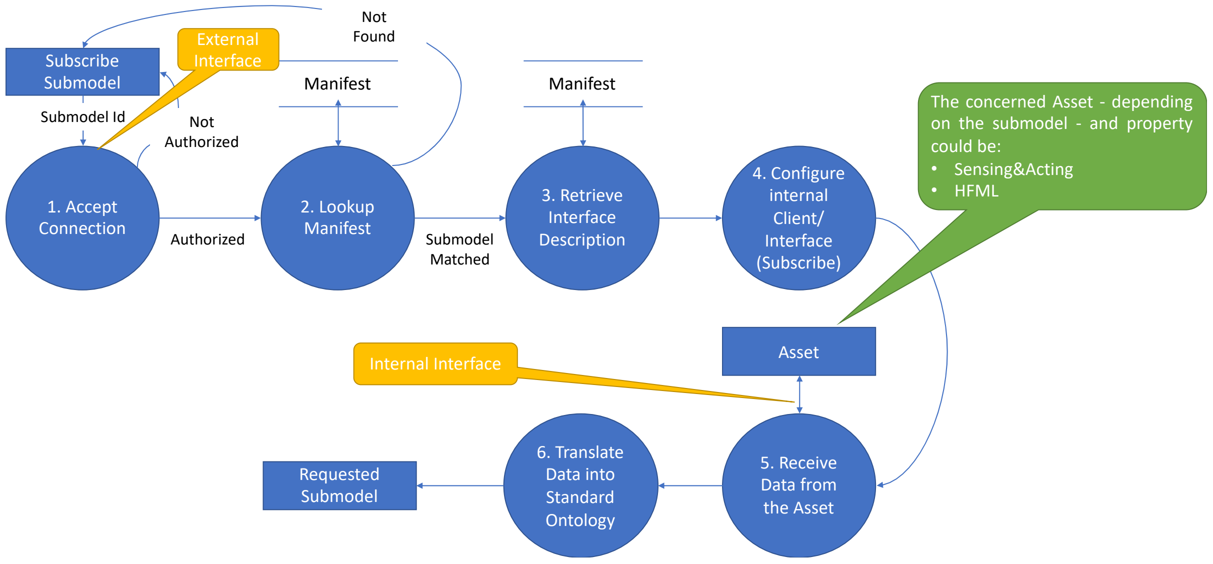

Subscribe Submodel
External Interface
Not Found
Manifest
Manifest
Submodel Id
Not Authorized
1. Accept Connection
Authorized
2. Lookup Manifest
Submodel Matched
3. Retrieve Interface Description
4. Configure internal Client/ Interface (Subscribe)
The concerned Asset - depending on the submodel - and property could be:
• Sensing&Acting
• HFML
Asset
Internal Interface
Requested Submodel
6. Translate Data into Standard Ontology
5. Receive Data from the Asset

Figure 25: Subscribe of a Sub-model

6.1.1.2 Request response Sub-model

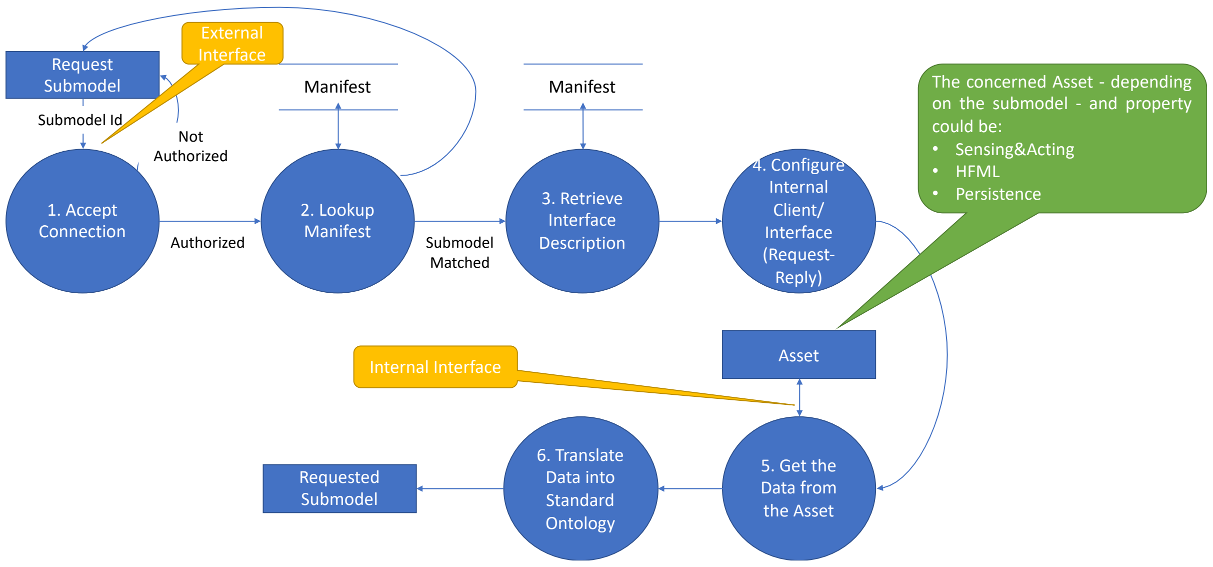

Request Submodel
External Interface
Manifest
Manifest
Submodel Id
Not Authorized
1. Accept Connection
Authorized
2. Lookup Manifest
Submodel Matched
3. Retrieve Interface Description
4. Configure Internal Client/ Interface (Request-Reply)
The concerned Asset - depending on the submodel - and property could be:
• Sensing&Acting
• HFML
• Persistence
Asset
Internal Interface
Requested Submodel
6. Translate Data into Standard Ontology
5. Get the Data from the Asset

Figure 26: Subscribe of a Sub-model

6.2 Use Case#2: Install a new Analytics Processor

Actors	1. **Data Streaming** 2. **Message Bus**
Trigger	A new command for installing a new analytics processor.
Normal flow	1. The Management Interface of the Message Translating and Routing Service receives a request for instantiating a new processor (PM). 2. The Management Interface registers the Processor Manifest to Asset Registry. 3. The Management Interface selects the appropriate client for source endpoint (source DSM). 4. The Management Interface selects the appropriate client for destination endpoint (sink DSM). 5. The Management Interface selects the appropriate processor of the processor library. 6. The Management Interface configure the data flow. 7. The Message Translating and Routing Service start the data flow which produces stream analytics and stores to the specific topic of Message bus.
Precondition	1. Data source is active 2. Processor client is available
KPI	N.A.
Frequency	Whenever a new data analytics installed
Exception	Not specified
Alternative flow	Not specified
PROPHESY components	Data Streaming (Asset Registry – Message Translating & Routing Service), Kafka Message Bus, analytics of WP4

7 References

[1] J. Kreps, N. Narkhede, J. Rao, and others, “Kafka: A distributed messaging system for log processing,” 2011.

[2] A. Warsky, “Evaluating persistent, replicated message queues (updated w/ Kafka),” Blog of Adam Warsky, Jul-2014. [Online]. Available: http://www.warski.org/blog/2014/07/evaluating-persistent-replicated-message-queues/. [Accessed: 20-Apr-2016].

[3] F. Pethig, O. Niggemann, and A. Walter, “Towards Industrie 4.0 compliant configuration of condition monitoring services,” in 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), 2017, pp. 271–276.

[4] U. Epple, “Merkmale als Grundlage der Interoperabilität technischer Systeme,” Autom. At, 2011.

[5] 2003, 2017 Gregor Hohpe, Bobby Woolf, “ Enterprise Integration Patterns - Messaging Patterns Overview” https://www.enterpriseintegrationpatterns.com/patterns/messaging/. [Accessed: 26-Nov-2018].

[6] “Glossary.” [Online]. Available: https://www.plattform-i40.de/I40/Navigation/EN/Service/Glossary/glossary.html. [Accessed: 01-Nov-2018]

[7] ZVEI, “Examples of the Asset Administration Shell for Industrie 4.0 Components – Basic Part.” Apr-2017.

[8] Plattform Industrie 4.0, “Online Library,” Aug-2018. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/vws-in-detail-presentation.html. [Accessed: 02-Nov-2018].

[9] ERPANET, “ERPANET - erpaGuidance.” Sep-2003.

[10] “Spring docs”, https://docs.spring.io/spring-integration/reference/html/overview.html, [Accessed: 26-Nov-2018].

[11] 2003, 2017 Gregor Hohpe, Bobby Woolf, “Enterprise Integration Patterns: messaging patterns: Publish-subscribe channel, https://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html, [Accessed: 26-Nov-2018].

[12] 2003, 2017 Gregor Hohpe, Bobby Woolf, “Enterprise Integration Patterns: messaging patterns: Message endpoint, https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageEndpoint.html, [Accessed: 26-Nov-2018].