



UNIVERSITÀ DEGLI STUDI DI CATANIA

**DIPARTIMENTO DI INGEGNERIA ELETTRICA ELETTRONICA
E INFORMATICA**

**Dottorato di Ricerca in Ingegneria dei Sistemi, Energetica,
Informatica e delle Telecomunicazioni**

XXXIII Ciclo

Marco Giuseppe Salafia

Definition of Novel IoT-based Solutions for Interoperability in Industry 4.0

—————
Ph.D Thesis
—————

**Coordinatore:
Chiar.mo Prof. P. Arena**

**Tutor:
Chiar.mo Prof. Ing. S. Cavalieri**

Contents

1	Introduction	1
1.1	Motivations and Goals	6
1.2	Structure of the Thesis	7
2	Overview on Asset Administration Shell	10
2.1	I4.0 Component and the role of AAS	11
2.1.1	Composite I4.0 Component	13
2.1.2	Asset Type and Asset Instance	14
2.2	Structure of the AAS	15
2.3	Implementation variants of an AAS	17
2.4	Asset Administration Shell metamodel	19
2.4.1	Common Classes	20
2.4.2	Main Classes	24
3	Overview on OPC UA	29
3.1	Information Modeling	30
3.1.1	Metamodel and Base Information Model	31
3.1.2	Type modeling for Objects	35
3.2	Data type system of OPC UA	38
3.2.1	Encoding of DataTypes	40
3.3	Common practices for the definition of new Information Models	42
3.3.1	Variables and DataTypes definition	43
3.3.2	Objects and ObjectTypes definition	44
3.3.3	AddressSpace Organisation	44

3.4	Research activities to enhance interoperability based on OPC UA	45
3.4.1	OPC UA Web Platform	46
3.4.2	Interoperability between OPC UA and OCF	50
3.5	Publications	52
4	OPC UA-based Asset Administration Shell	53
4.1	Introduction	54
4.2	Mapping AAS metamodel into OPC UA Information Model	55
4.2.1	Mapping AAS Entities	55
4.2.2	Structuring the OPC UA AddressSpace	59
4.2.3	Mapping AAS References	61
4.2.4	Mapping AAS common classes	64
4.2.5	Mapping HasDataSpecification and DataSpecification	66
4.2.6	Mapping ConceptDictionary and ConceptDescription	68
4.3	Case study: Operator Support System for assembly line	69
4.4	Discussion	74
4.5	Publications	76
5	AAS representing PLC based on IEC 61131-3	77
5.1	Introduction	78
5.2	Overview on IEC 61131-3	80
5.3	Submodel for IEC 61131-3	82
5.3.1	Configuration	84
5.3.2	Resource	85
5.3.3	Program	85
5.3.4	Function Block	87
5.3.5	Function	89
5.3.6	Task	90
5.3.7	Variable	90
5.3.8	Semantics for IEC 61131-3 elements	91
5.4	Using AASs to represent PLC and Real Plant	92
5.5	Case Study: Controlling a Drilling Machine	96

5.6	Implementation of the approach leveraging OPC UA	102
5.7	Discussion	105
5.8	Publications	107
6	A model for PdM based on AAS	108
6.1	Introduction	109
6.2	Overview on Predictive Maintenance	111
6.2.1	Data Acquisition	112
6.2.2	Data Processing	113
6.2.3	Maintenance Decision-making	114
6.3	AAS-based Model for Predictive Maintenance	116
6.3.1	Logical Block for PdM	118
6.3.2	AAS Submodels supporting PdM	121
6.3.3	Description of the PdM model	123
6.4	Case study: Modeling a Cloud-based Machine Learning PdM solution	126
6.4.1	Description of the use case	126
6.4.2	Representing the use case using the PdM model	127
6.5	Discussion	131
6.6	Publications	133
7	Conclusions	134
	Bibliography	139

Chapter 1

Introduction

Nowadays, we are facing a new industrial revolution, referred as the fourth one, introducing the application of modern concepts of Information and Communication Technology (ICT) in industrial contexts to create more flexible products and introducing new business models and added value [1, 2]. The first industrial revolution started at the end of the 18th century with the introduction of mechanical machines in production plant based on steam power. The second industrial revolution started at the beginning of the 20th century with the introduction of electricity and mass production. The third one started around 1970s with the adoption of electronics and Information Technology (IT), introducing the automation in manufacturing process by means of programmed machines [3]. The concept of the fourth industrial revolution was initially proposed in Germany with the term Industry 4.0 [4] to refer the transformation of the factories of today into smart factories intended to face the current challenges of shorter product life-cycles, highly customized products and global competition [5]. Such challenges originate from a global market containing several suppliers producing the same or similar products, thus leading customers to pay more attention to the quality of the services and features provided by producers. For these reasons, product customization, schedule accuracy and flexibility are the fundamental goals that Industry 4.0 wants to achieve, because agile and easily re-configurable production can face new products demands.

Traditional automation cannot achieve the required degree of flexibility. Instead modular factories composed by smart devices, also known with the term Cyber-Physical Systems (CPS), can overcome the currently rigid planning and production processes. Even though lot of several definitions exist, CPS can be defined as “a system comprising a set of interacting physical and digital components, which may be centralised or distributed, providing a combination of sensing, control, computation and networking functions, to influence outcomes in the real world through physical process” [6]. CPS connected to each others define smart manufacturing networks to integrate technologies from different vendors and heterogeneous data gathered across the supply chain to achieve business goals. From this point of view, both production lines and supply chains take advantages of data collected from devices, sensors and actuators to optimize their value chain. Based on this, connecting components and systems located at the shop level, also referred as Operational Technology (OT), and components and systems of the production management level, referred as IT, is required to achieve the connectivity [7] needed to use data from low levels to plan production processes. Technologies like Internet of Things (IoT) and Cloud computing, among others, play a key role in this context because enables the communication between devices and people inside the same enterprise and between partners of the value-chain network. This makes interoperability one of the major goals and one of the outstanding challenges of Industry 4.0. Interoperability can be defined as “the ability of two or more systems or components to exchange and *use* exchanged information in an heterogeneous network” [8]. Heterogeneous devices that were not designed to work together must *interoperate* to reach a common goal, and this is one of the biggest issue given the complex ecosystem of standards adopted, the presence of legacy systems, and smart components [9]. From digitalization point of view, communication protocols, semantic technologies and operational models must be specified and implemented to guarantee that components and systems can be smoothly be connected and that they can interoperate performing own operations and fulfilling business objectives [7]. In [10], there are identified four levels of interoperability: technical, syntactic, semantic, and organizational interoperability.

Technical Interoperability It is achieved among communications-electronics systems or part of them where services or information can be directly exchanged between them and their users. This type of interoperability often focuses on communication protocols and the infrastructure required for the protocols to function.

Syntactic Interoperability Is usually defined as the ability to exchange data and it is generally associated to data formats. In particular, syntactic interoperability involves the definition of well-defined syntax and data encoding, for instance, in the exchange of message.

Semantic Interoperability It is defined as the ability to operate on the data according to an agreed-upon semantics. It is usually related to the definition of content from a human interpretation perspective and, in particular, this level of interoperability is characterised by a common understanding of the definitions used for the content of the data being exchanged.

Organizational Interoperability It refers to the capability of organisation to effectively transfer meaningful data regardless the variety of information systems over significantly different types of infrastructure, possibly across various geographic regions and cultures. It strictly depends from technical, syntactic and semantic interoperability to be successful.

There are several interoperability assessment approaches in literature as discussed in [11] and depicted in Figure 1.1 where models for the interoperability assessment are ordered for popularity. Each of these models uses a different approach to assess the degree of interoperability, pointing out that the process of interoperability evaluation varies case by case on the basis of the subjects and needs in exam.

A simpler approach for the assessment of interoperability levels is used, for instance, in the standard IEC 62390 [12] and depicted in Figure 1.2 where different levels are defined specifying the degrees of compatibility and cooperation between profile based devices. A set of device features must be satisfied for each compatibility level.

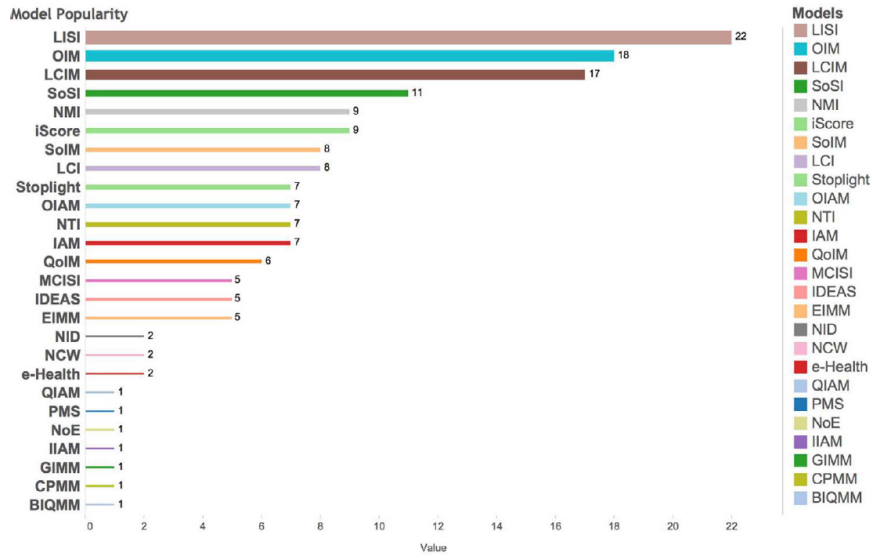


Figure 1.1: Popularity of the models for the interoperability assessment

Standards are the pillars of interoperability in Industry 4.0 because their adoption creates the basis for the interworking between components, devices and people in a value-chain network. Standards for communication and information exchange are required in this context to achieve interoperability [13] because same data can be used in different domain for different purposes. During the last few years, different organizations developed references architectures to align standards in the context of the fourth industrial revolution, like Industrial Internet Reference Architecture (IIRA) for Industrial Internet of Things (IIoT) published in 2015 by the Industrial Internet Consortium (IIC), and Reference Architecture Model for Industrie 4.0 (RAMI4.0) proposed by the german initiative "Platform Industrie 4.0". IIRA enables IIoT system architect to design their own systems using a common vocabulary, a standard-based architecture framework, where the architecture of software systems is described, analysed, and defined to face specific concerns from the viewpoint of different stakeholders. RAMI4.0 is a unified architectural reference model that provides a collective understanding for Industry 4.0 standards that can be regarded as a tool to map I4.0 concepts and use cases. It provides a three-dimensional model consisting of Layers, Life-Cycle & Value Stream,

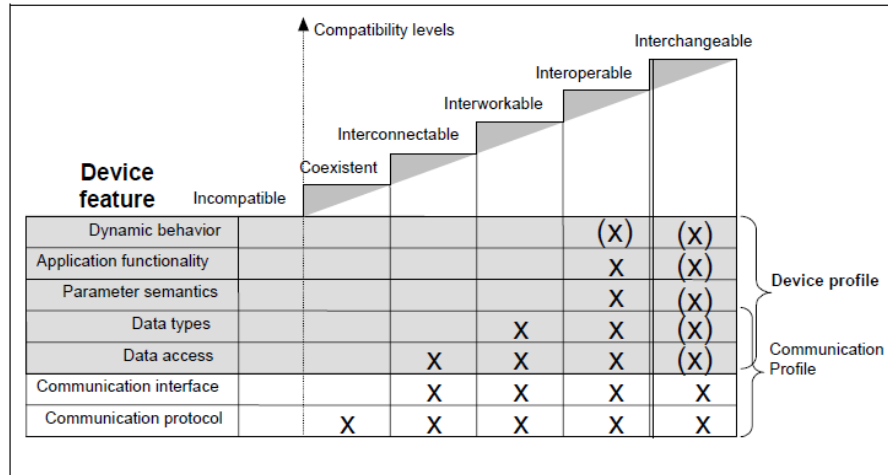


Figure 1.2: Levels of functional compatibility in IEC 62390

and Hierarchy Levels dimensions in the relevant axis, depicted in Figure 1.3. The purpose of this model is to represent a technical object (or asset), and all aspects relevant to it, from its development production and use right through to its disposal [14].

A central theme of RAMI4.0 is the definition of terms, properties and relationships specific to industrial standards [9]. RAMI4.0 defines the concept of Asset Administration Shell (AAS) as the cornerstone of the interoperability. AAS is defined as a digital representation of an asset, which is defined as an entity owned by an organization having either a perceived or actual value for the organization itself (e.g., machines, products, software, licenses, documents). The AAS exposes all information and functionalities relevant to an asset in the form of standardised and semantically-annotated properties and functionalities by means of a Industry 4.0-compliant communication – or I4.0-communication – interface. RAMI4.0 identifies as only possible solution for the implementation of an I4.0-compliant communication the standard IEC 62541 (OPC UA), which is a technology providing secure communication, standardised service sets and an Information Model to structure data with an approach similar to ontologies. Finally, RAMI4.0 defines the conjunction of an AAS with its relevant asset as the so-called I4.0 Component, which realises the concept of CPS in the context of RAMI4.0.

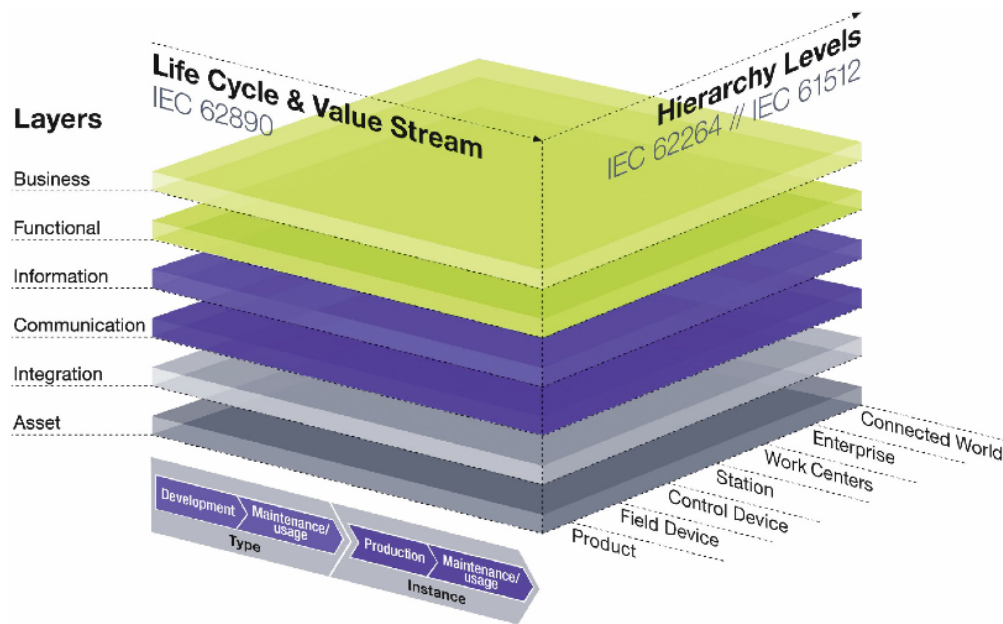


Figure 1.3: Reference Architecture Model for Industrie 4.0

The conducted research described in this thesis focuses on RAMI4.0 as a reference model, trying to figure out how to take advantage of the concept of AAS to achieve interoperability in Industry 4.0 scenario.

1.1 Motivations and Goals

The goals of this thesis is to point out how interoperability can be achieved by means of AAS and how such a digital representation of a physical asset can be exploited in real industrial scenarios to fulfill Industry 4.0 aims. The concept of AAS can play a central role to enable collaboration between different tools coming from different engineering domains. Tools can automatically detect available plant configurations and data models coming from other tools, easing re-configuration of plants or changing in the production process. For this reasons the internal structure of the AAS must be studied in order to understand the insights that makes it the corner stone of interoperability in Industry 4.0 so that it can be implemented and exploited in real implementations and use

cases.

The methodology followed for the conducted research consisted in the study of the available standards and the state of the art relevant to AAS. As said previously, since OPC UA is considered the only solution for communication in Industry 4.0 by RAMI4.0, in a first place we studied the feasibility for the representation of an AAS using the OPC UA Information model. Even though several works in literature already adopted OPC UA to implement AASs in industrial scenario, all of them do not use a common or consistent structure for the internal data of the AAS. This is due to an high-level description of the concept of AAS and to the lack of guidelines for real implementations. The AAS metamodel released in November 2019 in [15] is the first of a series of specifications that try to define a concrete representation of the concept of an AAS. This motivated our research in the adoption of OPC UA as a means for the AAS representation and the exchange of its information in a uniform and standardised manner. We deeply analysed the AAS Metamodel providing insights and reasoning for its representation using the modeling features of OPC UA. The importance of this work is due to the fact that interoperability is achieved allowing standardised information relevant to an asset (i.e. AAS) to be exposed in a standardised manner by means of a I4.0-compliant communication (i.e. OPC UA). The next steps in the research involved the adoption of the previous results to highlight how AAS can be adopted in two common industrial scenarios to enhance interoperability. In particular, we adopted the AAS for 1) the representation of Programmable Logic Controller (PLC) based on IEC 61131-3 and for 2) the definition of a Predictive Maintenance (PdM) model based on AASs.

1.2 Structure of the Thesis

The thesis is structured as follows. In Chapter 2 an overview on the AAS will be provided, highlighting the general features of an AAS and, in particular, the AAS metamodel which is the foundation of some of the research works. Chapter 3 gives an overview on the standard OPC UA providing also the description of some works, based on OPC UA, that we carried out to enhance

interoperability.

Chapter 4 describes in detail our contribution for the definition of an OPC-UA based AAS, providing mapping solutions and insights behind the representation of the main parts of the AAS metamodel using the OPC UA modelling features. This research led to the definition of an OPC UA Information Model for the representation of AAS inside an OPC UA Server. Among the several advantages, this work presents the possibility to exchange AAS information between industrial applications through OPC UA as a communication system. The major contribution consists in the insights behind modelling techniques that should be adopted during the definition of OPC UA Information Models exposing information relevant to the Industry 4.0 specific domains of interest, and comparing eventually the rationales provided with other mapping solutions present in literature.

In Chapter 5 a new approach for the representation of an IEC 61131-3 program inside the AAS of a PLC is presented. Usually, the IEC 61131-3 programs, the PLC where they run, and the real plant controlled by them are closely related. Considering the life-cycle of a production system, the description of PLC programs and their relationships with the physical parts of the plant should be clearly defined, leading to several advantages like, for instance, the definition of testing plant operations, maintenance operations at run-time and re-configuration process of the plant. This research work presents an AAS model able to represent IEC 61131-3 programs and their relationships with physical devices and the PLCs controlling them.

Chapter 6 describes a novel approach based on the adoption of AASs for the representation of PdM solutions, highlighting how the features of AAS are of paramount importance for the definition of flexible manufacturing against a maintenance program. Maintenance is one of the most important aspects in industrial and production environments, and the adoption of vendor-specific solutions for PdM and the heterogeneity of technologies adopted in the brown-field for condition monitoring of the machinery reduce the flexibility and the interoperability required by Industry 4.0. Our approach leverages on the adoption of AASs as a foundation for the definition of a PdM model to cope with the aforementioned issues.

Finally, Chapter 7 summarised the results of the research carried out.

Chapter 2

Overview on Asset Administration Shell

Reference models, like RAMI4.0 introduced in the previous chapter, provide a solution-neutral reference architectural model for applications using technologies advancement in manufacturing process and represent a common structure and language to specify and describe system architectures and thus to promote common understanding and system interoperability. The vision of Industry 4.0 encompasses a massive digitalization process where every asset from the physical world must be represented in the information world by a digital and uniquely-identifiable counterpart [14]. RAMI4.0 refers to such an entity with the name of AAS, but in literature and other reference architectures like Industrial Internet Reference Architecture (IIRA), it is referred with the name Digital Twin (DT). The conjunction of the physical asset with the AAS representing it in the digital world realises the concept of CPS in the form of an I4.0 Component, as specified in [14]. In addition, the AAS provides I4.0-compliant communication with the other I4.0 components in the value-chain network.

As the information model and the interface of AAS are both standardised (even though definition of specifications are still in progress), AASs can be exploited to cope with all the heterogeneous systems available in the industrial environment (industrial silos) [16]. Therefore, since AAS is an abstraction

providing a common structure for the information relevant to assets and a common way to exchange such information [43], it enables the cooperation of assets based on different technologies. This chapter provides an overview on the main concepts of I4.0 Component, AAS and the AAS metamodel introduced in [15].

2.1 I4.0 Component and the role of AAS

In the context of Industry 4.0, an asset is defined as an entity with a value for an organization, and this can be a machine, a product, but even a non-physical entity like a product type, a software, a documents, or a license. One central concept of Industry 4.0 is that assets can be combined together in any way, and these assets are formally described in sufficient detail to be used in the digital world. This solution provides sufficiently generic descriptions for assets configuration but, by means of an increasing degree of detail, it allows more specific descriptions too. To virtually represent configuration of assets and all the relationships between them, the **principle of recursive description of assets** [14] is used, which consists to characterise an asset as follows:

- the structural description must comply with RAMI4.0;
- the configuration describing the connections between two or more assets defines a new asset;
- components of an asset can in turn represent assets;
- the asset description is provided as a structured information in an AAS, which acts as a virtual representation of the asset.

Following this principle, every configuration can be represented in the digital world with any degree of granularity by describing structured assets and any combination thereof using RAMI4.0, and thus the AAS.

As already said, the conjunction of the AAS and the relevant asset forms the so called **I4.0 Component**, which realises the concept of CPS in RAMI4.0,

and thus identifies uniquely and globally identifiable participants able to communicate to each other, like depicted in Figure 2.1.

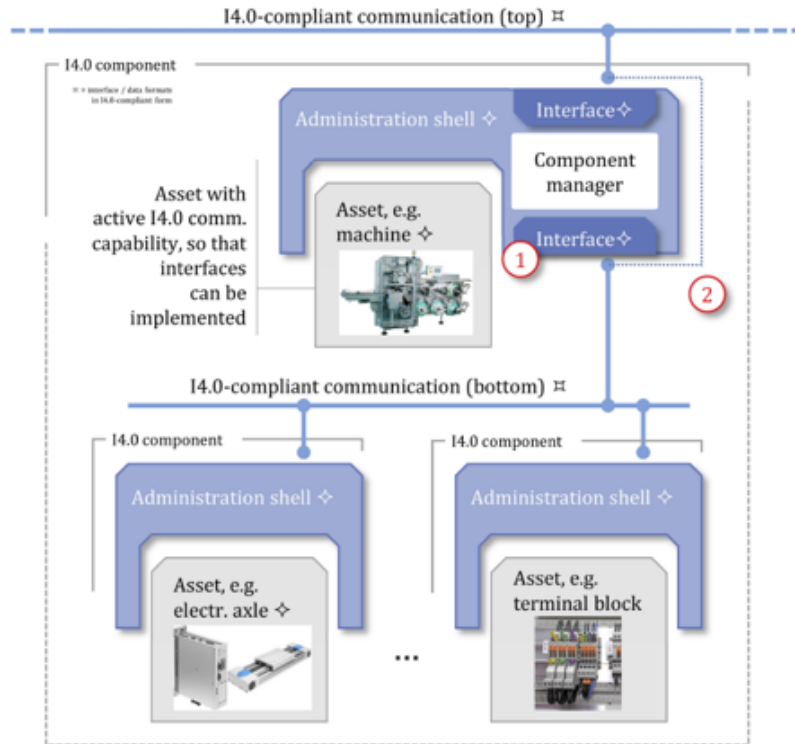


Figure 2.1: I4.0 Components communicating each other by means of an I4.0-compliant communication.

An I4.0 component must satisfy the following properties to represent its relevant asset in the information world:

- being clearly identifiable as an entity;
- being a *type* or an *instance* in a specific point of its life-cycle (e.g. a model of sensor or a real instance of that specific model of sensor);
- being able to communicate by means of an I4.0-compliant communication;
- being a representation of an asset by means of information (e.g. properties, operations).

- having a technical functionality, but its not mandatory;

The AAS is the entity that convert an asset in a I4.0 Component, and it is defined as “the virtual, digital and active representation of an asset”. The AAS covers the first 4 layers of RAMI4.0 (i.e., Business, Functional, Information, Communication) and partially the Integration layer, as shown in Figure 2.2. The AAS enables omnipresent communication and common understanding between hardware and software components and humans through their virtual representations.

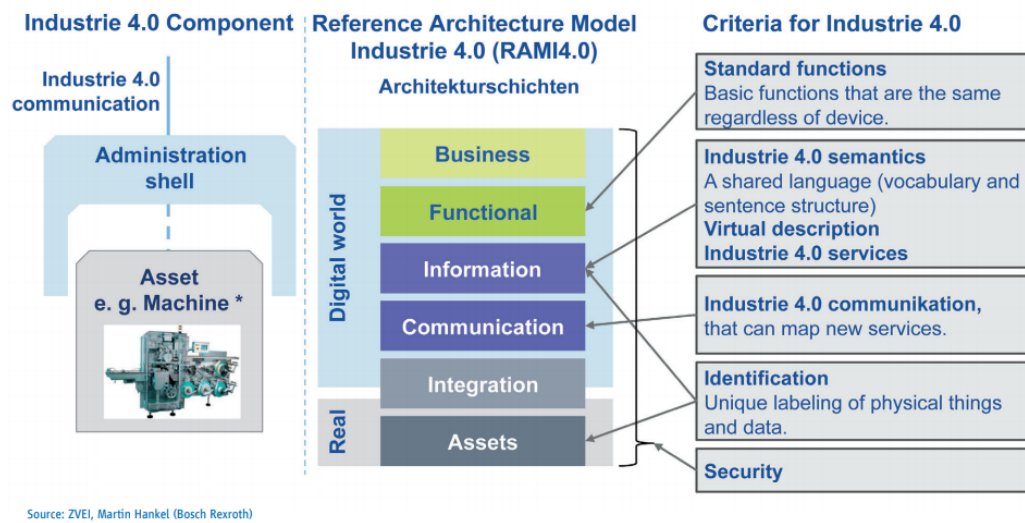


Figure 2.2: Comparison of AAS with RAMI4.0 layers.

2.1.1 Composite I4.0 Component

One of the main features that an I4.0 Component provides is “nestability” as discussed in RAMI4.0. This means that a machine may be seen as a composition of its parts, which in turn can also be I4.0 components. The AAS of the composite component reflects the composition relationship referencing the ASSs of its components. This reflects an application of the principle of recursive description of assets described previously. In a composite component, the AAS contains properties, functions and the status of the composite asset. Data and functions of the individual I4.0 components that has been

integrated in the composite I4.0 component can still be accessed via their relevant AAS. The integrated I4.0 components are referenced by the composite component, as shown in Figure 2.3, while the corresponding assets are called “self-managed assets”. Assets without an own AAS (named *anonymous assets*) can still be part of a composite component, in which case they will be called “co-managed assets” [17].

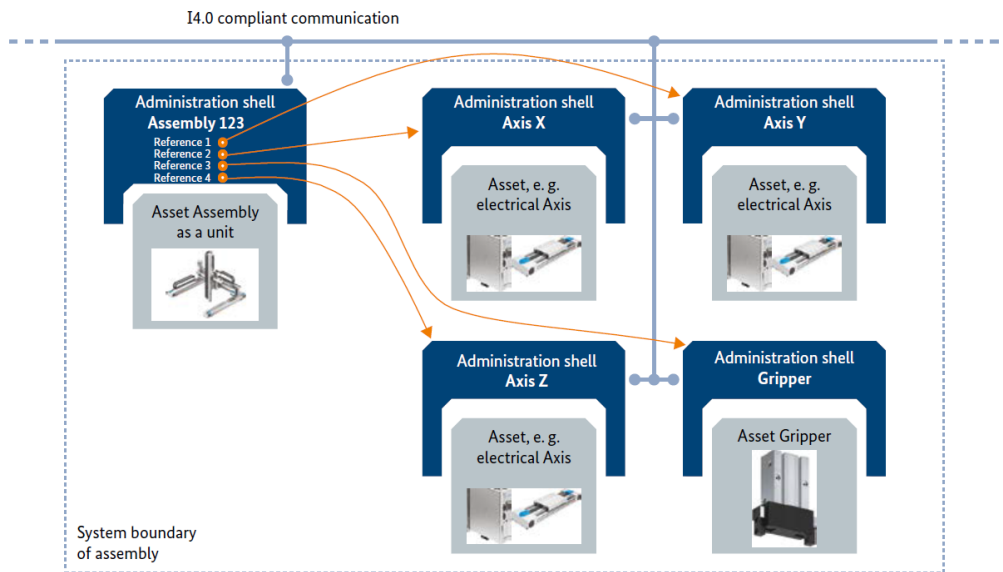


Figure 2.3: A Composite component.

2.1.2 Asset Type and Asset Instance

An asset can be defined with a state in a particular time and location through its lifetime, which is distinguished in **type** and **instance**. Both asset type and asset instance are unambiguously identifiable. An asset type defines a set of properties and functions that are characteristics for all the instances of that specific asset type. For example, a model for a temperature sensor is an asset type, which defines all the features for such a sensor and which all the sensor instances must comply to. An asset instance, instead, is a physical asset characterised by the properties of its type. An instance always

maintains a string and unambiguous relationship with its type throughout the whole life-cycle. Figure 2.4 summarises the life-cycle and the role of a type and an instance.

Fase		Descrizione
Type	<i>Development</i>	The asset type is defined, and its properties and functionalities are defined and implemented. All the artefacts (internal) are created (CAD files, schematics, software) and associated to the asset type.
	<i>Usage/Maintenance</i>	External information associated to the asset are created, like technical data sheets or marketing information. The selling process begins.
Instance	<i>Production</i>	Asset instances are created/produced on the basis of information of the asset type. Specific data related to production, logistics, quality and test are associated to the asset instance.
	<i>Usage/Maintenance</i>	The usage phase by the purchaser of the asset instance. Usage data are associated to with the asset instance and might be shared with other partners in the value-chain, like the manufacturer of the asset instance. Data about maintenance, re-design, optimization and decommissioning of the asset instance are also included.

Figure 2.4: Life-cycle phases and roles of type and instance.

2.2 Structure of the AAS

The internal structure of the AAS is described in a very high-level of abstraction in RAMI4.0 and in documents from Platform Industrie 4.0, like [18], as depicted in Figure 2.5. It is composed by an **Header** and a **Body**; the former contains all the information regarding the identification of both the AAS and the relevant asset, whilst the latter contains all the inherent information of the asset in the form of properties and functions (also referred as operations). Properties are defined as classified and mutually independent characteristics of a system that can be associated with values [19]. Functions, instead, model the capabilities and the actions that an asset performs. Both properties and functions are used to describe the functionalities of an asset and are grouped together under so-called **Submodels**. Each submodel describes a specific aspect relevant to an asset like, among others, energy efficiency, positioning,

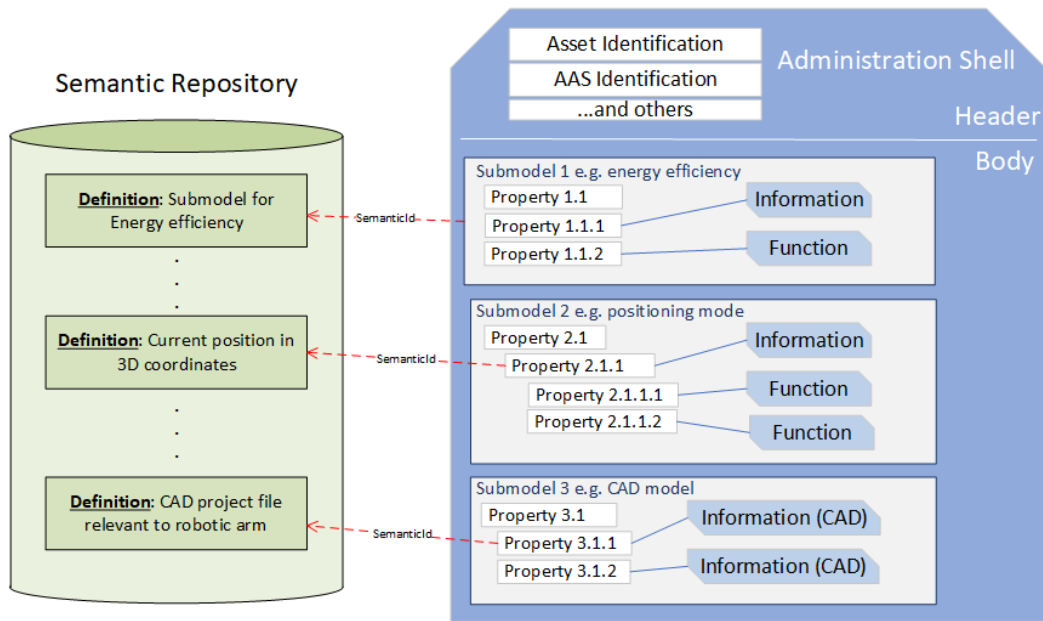


Figure 2.5: Structure of an AAS from an high-level point of view.

documentation, drilling, and maintenance. The aim is the standardization of a submodel for every aspect of an asset [20], but up to know no standardised submodel has been provided yet, even though several proposals start to appear in literature, like in [21], [22], and [23].

Properties inside a Submodel must be semantically annotated so that the inherent meaning of the value carried by properties cannot be ambiguous. For instance, the property “height” of a drilling machine is different from the property “height” of a pipe. Properties inside submodels features references to external dictionary that defines their semantics. Such dictionaries are based on the standard IEC 61360 for the property definition, and are hierarchically organised by increasingly detailed categories (e.g. Electric components → Electric Motors → Stepper Motor). Usually each category contains the definitions of properties allowed for assets belonging to that category. Examples of such dictionaries are IEC Common Data Dictionary (IEC CDD)¹ and eCl@ss². To each property of these dictionaries is associated a unique and

¹<https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset?OpenFrameSet>

²<https://www.eclass.eu/en/index.html>

globally identifier.

Properties or functions composing an AAS contains a special attribute (*semanticId*) pointing to a semantic definition contained in an external semantic repository, as depicted in Figure 2.5. The term “semantic repository” is used here as a generic name identifying any sort of database or catalogue where all the semantic definitions reside. For instance, IEC CDD and elC@ss can be considered semantic repositories. In Figure 2.5, semantic repository is represented as a single unit just from a logical point of view, since in practice a semantic repository can be constituted by different databases or dictionaries located in different parts of the IT infrastructure.

2.3 Implementation variants of an AAS

The AAS is a software entity that can be used in different implementation variants. An AAS can be provided in three different forms: 1) the passive AAS in file format, 2) the passive AAS with an Application Programming Interface (API), and 3) the active AAS [13].

The **passive AAS** in a file format is what is mainly described in [15], where it is presented as a common exchange file format (XML or JSON) between different partners of a value-chain network. Such an AAS is not able to communicate over an I4.0 communication channel and cannot interact with other AASs; it enables only a standardized information exchange of the information relevant to an asset during its life-cycle.

The **passive AAS with an API** provides essentially the same information of (1) but with the difference that its internal data are accessible only via a CRUD-oriented interface (e.g. REST). It follows that, unlike (1), such variant of AAS is accessible via the communication network. It is worth noting that (2) is a reactive entity in the sense that it can respond to requests coming from external clients but it cannot take any initiatives or establish communications with other AASs, hence the name “passive AAS”.

In addition to a CRUD interface, an **active AAS** can be part of an horizontal protocol-based interaction as specified in [24]. Active AASs interact to achieve some goals and are the foundation behind the concept of Plug-and-

Produce in Industry 4.0 [25]. The guideline VDI/VDE 2193 describe a I4.0 Language that can be adopted between the interaction of AASs, as discussed in [26], [27] and [28]. The active AAS relies on some internal algorithms implementing some business logic. For instance, a new I4.0 component inserted in a production line can interact with the AASs of similar devices nearby to receive configuration parameters, avoiding the manual configuration performed by a human operator.

In Figure 2.6, all the variants of AASs are compared placing them in the RAMI4.0 layers. Both AAS as a file and AAS with API are classified

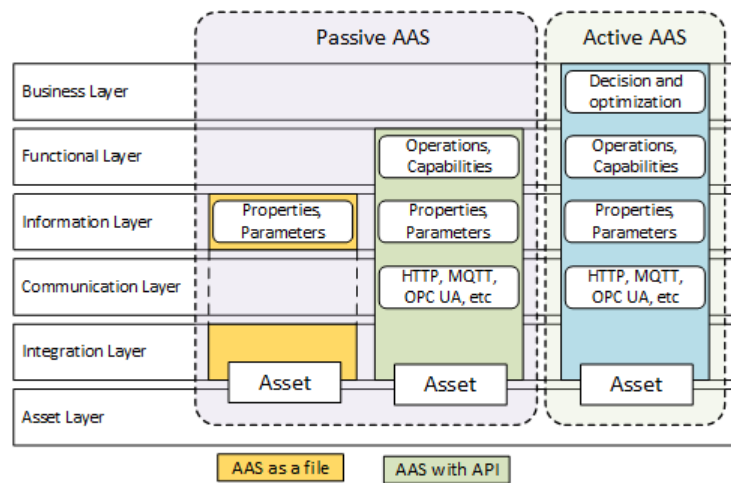


Figure 2.6: Comparison of the variants of the AAS in relation to the layers of RAMI 4.0.

as passive AAS. One of the main differences between the two variants is the lack of communication in the former. Besides being accessible from the communication network, an AAS with API provides an interface needed to access the information structured in the AAS. Since a passive AAS with API is necessarily embedded on computing device, it is possible that such an AAS can execute some operations (e.g. drilling, cutting, turning on, etc) when requested by an external client. Hence the presence of this type of AAS in the Functional Layer. An active AAS, instead, covers all the aspects owned by the passive AAS, providing in addition the implementation of some algorithms exposing some business logic. This variant of AAS is a proactive software

entity and may not require any external solicitation to execute some decision or optimization process. All the behaviours of the AAS is scheduled according to its internal business logic, usually represented by means of state machines.

Depending on the nature of the AAS, different solution for its deployment may exist. In general, RAMI4.0 do not put any constraints about the location of an AAS. A passive AAS in file format may be stored in a database, whilst both passive AAS with API and active AAS may be embedded in a smart device or deployed in a completely different location (even though a connection with the asset may be maintained). Furthermore, it is worth noting that different parts of an AAS may be separated across the infrastructure, thus the AAS is not required being a single monolithic software entity [14].

2.4 Asset Administration Shell metamodel

In November 2019 the first version of [15] was released providing a first specification of an AAS metamodel to structure an AAS in a uniform and consistent manner in order to facilitate the exchange of asset information between partners of a value chain. The AAS metamodel is presented as an UML class diagram and used as a reference model to structure the information inside an AAS and to create relationships between the elements composing it. In particular, the metamodel:

- defines a collection of classes used to structure information (referred in the remainder as *entities*);
- defines a collection of abstract classes (referred in the remainder as *common classes*) used to defines aspects (*attributes*) common to different entities of the metamodel;
- is based on the data types defined in the XML Schema Definition (XSD) standard³. Furthermore, it defines new data types when the ones defined in XSD do not satisfy the needs;
- uses an identification mechanism that can be either absolute or relative;

³<https://www.w3.org/TR/xmlschema-2/>

- defines a referencing mechanism between internal and external entities of the AAS;

The main class of the AAS metamodel are shown in the UML class diagram in Figure 2.7.

2.4.1 Common Classes

Common classes are abstract classes used to describe aspects shared by metamodel entities; they are Identifiable, Referable, HasKind, HasSemantics, and HasDataSpecification. Multiple inheritance is allowed in the AAS metamodel, therefore entities in the AAS metamodel can inherit from more than one common class. In the remainder of this thesis the name of a common class will be used as adjective to refer a particular metamodel entity inheriting from that common class, e.g. "HasKind entity" refers to an entity inheriting from the common class HasKind.

Identifiable and Referable

In the digitalization process of an asset, everything should be unambiguously identified: parts, products, people, software and services. But in order to achieve interoperability, relationships between entities shall be identifiable too. For this reason, the AAS metamodel makes a distinction between elements that are identifiable, referable or none of both. An Identifiable entity can be identified with a globally unique identifier, which makes possible to refer such an entity in any context.

Table 2.1 summarizes the attributes of identifiable entities. The type *Identifier* is a new structured type defined in the metamodel and consists of a string field (*id*), and a second field (*idType*) of type *IdentifierType*; *IdentifierType* is an enumerative type specifying the following values: IRDI, URI and Custom. Values of identifier type are used inside the attribute *identification* of Identifiable entities, as shown in Table 2.1.

In Table 2.2, the attributes of a Referable entity are summarised. A referable entity provides a short identifier (*idShort*) that is unique only in the

Attribute	Description	Type
administration	Administrative information	AdministrativeInformation
identification (mandatory)	Global unique identifier	Identifier

Table 2.1: Attributes of Identifiable.

context of its name space. The name space for a referable entity is defined as its parent element that is either referable or identifiable. It is worth noting that Identifiable entities are also referable but the vice versa is not true.

Attribute	Description	Type
idShort	Identifying string of the entity within its name space	String
category	Additional meta information about the class of the element, affecting the expected existence of certain attributes	String
description	Description or comment of the element	String
parent	Reference to the next referable parent element	Reference

Table 2.2: Attributes of Referable.

Reference mechanism of AAS metamodel The AAS metamodel provides the entity Reference in order to establish relationships between entities composing the AAS. References can be used to point entities external to the AAS and not only the ones defined internally. Reference can point only to entities that are at least Referable.

The AAS reference entity features the attribute *key*, which is logically structured as an ordered list of keys where each element refers an entity by means of its identifier. The structure of this key list resembles an URI structure, where the first key refers to the root element and every following key identifies the next element in the hierarchy leading to the referred element, identified by the last key of the list.

Each key in the list belongs to a structured type named *Key*. The mandatory fields of this type are *local*, *type*, *value* and *idType*. The attribute *local* specifies whether the referred element is local to the AAS or not. The attribute *type* specifies the class name of the referenced entity; its value belongs to a custom type named *KeyElements*, which is an enumeration whose values are the names of the entities in the metamodel. The attribute *value* is a string containing the identifier of the entity referred by the key. The attribute *idType* describes the kind of identifier used in attribute value; its value is of type *KeyType* which is an enumeration whose values are the kinds allowed for both global and local identifiers, i.e., IRDI, IRI, Custom, idShort and FragmentId.

HasKind

The common class HasKind identifies all those entities that can have the double nature of template and instance. Templates define common features for all its instances. The HasKind common class features an unique optional attribute, named *kind*, which can take either the value “Template” or “Instance”.

HasSemantics

The HasSemantics common class identifies the entities of the metamodel that can be described by means of a concept. A HasSemantics entity owns a reference to another external entity that describes its meaning in a proper manner. HasSemantics defines only one optional attribute, *semanticId*, which is a reference to the semantic definition of the element.

HasDataSpecification and DataSpecification

An entity that allows its instances to contain additional attributes to those already defined in the entity itself is identified as HasDataSpecification entity. Such an entity contains one or more References to so-called Data Specification Templates (DST), which are used to define the additional attributes. The common class HasDataSpecification defines only the optional attribute *hasDataSpecification*, which contains References pointing to the DSTs eventually

used.

Even if [15] specifies that DST does not belong explicitly to the metamodel, its internal structure is described using an entity named `DataSpecification`; it is `Identifiable`, so that its identifier can be used inside references. It consists of an entity named `DataSpecificationContent` containing the definition of the additional attributes.

2.4.2 Main Classes

`AssetAdministrationShell`

The main element of the entire AAS metamodel is represented by the `AssetAdministrationShell` entity, which is both `Identifiable` and `HasDataSpecification`. It provides more attributes based on how an AAS is structured [18] and summarised in Table 2.3.

Attribute	Description	Type
<code>derivedFrom</code>	A Reference to the AAS the current AAS was derived from	Reference
<code>asset (Mandatory)</code>	A Reference to the Asset entity	Reference
<code>submodel</code>	References to the Submodels	Reference
<code>conceptDictionary</code>	One or more Concept Dictionary entities	ConceptDictionary

Table 2.3: Attributes of the `AssetAdministrationShell`.

The attribute *derivedFrom* is used to establish a relationship between two AASs that are derived from each other by means of a Reference. In case of an AAS representing an asset instance, this reference points to the AAS representing the corresponding asset type or another asset instance it was derived from. The same holds for AAS of an asset type as types can also be derived from other types.

Asset

The Asset entity contains all metadata of an asset represented by an AAS. This entity is Identifiable and HasDataSpecification. Optionally, can feature a reference to a Submodel entity describing identification aspect of the asset itself.

It defines an attribute *kind* specifying whether the asset is a type or an instance, in accordance to the asset life-cycle described in Subsection 2.1.2.

Submodel and SubmodelElement

The Submodel entity defines a specific aspect of the asset represented by the AAS. It is used to structure the AAS into distinguishable parts, organizing related data and functionalities of a domain or subject. This entity is Identifiable, HasKind, HasDataSpecification, and HasSemantics. In case of a Submodel with *kind* = “Instance”, the *semanticId* attribute may refer to another Submodel entity with *kind* = “Template”.

Submodel represents a collection of SubmodelElements that are related to the same aspect of the asset identified by the Submodel itself. For this reason, the entity Submodel defines the attribute *submodelElement* that is a composition of zero or more SubmodelElements, as shown in Figure 2.8. A SubmodelElement entity is suitable for the description and differentiation of assets. The SubmodelElement entity is Referable, HasKind, HasDataSpecification and HasSemantics. All the SubmodelElements of a Submodel with *kind* = “Template” are in turn SubmodelElement templates (i.e., *kind* = “Template”). SubmodelElement is an abstract superclass for all those entities composing the internal structure of a Submodel, e.g. properties, files, operations.

In general, a SubmodelElement can contain other SubmodelElements creating an internal hierarchy. The concrete class SubmodelElementCollection (SEC) serves for this purpose as it is defined as a set or a list of SubmodelElements; such collection can be ordered and either allowing or refusing duplicate elements. SEC is a very important entity because it is the only one that allows the internal organization of a submodel, like a folder in a directory.

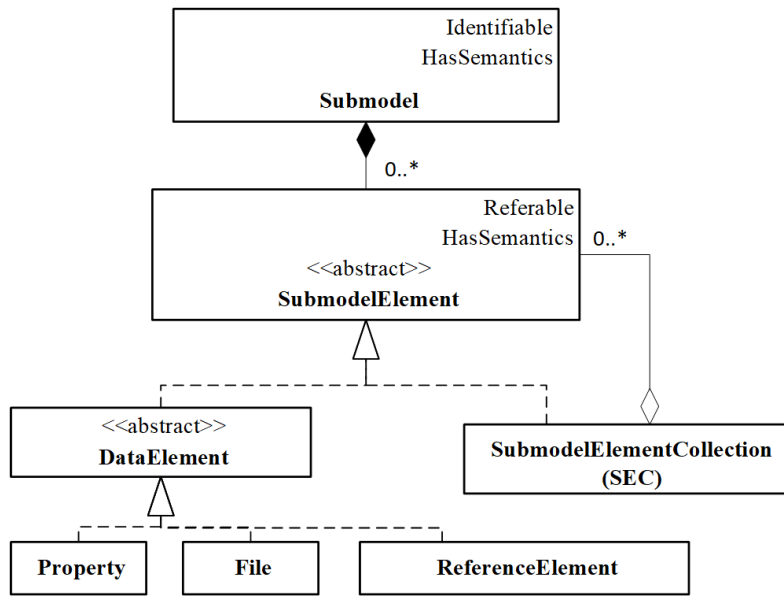


Figure 2.8: Class hierarchy relevant to Submodel structure.

DataElements

DataElement is an abstract class inheriting from SubmodelElement that identifies all the SubmodelElements that are no further composed out of other SubmodelElements.

A Property is a concrete DataElement that is made up by the additional attributes shown in Table 2.4. The attributes *value* and *valueType* are the most important; the latter specifies which kind of data value is contained in the former. This information is necessary to decode such a value.

Attribute	Description	Type
value	The value of the Property instance	ValueDataType
valueType	Data type of the value	DataTypeDef
valueId	A Reference to the global unique id of a coded value	Reference

Table 2.4: Attributes of Property.

Another concrete class of `DataElement` is `File` that represent the location of a real file. Its attribute value is an URI that can represent an absolute or a relative path. Finally, the class `ReferenceElement` is a `DataElement` that defines a logical reference to another element of the same AAS or to a different one, but it may also represent a reference to an external object or entity. For example, a `ReferenceElement` instance may be used to correlate two properties of different AASs. The attribute value of `ReferenceElement` contains an instance of the class `Reference`.

The classes `Property`, `File` and `ReferenceElement` are represented in the class hierarchy in Figure 2.8.

ConceptDictionary and ConceptDescription

`ConceptDescription` is one of the fundamental entities to achieve interoperability because it is used to define the semantics for the other entities inside the AAS metamodel. The entity `ConceptDescription` is `Identifiable` and `HasDataSpecification`. Every element in AAS that is `HasSemantics` should have its semantics described by a `ConceptDescription`, unless a more specific solution is adopted.

The entity `ConceptDictionary` represents a collection of `ConceptDescription` instances. `ConceptDictionary` is `Referable` and it defines an additional attribute named *conceptDescription* that is a composition of `References` pointing to `ConceptDescription` instances.

Typically, a `ConceptDictionary` of an AAS contains only `ConceptDescription`s used by elements within the submodels of the AAS. In certain scenarios, the `ConceptDictionary` contain copies of property definitions coming from external standards, like IEC CDD or eCl@ss. In this case, a `ConceptDescription` containing the entry of the external standard shall be added; for this reason, `ConceptDescription` defines the additional optional attribute *isCaseOf* containing a global reference to an external definition the concept is compatible with or was derived from. `ConceptDescription` should follow a standardized template to describe a concept. The only templates available in the metamodel are used to define both semantics of `Properties` according IEC 61360

and physical unit of measurement.

Chapter 3

Overview on OPC UA

In the past years, the major effort in the process of standardisation of automation software was the access to automation data in devices based on several different technologies, often vendor-specific, like bus systems, protocols, and interfaces. The standard Classic OPC, also known as OPC Data Access, was designated as an interface to communication drivers, providing a standardised way to read or write data in automation devices [29].

The standard OPC UA (IEC 62541) is defined as an evolution of its predecessor introducing a brand new technology that is platform-independent and scalable. OPC UA provides a secure communication protocols that is based on the Client-Server paradigm, and only in last few years a new specification introducing the Publisher-Subscriber mechanism has been released. OPC UA provides more than just a standardised communication channel between entities, but provides also a mechanism for information modelling and a set of standardised services to access data exposed by OPC UA Servers. All these features make OPC UA the interoperability middleware for Industry 4.0; in fact, RAMI4.0 identifies OPC UA as the only available solution to implement the communication layer between I4.0-Components [27, 30].

This chapter focuses on the information modeling features of OPC UA, providing a fundamental background for some of the research conducted and discussed in the remainder of this thesis. Furthermore, this chapter introduce some results we achieved during the research carried out to enhance

interoperability by means of OPC UA in the context of Industry 4.0.

3.1 Information Modeling

The main aim of OPC UA is making the information of an OPC UA Server accessible to several OPC UA Clients. Data are structured inside an OPC UA Server in the so-called *AddressSpace* following the specification of the standard OPC UA base model. The base model provides the set of fundamental “building blocks” – or metamodel – and the rules needed to structure the information in a meaningful manner so that data can be accessible and *understandable* by every OPC UA Client.

The OPC UA Information modeling is based on the following principles:

Object-oriented techniques The *AddressSpace* contains typed instances so that clients can manage instances of the same type in the same manner or work with base types to ignore specialised information.

Types are similar to instances The *AddressSpace* exposes type information in the same manner as instances so that clients can access them to understand how data are organised. This mechanism resembles the one of schemas for relational databases.

Information structured as a graph The *AddressSpace* is structured as a full-meshed graph of nodes connected by references. Since OPC UA allows the definition of hierarchies for nodes and references exposing different semantics, information can be organised in different meaningful ways. This mechanism is similar to ontologies coming from semantic web.

Type extensibility OPC UA allows the definition of new type hierarchies or new subtypes inheriting from existing ones. It allows also the definition of types for references connecting nodes in the *AddressSpace*, specifying brand-new semantics for the relationships between nodes.

No limitations on information representation An OPC UA Server representing a system that already contains a rich information model can expose that model without mapping it to a different model.

Server-side Information modeling OPC UA information models always reside in the AddressSpace of an OPC UA Server, never on an OPC UA Client.

An OPC UA Information Model provides a standard way to structure information inside a server and expose them to clients. The fundamental element used to structure the AddressSpace, and thus defining an Information Model, are Nodes and References. The concept of Node and Reference constitute what is referred as the OPC UA metamodel. The standard OPC UA provides its own base information model which every OPC UA Server must use as a foundation of its own information models. Every Information Model is identified by a Namespace URI; an OPC UA Server exposes in its AddressSpace a NamespaceArray containing all the Namespace URIs relevant to the Information Model adopted in the AddressSpace. Each URI in the NamespaceArray is accessed by means of an integer index named NamespaceIndex. The NamespaceIndex with value 0 refers always to the standard base Information Model and its mandatory for every OPC UA Server.

3.1.1 Metamodel and Base Information Model

Any kind of information is represented inside an OPC UA Server as an OPC UA Node. The set of all nodes inside the OPC UA Server defines its AddressSpace. Nodes belongs to exactly one of 8 NodeClasses, which identify the kind of information a Node represents. Nodes expose attributes containing metadata of the Node; some attributes are common to all NodeClasses, whereas other attributes depends on the NodeClass. The common attributes for all NodeClasses are listed in Table 3.1.

There are 8 NodeClass defined in the OPC UA metamodel: Variable, VariableType, DataType, ReferenceType, Method, View, Object and ObjectType.

Attribute	Description
NodeId	Uniquely Identifies a Node in an OPC UA Server
NodeClass	Identifies the NodeClass of the Node.
BrowseName	Identifies the node during the browsing service of OPC UA
DisplayName	A localized name that can be used to display the name of the node in a user interface
Description	A localized textual description of the Node
WriteMask	Is optional and it is a bitmask specifying which attributes of the Node are writable
UserWriteMask	Is optional and it is a bitmask specifying which attributes of the Node are writable by the user currently connected to the Server

Table 3.1: Common attributes.

Variable It is used to model values of the system inside the OPC UA AddressSpace. A Variable can be a *Property* or a *DataVariable* depending on the VariableType associated to the Variable Node. A Property represent a server-defined metadata associated to another Node and commonly used when metadata exposed from Node attributes are not sufficient. A DataVariable, instead, represent data associated to an Object Node. The Variable NodeClass features the attribute *Value* containing the actual value of the Variable. The attribute *DataTypes* specifies the type definition for the value contained in the attribute Value.

VariableType This NodeClass provides the type definition for another Variable Node. The base model defines two VariableTypes: PropertyType and BaseDataVariableType for Property and DataVariable, respectively.

DataTypes This NodeClass provides the type definition for values contained in the Value attribute of a Variable Node.

ReferenceType Defines the type for References used to connect Nodes inside the AddressSpace. This NodeClass is used mainly for the definition of a semantics for the relationship between Nodes.

Method This NodeClass is used to model callable functions inside the AddressSpace.

Object This NodeClass represents entities in the real world like system components, both hardware and software. An OPC UA Object is like a container for other Objects, variables and Methods. Since an Object does not provide a value of its own, a DataVariable must be connected to an Object for this purpose. For instance, a temperature sensor device may be modeled as an Object whilst the measured temperature value may be modeled as a DataVariable Node connected to the Object.

ObjectType It is used for the definition of a type for Object Nodes. The standard OPC UA allows the extension of the standard base ObjectTypes for the definition of new ones. All the Objects inside the AddressSpace inherit, either directly or indirectly, from the BaseObjectType ObjectType. One of the most important ObjectType defined in the base Information Model of OPC UA is the FolderTypeObject Type which is used to define "Folder" Objects used to organise the AddressSpace in a proper manner.

The OPC UA standard provides a graphical notation to represent some Nodes depicted in Figure 3.1.

The Nodes inside the AddressSpace are connected by means of References creating semantic relationship between them. Reference are classified as *Symmetric* and *Asymmetric*, where in the former the semantics is the same regardless the direction followed in the browsing, whilst in the latter the semantics differs on the basis of the direction of the reference. Furthermore, References can be classified also in *Hierarchical* and *NonHierarchical*.

The semantics associated to Hierarchical References is that of spanning a hierarchy. Such type of References do not forbid loops, therefore it is possible to reach the same starting node following only Hierarchical references. By

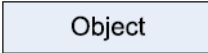

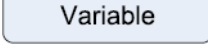
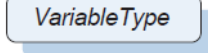
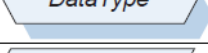
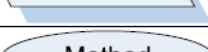

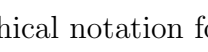
NodeClass	Graphical Representation	Comment
Object		Can contain the TypeDefinition separated by "::", e.g., "Object1::Type1"
ObjectType		Abstract types use italic, concrete types not
Variable		Can contain the TypeDefinition separated by "::", e.g., "Variable1::Type1"
VariableType		Abstract types use italic, concrete types not
DataType		Abstract types use italic, concrete types not
ReferenceType		Abstract types use italic, concrete types not
Method		–
View		–

Figure 3.1: Graphical notation for some OPC UA NodeClass.

the way, it is forbidden having self-pointing Hierarchical references. The base Information model provides some standard ReferenceType for Hierarchical References. In the following some of the most important will be described.

HasComponent and HasOrderedComponent If the source Node is an Object, the target Nodes may be Objects, DataVariables and Methods; the semantics of this Reference is that the source Object is “composed” by the target OPC UA Nodes. If the source Node is a DataVariable, the target Nodes may be other OPC UA DataVariables; the semantics is that the source Variable is “composed” by a set of other Variables. The difference between the two variants is that in the former there is no order relationship in the browse of components, whereas in the latter an order relationship exists and every browse of the components must be returned with the same sequence.

HasProperty This Reference connect the source Node with its Property. The semantics is that the source Node features a property identified by the target Node.

Organizes This reference is used to connect a source Object of type FolderType – also referred as Folder – to other Objects and/or Variables. The semantics is that the source Node organizes the target Node like a container.

HasChild It defines a non-looping hierarchy between Nodes.

Aggregates The semantics of this ReferenceType indicates that a Node “belongs” to another Node.

HasSubtype This Reference is used to connect an ObjectType or a VariableType to another ObjectType or VariableType, respectively. This reference creates the inheritance semantics between Nodes defining types.

NonHierarchical References are used to define some relationship between Nodes that do not span a hierarchy. The base Information model defines several NonHierarchical References; one of them is *HasTypeDefinition* which is used to bind an Object or a Variable to its relevant ObjectType or VariableType.

The graphical notation for References defined in the OPC UA standard is depicted in Figure 3.2.

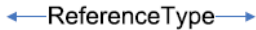

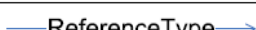




ReferenceType	Graphical Representation
Any symmetric ReferenceType	
Any asymmetric ReferenceType	
Any hierarchical ReferenceType	
HasComponent	
HasProperty	
HasTypeDefinition	
HasSubtype	

Figure 3.2: Notation for References based on referenceTypes.

3.1.2 Type modeling for Objects

The information modeling mechanism of OPC UA uses an object-oriented approach, thus Objects are considered instances of ObjectTypes. This means

that Nodes connected to the ObjectTypes by means of Hierarchical References can be present in the relevant Object representing an instance of that ObjectType.

Even though both Object and ObjectType represent conceptually different things, it is worth noting that from the point of view of the OPC UA metamodel they are both Nodes. For instance, Variables connected under an Object exposes some value related to that Object, whereas Variables connected to an ObjectType have no value to expose (because its just a type). OPC UA defines a specific mechanism to differentiate the nature of Variables and Objects connected to ObjectTypes from the ones connected to an Object; such mechanism used the so-called *ModellingRules* which specify how Variables and Objects connected to an ObjectType behaves respect to an instance of that ObjectType.

Each OPC UA ObjectType must defines which ones among its Properties and Components must or may be present in its relevant instances. ObjectType instances may have some mandatory elements, whilst other elements may be optional. The Reference *HasModellingRule* allows to point out this kind of information for each OPC UA type. For each Variable or Object (henceforward called InstanceDeclaration) referenced by an OPC UA type Node, a HasModellingRule Reference points to a ModellingRule Object as target Node. A ModellingRule associated to an InstanceDeclaration specifies whether a copy of such InstanceDeclaration must be present or not in every instance of an OPC UA type Node. A ModellingRule Mandatory for a specific InstanceDeclaration specifies that instances of the OPC UA type must have a copy of that InstanceDeclaration. A ModellingRule Optional, instead, specifies that instances of the OPC UA type may have a copy of that InstanceDeclaration, but it is not mandatory. Other two ModellingRules exist named MandatoryPlaceholder and OptionalPlaceholder. The difference with the previous ones is that the counterparts of InstanceDeclarations in the relevant instances may be more than one, regardless of the BrowseName of the InstanceDeclaration.

A graphical representation has been also defined for the InstanceDeclaration and ModellingRule Object and for HasModellingRule Reference. The

name displayed inside each InstanceDeclaration is relevant to its BrowseName. The same happens for each counterpart of an InstanceDeclaration relevant to an instance of OPC UA type. In case of an InstanceDeclaration having the OptionalPlaceholder and MandatoryPlaceholder ModellingRule, the BrowseName will be enclosed within angle brackets. Furthermore, a ModellingRule Object and the relevant HasModellingRule Reference are not graphically represented but only shown as a text containing the kind of the ModellingRule Object written within square brackets and put inside the source InstanceDeclaration Node (i.e. [Mandatory], [Optional], [OptionalPlaceholder], [MandatoryPlaceholder]). Figure 3.3 shows an example of the graphical notation just described.

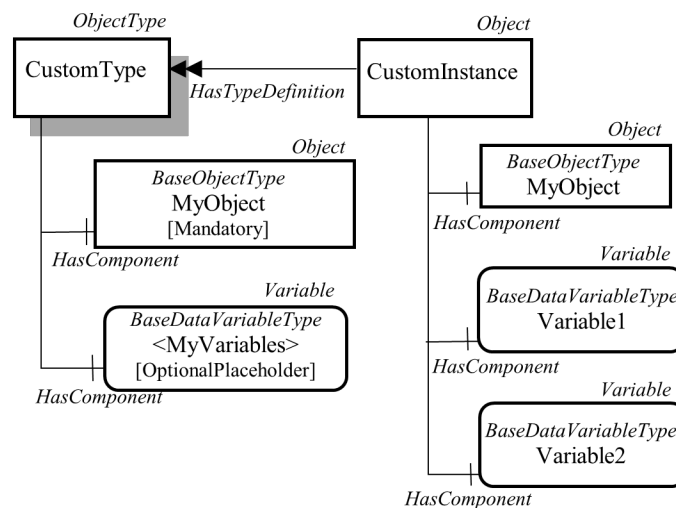


Figure 3.3: Example of graphical representation of InstanceDeclarations, ModellingRule and HasModellingRule References.

Very recently, the OPCFoundation released an amendment introducing a new feature in Address Space model called Interface [31]. An Interface is an ObjectType representing a generic feature that can be used by different Objects or ObjectTypes. HasInterface is a new NonHierarchical ReferenceType; an Object may have more HasInterface References connected to different Interfaces. When an Object references an Interface by means of a HasInterface Reference, it inherits all the InstanceDeclarations exposed by the Interface,

following the same rules used for an Object that inherits all InstanceDeclarations exposed by its ObjectType. The difference with the usual type inheritance is that Interface mechanism allows multiple inheritance, thus an Object can refer multiple Interfaces by means of HasInterface References.

3.2 Data type system of OPC UA

All attributes of Nodes except for the *Value* attribute of Variable and VariableType have a fixed data type. For Variable and VariableType Nodes, the attribute *DataType* is used in combination with the attribute *ValueRank* and *ArrayDimension* (defined only for these two NodeClasses) to define the data type of the *Value* attribute. DataTypes are defined as Nodes in the AddressSpace and the attribute *DataType* of Variable and VariableType contains always a NodeId of a specific DataType Node.

The fact that data types are represented as Nodes in the AddressSpace allows OPC UA Servers to define their own DataTypes, so that OPC UA Clients can access such information and understand how decode values. There are four kinds of DataType in OPC UA: *Built-in*, *Simple*, *Enumeration* and *Structured*. All these kinds of DataTypes are represented in the AddressSpace.

The DataType mechanism of OPC UA is based on an encoding mechanism which is directly implemented in the OPC UA Stack. Such encoding mechanism provides the a set of fundamental data types, named **built-in types** that must be used for the serialization, and thus the transport, of every value of a DataType defined in the AddressSpace. There are 25 built-in types defined in the OPC UA standard, listed in Figure 3.4.

The aforementioned kinds of DataType represented in the AddressSpace will be described in the following, describing also how they are represented against the built-in types provided by the encoding mechanism of OPC UA:

Built-in DataType This DataType defines a set of fixed DataTypes specified by the OPC UA standard and cannot be extended by other Information Models. They provide basic types like Int32, Boolean, Double but also OPC UA-specific types like NodeId, LocalizedText and Quali-

ID	Name	Description
1	Boolean	A two-state logical value (true or false).
2	SByte	An integer value between -128 and 127 inclusive.
3	Byte	An integer value between 0 and 255 inclusive.
4	Int16	An integer value between -32 768 and 32 767 inclusive.
5	UInt16	An integer value between 0 and 65 535 inclusive.
6	Int32	An integer value between -2 147 483 648 and 2 147 483 647 inclusive.
7	UInt32	An integer value between 0 and 4 294 967 295 inclusive.
8	Int64	An integer value between -9 223 372 036 854 775 808 and 9 223 372 036 854 775 807 inclusive.
9	UInt64	An integer value between 0 and 18 446 744 073 709 551 615 inclusive.
10	Float	An IEEE single precision (32 bit) floating point value.
11	Double	An IEEE double precision (64 bit) floating point value.
12	String	A sequence of Unicode characters.
13	DateTime	An instance in time.
14	Guid	A 16-byte value that can be used as a globally unique identifier.
15	ByteString	A sequence of octets.
16	XmlElement	An XML element.
17	NodeId	An identifier for a node in the address space of an OPC UA Server.
18	ExpandedNodeId	A NodeId that allows the namespace URI to be specified instead of an index.
19	StatusCode	A numeric identifier for an error or condition that is associated with a value or an operation.
20	QualifiedName	A name qualified by a namespace.
21	LocalizedText	Human readable text with an optional locale identifier.
22	ExtensionObject	A structure that contains an application specific data type that may not be recognized by the receiver.
23	DataValue	A data value with an associated status code and timestamps.
24	Variant	A union of all of the types specified above.
25	DiagnosticInfo	A structure that contains detailed error and diagnostic information associated with a StatusCode.

Figure 3.4: Built-in types.

fiedName. Such DataTypes are directly mapped with the corresponding built-in type with the same name at encoding level.

Simple DataType This DataType are subtypes of the Built-in DataTypes and are encoded exactly like these last. In other words, a value of this DataType cannot be distinguished from the same value of a Built-In DataType; only accessing the relevant DataType attribute of Variable or VariableType reveals that the value belongs to a Simple DataType. Simple DataTypes are used to defines a semantics for data types; for instance, Duration is a Simple DataType that extends the Double Built-In DataType to define values representing intervals of time in milliseconds. Apart from this, values of Duration and values of Double are encoded in the same way.

Enumeration DataType This DataType represents a discrete set of named values. Values of this kind of DataType are encoded using the built-in type Int32, i.e. an integer. The Node representing an Enumeration DataType features two mutual exclusive Properties named *EnumStrings* and *EnumValues*. The Value attribute of EnumStrings is an Array of LocalizedTexts containing human-readable representations of each enumerated value. The integer value represents the index of the relevant LocalizedText in the array contained in EnumStrings. The value attribute of EnumValues, instead, is used to represent Enumeration values of discontinuous integer values. Nodes of this kind of DataType are subtype of the standard Enumeration DataType Node in the AddressSpace.

Structured DataType This DataType are used to represent structured data and are the most powerful construct specifying user-defined complex DataTypes. Structured DataTypes are mapped by the encoding layer using the built-in type ExtensionObject, which behaves as a sort of container capable of managing the representation of any possible data presenting any type of structure. Nodes of this kind of DataType are subtype of the standard Structure DataType Node in the AddressSpace.

3.2.1 Encoding of DataTypes

An OPC UA Server must always define how values of a certain DataType must be encoded when sended to an OPC UA Client (or decoded by the client). As said previously, OPC UA already defines encoding and decoding rules for Built-in DataType and thus for Simple and Enumeration DataTypes too. For Structured DataType, instead, an OPC UA Server must explicitly specify in its AddressSpace how the value is encoded/decoded. In this way, an OPC UA Client retrieving such information can decode the data when reading the value or can encode the value when it desire to write it on the server.

When an OPC UA Client connects to an OPC UA Server can use one of three possible data encoding for the transmission of information: Binary, XML or JSON [32]. Each data encoding provides the rules needed for the se-

rialization of every possible `DataTypes` using the relevant technology selected. The *DefaultBinary* encoding is the one defined by the OPC UA standard and the one that must be implemented in every OPC UA Server for Binary encoding.

Encoding rules for structured `DataTypes` must be exposed in the `AddressSpace` allowing clients to retrieve them. A `DataTypes` Node representing a Structured `DataTypes` points always to a *DataTypesEncodingType* Object containing the encoding rules for the data encoding of the current connection. The `NodeId` of such an Object is always sent together with every relevant structured value during the transmission so that the client knows which encoding rules has been used by the server to encode the value. If the binary encoding is used for the transmission, the *DataTypesEncodingType* Object is named *DefaultBinary*. The OPC UA Server exposes in the `AddressSpace` a *DataTypesDictionaryType* Variable (referred as Dictionary, for brevity) containing all the description relevant to the selected data encoding for every Structured `DataTypes`.

In Figure 3.5, an example shows this mechanism for the description of a Structured `DataTypes` named `MyType`. The Node `MyType` is connected to the relevant *DefaultBinary* Object of type *DataTypesEncodingType* by means of a `NonHierarchicalReference` named `HasEncoding`. In turn, *DefaultBinary* is connected by means of a `NonHierarchicalReference` named `HasDescription` to a Variable of type *DataTypesDescriptionType*, named in the example `MyTypeDescription`, containing in its `Value` attribute the entry of the Dictionary to find the encoding rules of `MyType`. The Dictionary is depicted in figure as a Variable with name `MyTypeDicitonary` and refers to `MyTypeDescription` with an `HasComponentReference`. As shown by figure, the Dictionary contains in its `Value` attribute a long XML-formatted string consisting of one or more entries called `StructuredTypes`; each entry contains several `Field` elements. A `Field` refers to a component of the Structured `DataTypes` and features a `TypeName` attribute that describes the relevant component. In the example shown by Figure 2, `MyType` is a structure composed by two integer fields (i.e., Built-In Data Type `Int32`).

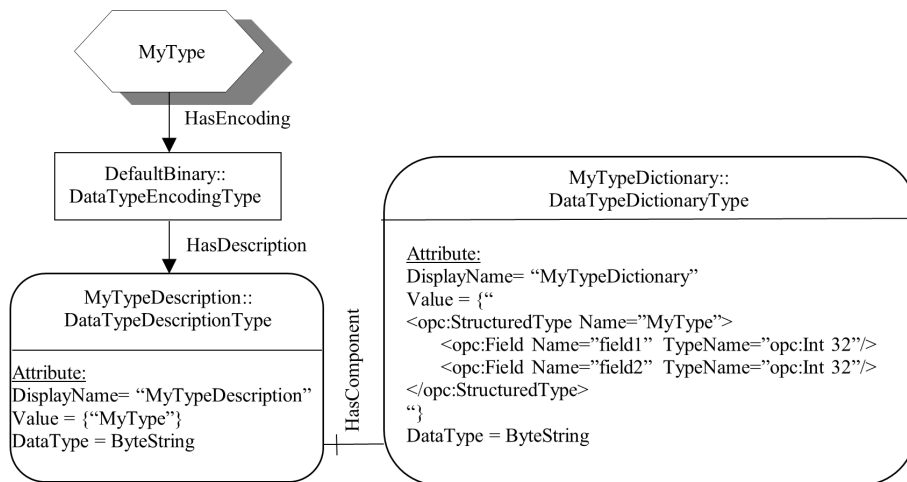


Figure 3.5: Example of OPC UA Structured DataType description.

3.3 Common practices for the definition of new Information Models

OPC UA Information Models structure and expose information coming from different domains of interest, like IEC 61850¹ or IEC 61131-3². In general, the definition of an OPC UA Information Model requires a phase where all the requirements of the original domain of interest are gathered and compared with the standard elements offered by the standard OPC UA Information Model in order to find the suitable solution to satisfy such requirements in the AddressSpace. Often, this process is not straightforward because some concepts from the source domain cannot be mapped directly into OPC UA; in these cases, the definition of new element types extending the original OPC UA elements must be realized.

In the following some of the common practices adopted for this task will be briefly discussed and in Chapter 4 will be discussed how we took advantages of such common practices for our research.

¹<https://opcfoundation.org/developer-tools/specifications-opc-ua-information-models/opc-unified-architecture-for-iec61850/>

²<https://opcfoundation.org/developer-tools/specifications-opc-ua-information-models/opc-ua-for-plcopen/>

3.3.1 Variables and DataTypes definition

Nodes of Variable NodeClass are usually used to represent data. For instance, a Variable may represent the measurement of a temperature sensor or the engineering unit of the measured temperature. In the former case it represent a DataVariable whilst in the latter case it represents a Property. OPC UA defines two main VariableTypes that Variables must inherit from: BaseDataVariableType and PropertyType. A variable of type BaseDataVariables defines a DataVariable and a Variable of type PropertyType defines a Property. As pointed out in [29], it is not always so easy to decide when a DataVariable or a Property should be used when modelling data. In general, DataVariable may be chosen to represent data associated to an Object, whereas a Property may be used to represent some characteristic of a Node that usually cannot be described by means of the attributes of the Node itself.

As said in the previous section, DataTypes are used to define the data type for values contained in Variables, and types can be Built-in, Simple, Enumeration or Structured. In case of a structured value, the adoption of a Structured DataType is not the only solution. One of the common practice consist in modeling a structured value as a complex Object featuring several Variables Nodes as components (i.e., linked to the Object by HasComponent References), each of which represents a field of the structured value. Using both a Structured DataType and a complex Object to represent a structured value is a valid solution, however there are pros and cons for each solution. Briefly, using a Structured DataType is possible to easily access all data at once, whereas using an Object with components requires that each variable component is accessed one by one. In other words, structured DataType provides an implicit transaction context during the information access whereas Object does not, and it must be explicitly managed. On the other hand, using Object in order to access individual data of a structured value is easier than accessing a value belonging to structured DataType. The latter involves an overhead for this kind of operation because all the structured value must be read to retrieve the value of a single field.

3.3.2 Objects and ObjectTypes definition

The NodeClass Object is used to represent entities like entire systems, system components, real-world and software objects. In general, the meanings that an Object can assume are unlimited; the important thing is understanding how Object and its ObjectType are the building blocks of a well-organized AddressSpace and, for each entity that must be represented in the AddressSpace, a relevant ObjectType should be properly defined.

When modelling a system, representation of domain-specific attributes belonging to a particular entity occurs. First of all, the modelling of the attributes must be realized during the definition of ObjectTypes mapping such entities in the OPC UA Information Model. The two most frequent cases consist of attributes containing data values or containing some other complex object. An attribute containing value can be mapped as OPC UA Property (connected with a HasProperty Reference) or DataVariable (connected with a HasComponent Reference) depending on the consideration made previously on Variables. An attribute containing a complex object, instead, may be mapped as OPC UA Object component, which must be connected to the OPC UA ObjectType by a HasComponent Reference; this is legit as this reference represents a part-of relationship and domain-specific attributes may be considered parts of an entity.

OPC UA Objects are also used for the AddressSpace organisation, as explained in the following.

3.3.3 AddressSpace Organisation

When an OPC UA Information Model is used to model a system made up by several entities, a good practice consists of defining an entry point to all the relevant Nodes. Usually, a Folder Object contained in the standard “Objects” folder is used as an entry point to the subset of the AddressSpace containing all the OPC UA Nodes modelling the entities present in the system. All these nodes may be organized in different ways according to the needs to be fulfilled, as explained in the following. A Hierarchical relationship may occur when entities are organized in a way that resembles the same organization

existing between folder and the relevant content in a generic file system. In this case, this relationship may be modelled using a Folder Object modelling the topmost entity and connecting it to the nodes modelling the child entities by means of organizes references. If the hierarchical relationship, instead, resembles an aggregation, particular OPC UA hierarchical references, like HasComponent and HasProperty, may be used to connect the OPC UA nodes modelling the original entities.

Considering a relationship between two entities belonging to two different hierarchies, a common modelling practice in the organization of OPC UA Information Model involves the use of non-hierarchical OPC UA references. For instance, an object belonging to a Folder Object may be linked to the relevant ObjectType (usually belonging to the standard “Types” Folder) by a HasTypeDefinition reference. In general, it is possible to say that non-hierarchical references organizes the AddressSpace from a semantics point of view [29]. Usually ad-hoc non-hierarchical ReferenceTypes must be defined in order to better represent the kind of relationships between entities to be modelled.

3.4 Research activities to enhance interoperability based on OPC UA

Most of our research to enhance interoperability in the context of Industry 4.0 involved the adoption of OPC UA. As already said, OPC UA plays an important role in current industry environments [33] and is considered the most accepted protocol harmonizing the machine-to-machine (M2M) interaction [34]. During these last years, OPC UA has proven to be an effective communications middleware mainly in industrial applications [35].

Although this standard already combines features coming from both industrial and ICT contexts, current literature presents several approaches aimed to introduce ICT enhancements into OPC UA in order to further improve its usability in industrial environments and, in particular, in IoT environments. Some of these approaches are based on the proposal to make OPC UA REST-

ful, due to many advantages of RESTful services in industrial settings and in IoT. In Subsection 3.4.1, we discuss an approach differing from other existing solutions to realise a lightweight OPC UA RESTful interface reducing the complexity of the basic knowledge that must be held by a generic user. As a consequence, such approach allows enhancement of OPC UA interoperability towards resource-constrained devices, especially in IoT environment.

Since there is no universal language for the IoT, interoperability is an imperative requirement also in this context. To overcome this, industry players have come together to form initiatives and consortiums of standards around the various IoT components, including connected buildings, connected home and Industrial IoT. Open Connectivity Foundation (OCF)³ is one of the biggest connectivity standards organizations for IoT, whose specification has been standardised in ISO/IEC 30118. In Subsection 3.4.2, we discuss a novel solution towards interoperability between OPC UA and IoT providing mapping solution for the definition of an OCF Bridge device allowing the seamless interaction between OPC UA-based devices and OCF-based devices.

3.4.1 OPC UA Web Platform

The aim of this research is the definition of a Cloud platform based on web technologies to make accessible OPC UA Servers to IoT Devices. Such platform realised a RESTful interface to masquerade the complexity of OPC UA to constrained devices.

REpresentational State Transfer (REST) is an architectural style defined by Roy Fielding in his Ph.D dissertation [36]; a Web Service based on REST is called RESTful Web Service [11]. The advantages of using a RESTful Web Services has already discussed in [37] and [38], and can be summarised in communication advantages (e.g., reduced communication overhead and possible introduction of caching layers) and system design advantages including stable service interfaces across applications and the use of resource-oriented information models in CPS. These reasons make the adoption of RESTful services a quite frequent choice in the context of IoT architectures [39].

³<https://openconnectivity.org/>

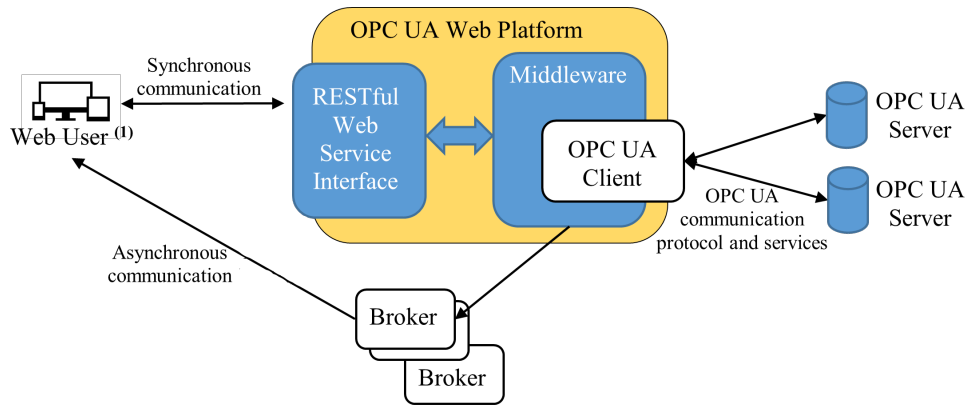
An OPC UA Server exposes to OPC UA Clients several different kind of Information, including Variables, Objects, Methods, type and encoding information, among others. Accessing to such information requires that an OPC UA Client have a full knowledge of the OPC UA data model. Furthermore the number of service requests required to access an OPC UA Server can be high, also in the case of a simple request from the client. This is mainly due to the fact that OPC UA implements a *statefull* communication between client and servers, hence sessions maintaining the status of the interaction is maintained. For instance, just for the reading of a simple value exposed by an OPC UA Server, the OPC UA Client must first create a Session (which first requires the creation of a Secure Channel) and only then make the request to read the value. All these operation requires the execution of several different service calls. It follows that *stateless* approaches, like REST, are more desired in IoT communication where network traffic and resources consumption in general must be low.

Complex data model and intensive message exchanges may create some difficulties for a resource-constrained device acting as client. In IIoT environment, certain applications may be constrained to access data sets of limited complexity and to limit the number of transactions needed to access them. This mainly occurs when applications run on physical devices featuring a set of limited hardware and/or software resources.

For these reasons, in this research we defined a web platform offering to generic clients a lightweighted interface to access OPC UA Servers. We named this web platform *OPC UA Web Platform*. We consider an interface being lightweight when the services it provides requires less message exchanges and semantics to be held from a client.

The platform defined in our research is based on REST architecture due to the advantages proposed in [37], and it is shown in Figure 3.6. It offers to a Web User the access to one or more OPC UA Servers. The generic term “Web User” will identify a generic application which consumes the services offered by the OPC UA Web Platform to access information maintained by OPC UA Servers. Web Users have no knowledge of the OPC UA standard and no awareness of the presence of OPC UA Servers behind the OPC UA

Web Platform. Web User is neither required to be an OPC UA Client nor to implement the OPC UA communication stack (i.e., OPC UA protocol and services).



Note ⁽¹⁾: Web User is a generic application running on a generic (smart) device which consumes the services offered by the OPC UA Web Platform. It is constrained neither to be OPC UA compliant nor to implement OPC UA communication stack.

Figure 3.6: OPC UA Web Platform.

The main parts composing the OPC UA Web Platform are:

RESTful Web Service Interface It accepts requests submitted by an authenticated Web User. Communication between Web User and OPC UA Web Platform is stateless, thus each Web User's request is independent from any stored context on the OPC UA Web Platform, and each Web User's request must contain all the information necessary to the OPC UA Web Platform to accomplish the requested service and to generate the relevant response.

Middleware It performs all the operations needed to fulfil each request coming from a Web User. It is in charge to transform every request done to the RESTful Interface in OPC UA requests that must be forwarded to the relevant OPC UA Servers. For this reason, the Middleware includes an OPC UA Client used to access the OPC UA Servers. Communication between OPC UA Client and OPC UA Servers occurs according the standard OPC UA communication protocol and services.

Communication between Web User and the OPC UA Web Platform is mainly synchronous (based on RESTful web services) but provide also some asynchronous services for monitoring (based on Publish/Subscribe Pattern). A client-server approach based on request-response model is not an ideal solution for networks with multiple servers and clients. An embedded server that handles multiple clients can be an issue as well as a network traffic caused by request and response messages [40]. Publish-Subscribe communication reduce the network traffic and decouple the actors of the communication (loose coupling).

Synchronous communication is realised through the HTTP protocol using a connection encrypted by the Transport Layer Security (TLS), i.e. HTTPS. It realises the encryption of data transmitted between Web User and OPC UA Web Platform, and vice versa. Asynchronous communication, instead, is realised using both MQTT⁴ and Microsoft SignalR⁵.

The authentication is guaranteed using a token-based mechanism for all the request done to the platform interface. In particular, the open standard JSON Web Tokens (JWT)⁶ is the web technology used to implement such token-based authentication.

The Web Platform completely hides the OPC UA Information Model of the OPC UA servers connected and exposes, instead, a simple graph-based resource model where each node maps a relevant Node in an OPC UA Server. The exposed graph is accessible by means of the CRUD functionalities implemented by the RESTful interface of the platform. Every RESTful service is mapped by the platform in one or more OPC UA request by the Middleware. As a result, a device not compliant with OPC UA can still access the information contained in OPC UA server just using simple HTTP requests and web mechanism guaranteeing security in the data exchange. All data are returned to a client encoded in JSON format: the platform is in charge to cope with all the encoding mechanism to translate values of OPC UA DataTypes (also Structured ones) in comprehensive JSON data.

⁴<https://mqtt.org/>

⁵<https://dotnet.microsoft.com/apps/aspnet/signalr>

⁶<https://jwt.io/>

The platform defined for this research has been implemented and made available on Github⁷.

3.4.2 Interoperability between OPC UA and OCF

This research proposes a novel solution to make interoperable OPC UA and IoT ecosystems. One of the major goals of Industry 4.0 is interoperability of industrial applications enabling ICT mainly based on IoT. In fact, accessing data from smart sensor or provide them with new data from IT systems is of paramount importance in this scenario since process data create new values and help the definition of new business models.

To achieve this goal, the integration of communication standards and technologies currently used in both industrial scenario and IoT is required. The standard OCF seems a promising solution to standardise the exchange of information in IoT, therefore we chose OCF as a possible technology to be integrated with OPC UA. A mapping between the information models of both OCF and OPC UA allows the definition of a device with the role of *OPC UA-OCF Bridge* which implements the mapping rules and enables the seamless interaction between OCF and OPC UA devices.

Integration of applications belonging to different ecosystems (e.g., featuring different information models, communication protocols and services) can be realised using several approaches and achieving different levels of interoperability according to the main features of the integration itself [41]. We proposed a solution involving the definition of rules to map each element of the OPC UA Information Model in a corresponding element of the OCF Resource Model, and vice versa. This process includes the definition of new models inside both the OPC UA AddressSpace and OCF Resource Model to realise the correspondences between elements even when the native models do not fit for some representations.

The mapping specifies how each element of the OCF Resource Model is mapped in a corresponding element of the OPC UA AddressSpace of an OPC UA Server, so that we defined a proper extension of the OPC UA Information

⁷<https://github.com/OPCUAUniCT/OPCUAWebPlatformUniCT>

Model. Through this proposal, information maintained by a generic OCF Device can be published by an OPC UA Server providing them to whatever OPC UA Client connected to the OPC UA Server.

The foundation of the research is the definition of a OPC UA-OCF Bridge device depicted in Figure 3.7. The Bridge exposes a proper Server to each of the two ecosystems, i.e. a virtual OPC UA Server for OPC UA Clients and a virtual OCF Device acting as a Server. Both expose the information of the information of one ecosystem to the other.

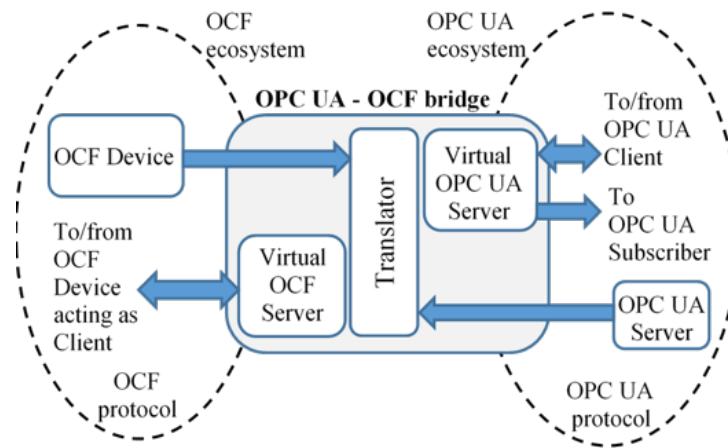


Figure 3.7: Mapping between OPC UA and OCF ecosystems.

The *Virtual OCF Server* is an OCF Device of a proper device type we defined with the aim of exposing information coming from an OPC UA Server to client devices in the OCF ecosystem. The mapping process may involve the entire OPC UA AddressSpace of a specific OPC UA Server or part of it.

The *Virtual OPC UA Server* is an OPC UA Server using the Information Model defined in the research carried out. The Information Model defined allows the mapping of a generic OCF Device in corresponding elements of the OPC UA AddressSpace of the Virtual OPC UA Server in order to expose information coming from OCF to OPC UA Clients. The main component of the Bridge is the *Translator* which realise the mapping rules defined in the conducted research.

The advantages of the proposed solution can be better understand considering the following realistic application scenarios.

A typical application in an OPC UA ecosystem is a Supervisory Control and Data Acquisition (SCADA) system, generally based on OPC UA Clients that exchange data with one or more OPC UA Servers. According to our solution involving the use of an OPC UA-OCF Bridge, a SCADA application can be the client of a Virtual OPC UA Server to collect information coming from the OCF ecosystem (e.g., sensors and actuators in factory automation scenarios). Moreover, the SCADA can also send commands to OCF devices by means of the Virtual OCF Server.

In a typical application of factory, home and build automation realised in an OCF ecosystem is represented by an OCF Device performing functions of controllers and/or data analysis. According to our solution, the OCF ecosystem can receive information coming from an OPC UA Server by means of the Virtual OCF Server of the OPC UA-OCF Bridge. The information retrieved can be used for control and data analytics in addition to the OCF ecosystem information. Finally, an OCF Device acting as controller may send commands to the OPC UA ecosystem by means of the Virtual OPC UA Server.

The OPC UA Information Model for the mapping of OCF Resource Model has been implemented and publicly available on GitHub⁸.

3.5 Publications

The research carried out on OPC UA Web Platform has been presented in international conferences [42, 43, 44] and published in the scientific journal “Computer Standards and Interfaces [45].

The solution proposed for the OPC UA-OCF mapping has been presented in international conferences [46, 47], and published in the scientific journals “IEEE Access” [48] and “Journal of Industrial Information Integration” [49].

⁸<https://github.com/OPCUAUniCT/OPCUA-OCF-Information-Models-Mapping>

Chapter 4

OPC UA-based Asset Administration Shell

Previous chapter highlighted the relevance of both OPC UA and AAS for interoperability in Industry 4.0. The possibility to use OPC UA to realise the concept of the AAS pushed us to assess the state of the art and the possibility to investigate new scenarios taking advantage of this research. For instance, AAS digital information may be exchanged between industrial applications through the OPC UA communication system but, to realise a seamless exchange, information models must be consistent and maintain the information content.

The current version of the AAS metamodel specification [15] provides a proposal of mapping into several technologies, including OPC UA. Since the AAS metamodel, and thus its mapping in OPC UA, is continuously developed and improved, the goal of this research is providing reasoning and insights about mapping choices that aim to maintain the consistency of information during the exchange and conversion of the AAS between partners of the value-chain and avoid information loss.

The results of this research are preliminary to the research works described in the following chapters of this thesis.

4.1 Introduction

State of the art presents lot of different works describing industrial scenarios where implementation of the AAS is realised to accomplish some tasks. So far, different proposals have been presented in literature. One of the most known implementations is openAAS¹, which is based on OPC UA for the internal representation of the AAS. Since openAAS was defined before the release of the AAS metamodel, the internal representation of the AAS realised using this framework is not compliant with the AAS metamodel. Lüder et al. [50] presented an implementation based on companion specification of OPC UA for AutomationML and IEC 61131-3 to structure information and functions of an asset, respectively. Ye and Hong [51] propose an AAS template implementing AASs using OPC UA and AutomationML for a manufacturing system. Another implementation for AAS is described in [52] where authors propose using semantic web technologies like Resource Description Framework (RDF) as a middle layer to support interoperability between the data of legacy systems and data generated by I4.0 components of the AAS. All these implementation cannot cooperate each other because they are not defined in accordance to a same structure for the internal information of the AAS, i.e. the AAS metamodel.

Since AAS is defined as “virtual, digital and active representation of an I4.0 Component in the I4.0 system” [14], in our research we identified OPC UA as a good solution in order to give the “active” aspect to the AAS, providing communication capabilities and secure access to information stored in a passive AAS.

This research work tries to figure out how the fundamental features of the AAS metamodel can be realised in an OPC UA Information model, so that OPC UA-based implementation of the AAS can meet the same requirements as the AAS metamodel.

The approach followed for the realisation of this work involved the de-structuring of the AAS metamodel in its fundamental parts. This process led to a bottom-up solution for the mapping, where fined-grained elements are

¹<https://github.com/acplt/openAAS>

mapped first, then followed by elements derived from their composition. As will be described in the following, this approach realises a consistent mapping for the OPC UA Information Model. Finally, we validated the results in a use case involving an assembly system which uses an OPC UA-based Operator Support System (OSS) reading the AASs stored in an OPC UA Server to retrieve information for the parts of the assembled products.

4.2 Mapping AAS metamodel into OPC UA Information Model

In Chapter 3, some of the common practices adopted for the definition of an OPC UA Information model has been discussed. Such common practices will be applied for the mapping of the AAS metamodel into the OPC UA Information Model, providing the reasoning behind the main decisions that must be taken for the mapping, and pointing out pros and cons when different strategies can be adopted.

4.2.1 Mapping AAS Entities

In the process of defining an OPC UA Information Model, one of the most important design decisions to be taken is choosing the right OPC UA NodeClass to map the main elements of the domain-specific information model to map. In our case, the first step in the conducted research consisted in deciding which NodeClass should be used to map each main element of the AAS metamodel.

Entities in the AAS are defined as classes specifying some attributes. Entities can be classified in entities structuring the AAS (e.g., AssetAdministrationShell, Submodel, and Asset) and entities defining types for attribute values (e.g., Identifier, Key). Attributes, instead, can be classified as attributes containing values, attributes realizing composition and attributes containing AAS References. In particular, attributes realizing composition may contain also AAS References.

As said in chapter 3, since Objects structure the AddressSpace, it seems reasonable using Objects for the representation of the main entities of the AAS metamodel structuring the AAS. To semantically distinguish OPC UA Objects mapping AAS entities between each other, declaration of ObjectTypes for each metamodel entity is needed. For instance, an ObjectType `AASType` may be defined to represent all Object Nodes mapping an `AssetAdministrationShell` entity; an ObjectType `AssetType` can be defined for all Object Nodes mapping an `Asset` entity. In general, for all entities of the metamodel constituting the AAS structure, an ObjectType should be properly defined. Mapping proposed in [15] seems based on the same assumption, in fact Objects map main entities, providing naming convention for the relevant ObjectTypes.

AAS entities in the metamodel defining new types for attribute values (from now on referred as type in the context of AAS metamodel) often realize structures and enumerations and thus can easily be mapped as OPC UA DataTypes because, as said in chapter 3, they can be both Structured and Enumeration. For instance, the AAS metamodel defines two main types for the identification of entities: `Identifier` and `IdentifierType`. The former is a structured type and the latter is an enumerative type. `Identifier` is composed by a string field (`id`), and a second field (`idType`) of type `IdentifierType`. Values of `Identifier` are used inside the attribute identification of `Identifiable` entities, as shown in Table 2.1. OPC UA Structured DataType and Enumeration DataType can be used to map `Identifier` and `IdentifierType`, respectively. A possible mapping solution is depicted in Figure 4.1.

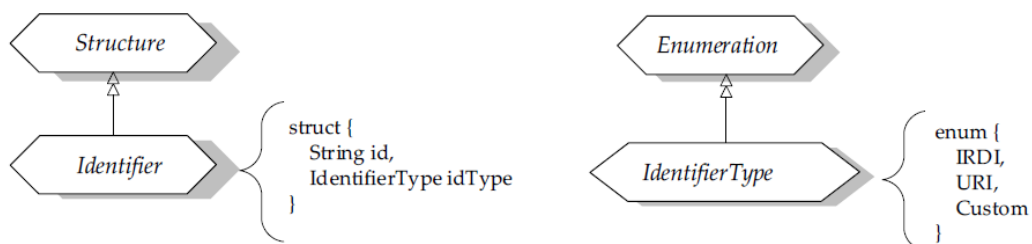


Figure 4.1: Identifier and IdentifierType mapped as DataTypes.

In [15] mapping solutions for type entities vary from case to case, leading to an inconsistent mapping approach because no unique rule seems to be adopted for this kind of entity. For instance, unlike our approach, Identifier is mapped with an ObjectType named AASIdentifierType structured with two Properties: id and idType. The two solutions are quite equivalent but, of course, they will lead to different implementation strategies for entities featuring attributes containing values, like the identification attribute inherited by identifiable (containing Identifier values), as will be detailed in the remainder of this section.

Since entities structuring the AAS has been mapped as OPC UA Object and the relevant ObjectType, the mapping of the attributes must be considered as the very next step. Considering attributes containing a value (from now on referred as value attributes), they describe features of the AAS entities like, for instance, the attribute *description* (see Table 2.2) of a Referable entity whose value is a brief description of the entity itself. Our solution to map type entities with DataTypes led us to choose OPC UA Properties for the mapping of value attributes. In fact, Properties representing such attributes may contain values encoded using the DataTypes modelling the relevant AAS type entities. For example, considering the attribute *identification*, an OPC UA Property can be used to map such an attribute, where the DataTypes shown by Figure 4.1 are internally used by the property to represent the relevant value.

In [15], Identifier has been mapped using an ObjectTypem named AASIdentifierType, setting a constraint on the kind of NodeClass that can be used to map the attribute *identification*, which according to this mapping solution can only be mapped as a component of another Object (i.e., the Object modelling the entity featuring the attribute *identification*). In our opinion the semantics of OPC UA Properties better reflect the meaning of value attributes than semantics of an OPC UA component, which represents a part-of relationship. In fact, the identifier of an entity cannot be considered a part of the entity but an inherent information of the entity itself.

Attributes of AAS entities reflecting composition (from now on referred as composition attributes) contain a collection of other entities. Unlike value

attributes, composition attributes do not contain values but instances of other AAS entities. An example of composition attribute is *conceptDictionary* of the entity *AssetAdministrationShell* (see Table 2.3), which contains a list of *ConceptDictionary* entities related to the AAS. Some solutions to map in OPC UA the *conceptDictionary* attribute are depicted in Figure 4.2. The solutions shown can be generalised to all composition attributes.

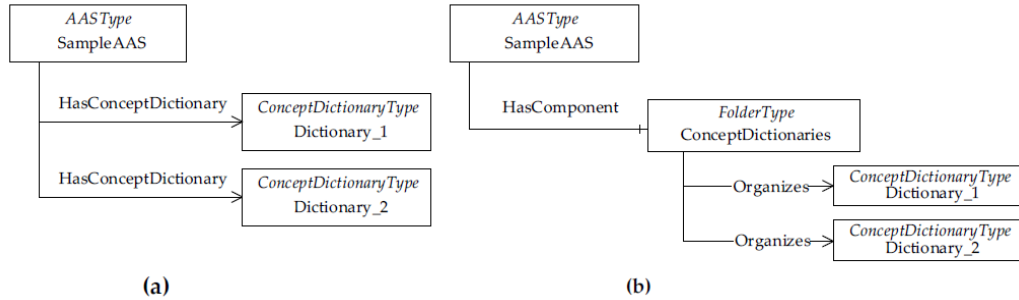


Figure 4.2: The (a) basic and (b) optimized mapping solutions for attributes defining compositions.

Since *ConceptDictionary* is an entity structuring the AAS, in accordance to our decision to map structural entities with Objects, all the *ConceptDictionary* entities contained in the *conceptDictionary* attribute are mapped as OPC UA Objects; furthermore, this solution may be realized defining a new Hierarchical ReferenceType (which could be called “*HasConceptDictionary*” as shown in Figure 4.2a) and using References of this new type to connect the Object mapping the *AssetAdministrationShell* (“*SampleAAS*” Object in the Figure 4.2) to the Objects mapping the *ConceptDictionary* entities contained in the attribute *conceptDictionary* (“*Dictionary_1*” and “*Dictionary_2*” in Figure 4.2). The reasoning behind this mapping solution consists in using ad-hoc defined hierarchical references to represent the list of the entities contained in the composition attributes. In [15] this approach is adopted but no specific ReferenceType is defined for the mapping, using *HasComponent* References instead. The solution here proposed requires the definition of new Hierarchical ReferenceTypes (that can inherit from *HasComponent*) to semantically enrich the connection between an object and its components. The

use of ad-hoc defined ReferenceType has the advantages to give more clarity about the structure of an Object and provides more filtering options during the browsing of an Object to retrieve the content of the mapped composition attributes. It is worth noting that the solution proposed so far, although it realises the mapping of composition attributes, it has the disadvantage that such attributes disappear in the mapping process (even though its informative content is spread over multiple hierarchical references), e.g., in Figure 4.2a, there is no OPC UA elements representing at glance the attribute *conceptDictionary*. A second solution based on the use of Folder Objects and depicted in Figure 4.2b provides a cleaner solution to map this kind of attribute. Such attributes can be directly mapped as Folder Objects organising the OPC UA objects mapping the entities of the composition using Organize References. In figure, it has been assumed to name the Folder using the plural noun of the mapped attribute (i.e., “ConceptDictionaries” for the attribute *conceptDictionary*).

The last category of attributes of the AAS metamodel are attributes containing AAS references to other entities. This discussion is postponed in subsection 4.2.3. For the sake of clarity, in the remainder of this chapter AAS Reference refers to the referencing mechanism of the AAS metamodel, whilst OPC UA Reference refers to referencing mechanism of OPC UA.

In general, every attribute described for entities in the AAS metamodel is annotated with a cardinality specifying whether the attribute is mandatory or optional for the entity. This behavior shall be maintained when an attribute is mapped either as a property or a component of an OPC UA ObjectType. All the InstanceDeclarations defined for the mapping of attributes in the context of an ObjectType must feature a ModellingRule that must be chosen in accordance to the cardinality of the attribute modeled. This solution seems adopted also in [15].

4.2.2 Structuring the OPC UA AddressSpace

The common practices described in Chapter 3 on structuring the AddressSpace involves the use of Folder Object contained in the standard “Objects”

folder as an entry point into the domain-specific Information Model. Since the entity `AssetAdministrationShell` is the top-most entity in the hierarchy defined by the AAS metamodel, it follows that defining a Folder Object named “Asset Administration Shells” as a component of the “Objects” Folder (see Figure 4.3) respects the aforementioned good practice. Such a Folder is used to organize all the objects that are instances of `AASType` and thus represent AASs. This same solution is adopted in [15] to structure the `AddressSpace` in an OPC UA Server, where a Folder named “AASROOT” is used to aggregate all Object representing AASs. Since all the Objects representing AAS are located under a single folder (i.e., “Asset Administration Shells”), an OPC UA Client can take advantage of this structure to select the desired AAS and to browse all the sub-entities it contains. In the example depicted in figure, the `AASType` Objects are connected to the `AssetType` Objects mapping the relevant Assets. The kind `ReferenceType` (i.e. Hierarchical, NonHierarchical) used for the references connecting the Objects are not important for moment and will be detailed in the remainder of the section.

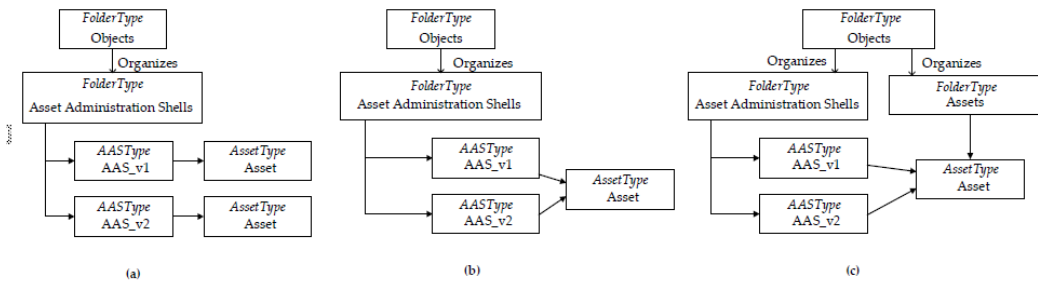


Figure 4.3: The (a) basic and (b)(c) optimized strategies to structure objects in the `AddressSpace`.

Figure 4a depicts a simple scenario where two `AASType` Objects represent two different versions of the same AAS (i.e., “AAS_v1” and “AAS_v2”). This is a naïve solution because, from a logical point of view, the two instances of `AssetType` represents the same Asset. This means that a solution like this led to redundancy due to the mapping of the same entity on multiple Nodes. We analysed the metamodel and found that this situation arises when `Identifiable` entities are involved, because `Identifiable` entities are the only ones that can be

shared across different AASs. An improvement to the first solution described is depicted in Figure 4.3b, where a single node representing an asset is shared across two objects representing the AASs. This simple solution solves the problem of redundant data but has the drawback of having very important data nested inside the AddressSpace structure. In fact, looking at Figure 4.3b, in order to know which assets are contained in the AddressSpace of the OPC UA Server, a client should browse the folder “Asset Administration Shells” and repeat the browsing recursively until it finds an AssetType Object, then go back again and repeat the procedure for all the AASs. This kind of operation leads to a graph traversal which can be very complex in some cases.

To address this problem, a third and more efficient solution for AddressSpace organization involves the creation of a Folder Object as entry point for each kind of Identifiable entity defined in the AAS metamodel. This structures the AddressSpace like a sort of look-up table for identifiable entities, which is an important feature in the context of the AAS environment. The validity of this solution is confirmed by the choices made for the mappings of AAS into XML and JSON provided by [15]. Figure 4.3c shows this solution realised with the creation of a Folder Object named “Assets” to organize all the AssetType objects inside the AddressSpace, in the same manner as the folder “Asset Administration Shells” organizes AASType Objects.

4.2.3 Mapping AAS References

The referencing mechanism provided by AAs metamodel leverage on the entity Reference used to connect other entities composing AASs. AAS Reference is made up by a list of keys (containing in turn entity identifiers) composing an unambiguous path to the pointed entity. The most important aspect to consider in the mapping process of AAS References is guaranteeing that the order of keys constituting the path is respected.

A naïve solution could be to use OPC UA References to map AAS References since both create connections between entities and nodes, respectively, but AAS References contain inherent attributes (i.e., key) whilst OPC UA References, for definition, contain neither attributes nor properties and com-

ponents. Furthermore, AAS References can point to an external source, and such behavior cannot be replicated using OPC UA References, which can point only to nodes contained in the AddressSpace.

The main aim of AAS References is connecting entities structuring the AAS and also external entities. Therefore, we realised that AAS Reference can be mapped as OPC UA Object, similarly to the approach used for entities structuring the AAS. The entity AAS Reference may be mapped with an ObjectType named AASReferenceType, whose structure is depicted by Figure 4.4.

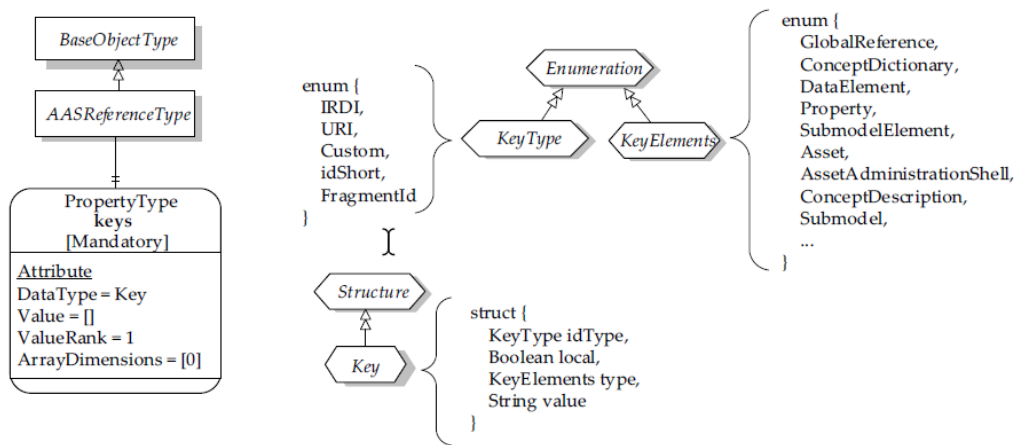


Figure 4.4: Structure of the AASReferenceType ObjectType and relevant DataTypes.

Following the same reasoning behind the mapping solution described in subsection 4.2.1, the attribute *key* of AAS Reference is mapped as an OPC UA Property. Furthermore, it has been assumed to map the *Key* type used for the values inside the attribute *key* as an OPC UA Structured DataType because this type entity is a structure. Since the type entities *KeyType* and *KeyElements* are enumeration, they are accordingly mapped as OPC UA Enumeration DataTypes, as shown in Figure 4.4. This solution has the great advantage to organize all the keys composing the path to an entity as an ordered array in the attribute Value of the OPC UA Property defined, so that the original order is respected in the mapping. The root of the path is

identified by the element at index 0 and the element at last index identifies the entity pointed by the AAS Reference.

The solution described so far respects the structure of the AAS meta-model about referencing mechanism but presents some limitations from the point of view of an OPC UA Client. The information of Reference is mapped correctly inside an OPC UA Property but an OPC UA Client reading its content cannot take advantage of such information because there is no mechanism that translate the path read from the Property value to the Node of the AddressSpace referred by the path. This limitation can be overcome using OPC UA References to connect the Object mapping an AAS Reference with the Node mapping the referenced entity. The type of such UA Reference may be an ad-hoc defined NonHierarchical ReferenceType representing the semantics associated to the AAS Reference, in fact AAS Reference points to entities to define some relationship, like the attribute *asset* of the entity *AssetAdministrationShell* which points to the relevant asset entity. In this last case, the “HasAsset” NonHierarchical ReferenceType can be defined to represent this relationship. It is worth noting that with this simple enhancement the number of browsing requests to know which object is pointed by an AASReferenceType instance is drastically reduced. In [15] similar consideration are done to map AAS References in OPC UA, in fact AAS Reference is mapped as an ObjectType with a property named “keys[]”. Furthermore, a NonHierarchical ReferenceType named “AASReference” has been defined and is used to connect objects mapping AAS References to the targeted object in the AddressSpace.

There are two substantial differences between the solution proposed in [15] and the one here described: 1) there is no specific mapping for keys, therefore the complete path of the AAS Reference is converted in a string formatted following a specific serialization rule mandated in [15]; 2) OPC UA References of the same type (named “AASReference”) are used to connect Objects mapping AAS References to the targeted Object in the AddressSpace, regardless the semantics exposed by the AAS references in the AAS.

The disadvantage of 1) is that clients must parse the string path to retrieve all the information it contains, whilst mapping key values by means of

DataTypes, as we suggested, allows information to be understood at OPC UA level. The disadvantage of 2), instead, is that an OPC UA Client browsing the AddressSpace cannot distinguish the type of the Object pointed by the OPC UA Reference on the basis of the reference itself, but it must reach the Object through the reference to know its type. The solution here provided requires the definition of different ReferenceTypes according to the object to be pointed, so that an OPC UA Client browsing the AddressSpace can immediately understand the type of object pointed by the reference, only by the analysis of the type of the reference itself.

4.2.4 Mapping AAS common classes

In subsection 4.2.1, general mapping solutions have been covered for main categories of elements composing the AAS metamodel. In particular, fine-grained mapping solutions about attributes has been provided. A step further will be done about mapping common classes in OPC UA considering a new OPC UA feature, i.e. Interfaces, in order to create a direct representation of such common classes in the AddressSpace.

Following the considerations done in subsection 4.2.1, all the general rules described can be extended to all the metamodel entities inheriting from common classes. Let us consider the entity Submodel as an example to highlight the possibilities for the mapping of common classes in OPC UA. Figure 4.5a shows the mapping of SubmodelEntity defining an ObjectType named SubmodelType, where all the attributes inherited from the common classes of Submodel are mapped as OPC UA Properties and for each of them a ModellingRule is selected according the cardinality specified by each common class for every attribute. This solution is quite simple and allows to maintain the same structure of the AAS metamodel, but does not scale because the attribute mapped are not reusable and are valid just for the instances of the type SubmodelType. For instance, all other ObjectTypes mapping Identifiable entities shall define again the Properties “identification”, “idShort” and “description” as InstanceDeclarations.

Figure 4.5b shows another solution that leverages on the new OPC UA

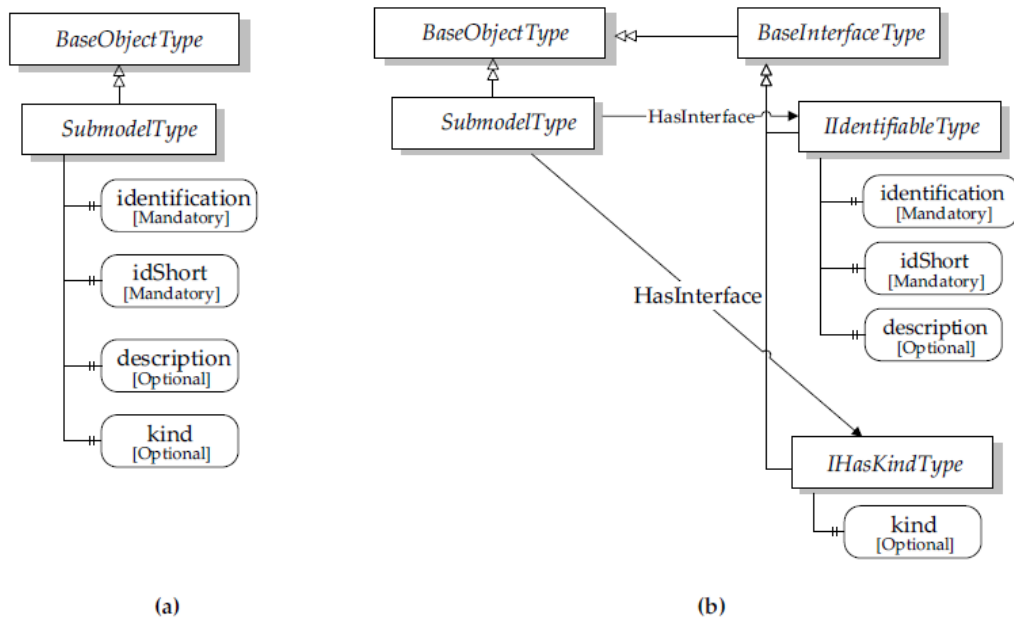


Figure 4.5: Mapping common Classes using (a) classic OPC UA and (b) new OPC UA Interfaces mechanism.

Interfaces mechanism. Both Identifiable and HasKind common classes are mapped as interfaces called *IIdentifiableType* and *IHasKindType*, respectively. All InstanceDeclaration defined under the Interface Objects are inherited by the *SubmodelType* ObjectType. Compared to the solution shown by Figure 4.5a, attributes of common classes are mapped only once and different ObjectTypes mapping metamodel entities can point to the relevant interfaces by means of *HasInterface* References. This mapping solution scale better and provide reusable interfaces in the mapping of the other AAS entities. In [15], the OPC UA Interface-based mapping solution is used just for the common classes Referable and Identifiable. For all other common classes different solutions are applied case by case.

It is worth noting that all common classes can be mapped applying one of the two proposed solutions but for *HasDataSpecification* a different approach should be used, as will be discussed in the following subsection.

4.2.5 Mapping HasDataSpecification and DataSpecification

In the AAS metamodel, instances of entities inheriting from the common class `HasDataSpecification` feature more attributes than the ones defined by its original class. The additional attributes are defined in a DST that the `HasDataSpecification` entity must point to by means of an AAS Reference. A behaviour like this cannot be realised following the standard inheritance mechanism provided by OPC UA because this would require that instances of a same `ObjectType` could feature different components and/or Properties depending on which DST they are referring to. Moreover, a `HasDataSpecification` entity can point to more than one DST, and this further complicates the scenario.

Beside Interface, in [31] a new interesting feature named `AddIn` is specified. An `AddIn` is an Object that associate features (represented by its `ObjectType`) to the Node it is applied to. OPC UA `AddIn` model differs from Interface model in that it is based on composition and not on inheritance. An `AddIn` is applied to a Node by adding a Reference pointing to the `AddIn` Instance; a `HasAddIn` Reference or a subtype shall be used. There are no restrictions for `AddIn` `ObjectTypes` and there is no special super type for `AddIns`. This feature fits for the mapping for `HasDataSpecification` entities.

According to the `AddIn` mechanism, the mapping of `HasDataSpecification` entities can be done defining an `ObjectType` named `DataSpecificationType` which is an `AddIn` mapping the entity `DataSpecification`. `DataSpecification` is an abstract entity all the DST entities must inherit from. In order to map a concrete DST entity, an `ObjectType` inheriting from `DataSpecificationType` must be created. Recalling that DST entities are Identifiable, all the `ObjectTypes` created must expose all the related attributes mapped properly. In the following, it will be assumed to use the solution depicted in Figure 4.5a.

As depicted in Figure 4.6, an `ObjectType` named `DataSpecificationIEC61360Type` is created to map the DST `DataSpecificationIEC61360` as an OPC UA `AddIn` `ObjectType` defining Properties and components as `InstanceDeclarations` mapping all the additional attributes defined by the DST.

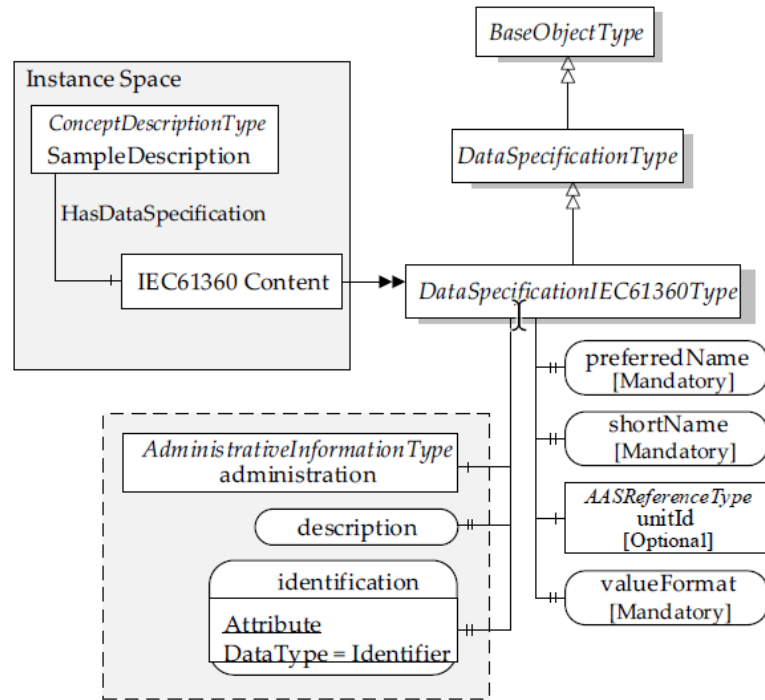


Figure 4.6: Mapping data specification templates (DST) entities as AddIn ObjectTypes and using them to map HasDataSpecification entities.

One important aspect about this mapping solution is that all the attributes of DataSpecificationIEC61360 inherited by Identifiable are mapped as Properties and components containing all the information useful for the identification of the DST in the OPC UA AddressSpace, and thus these are not InstanceDeclarations but actual Properties and Components of the Object-Type DataSpecificationIEC61360Type.

To describe the solution specified for the mapping of instances of HasDataSpecification entities, the case of an instance of a ConceptDescription describing an AAS property using the DataSpecificationIEC61360 will be used. In Figure 4.6, the ConceptDescriptionType ObjectType is assumed to map the ConceptDescription entity and the Object “SampleDescription” is an instance of it. An AddIn Object of the DataSpecificationIEC61360Type ObjectType is created (i.e., “IEC61360 Content”), connecting SampleDescription by means of a HasDataSpecification reference. We preferred defining the Has-

DataSpecification ReferenceType as subtype of HasAddIn so that a generic OPC UA client may easily detect the References involved in the mapping of HasDataSpecification entity. In [15], the concept of DST has been mapped using the OPC UA AddIn mechanism too.

4.2.6 Mapping ConceptDictionary and ConceptDescription

ConceptDictionary and ConceptDescription entities can be mapped in an OPC UA Information Model defining proper ObjectTypes. Very recently, OPC Foundation released an amendment [53] of the OPC UA Specification defining new ObjectTypes and ReferenceTypes to define classification and additional semantics of a device in terms of an external data dictionary. Such new types can be used to attach semantics to nodes in the AddressSpace referring entries in an external dictionary like IEC CDD or eCI@ss. In particular, two main ObjectTypes defined in the amendment [35] are DictionaryFolderType that represents a dictionary, and DictionaryEntryType that represents a pointer to an entry in an external dictionary.

Although mapping ConceptDictionary and ConceptDescription by using these new ObjectTypes seems feasible and coherent with the strategies proposed so far, it must be considered that both ConceptDicitonary and ConceptDescription inherit from some common classes and thus feature attributes that seem not directly representable by properties defined in the two ObjectTypes specified in [53]. Furthermore, ConceptDescription is an HasDataSpecification entity and its instances may feature different attributes depending on the referred DST. Therefore, a suitable mapping must be provided to reflect such mechanism, like the one described in the previous subsection. In [15], it is used the DictionaryEntryType ObjectType to map the ConceptDescription entity. In particular, it defines specific subtypes of DictionaryEntryType that have at least one AddIn Object to allow the usage of the IEC 61360 DST, as described in the previous subsection.

4.3 Case study: Operator Support System for assembly line

The case study is based on an Industry 4.0 assembly system and Operator Support System (OSS) [54]. Several models of a certain product are assembled by human operators in the same flow line where an OSS, provided by the assembly system, gives to the operator all information needed to perform the assembly cycle correctly and this information are provided as functions of the model to assemble.

In this case study, the assembly of products composed of several different components is assumed, among which there is always a motor controller. Therefore, different models to be assembled feature a motor controller as a component, but each model requires a motor controller with different specification; in particular different product models require motor controllers with different maximum rotation speed. For example, assembly of Model X requires a motor controller with a maximum rotation speed greater or equal to 2000, whilst the Model Y must be assembled including a motor controller with a maximum rotation speed greater or equal to 3000. For each product arrived to the human operator in the flow line, the OSS must suggest him the right motor controller to be assembled according to the model of the product received; the OSS must specify an unambiguous id of the product part to be assembled in order to avoid assembling errors by the human operator.

Every motor controller used as component in the assembly line provides an AAS containing all the information useful for the assembly, i.e. max rotation speed. A simplified version of the AAS is depicted in Figure 4.7 as an UML instance diagram using the same notation specified in [15]. The use case here described involves the representation of the AAS inside an OPC UA Server, so that an OSS based on OPC UA communication can take advantage of the mapping solution described so far to simplify the assembly process. The AAS (named SampleAAS) contains a Submodel (named 123456789) and an asset (3S7PLFDRS35). The Submodel features only a property (NMax). Furthermore, the AAS has a ConceptDictionary (SampleDict) containing a ConceptDescription (NMaxDef).

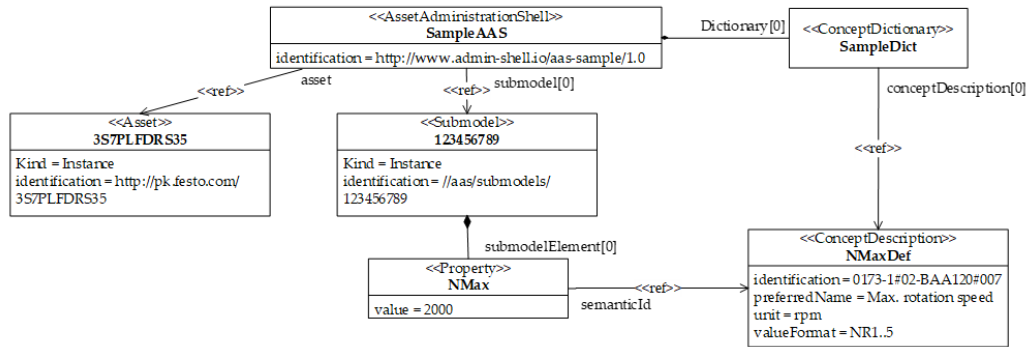


Figure 4.7: UML instance diagram showing the AAS of the case study.

All the Identifiable entities feature the attribute identification containing a globally unique identifier. Since NMax is a HasSemantics entity, its attribute semanticId contains the identifier of the ConceptDescription defining its semantics, i.e., NMaxDef. In particular, the semantics specifies that the Property value (2000) represents the maximum rotation speed supported by the motor controller, and it is expressed in rpm (revolutions per minute).

The AddressSpace that will contain the AASs of motor controller may be organized creating a Folder for each kind of identifiable entity, as discussed in Section 4.2.2. Therefore, the Folders “Asset Administration Shells” and “Assets” (shown in Figure 4.8) will organize objects mapping the AAS and the asset, respectively; in a similar manner, the folders “Submodels” and “ConceptDescriptions” will organize objects mapping the Submodel and the ConceptDescription, respectively. It is worth noting that these last two Folders are depicted in Figure 4.8, but for space reason, their contents are shown in Figure 4.9.

All the identifiable entities in the use case are mapped using instances of OPC UA ObjectTypes: AASType for SampleAAS (see Figure 4.8), AssetType for 3S7PLFDRS35 (see Figure 4.8), SubmodelType for 123456789 (see Figure 4.8) and ConceptDescriptionType for NMaxDef (see Figure 4.9). Since all these ObjectTypes represent Identifiable entities of the metamodel, they point to an OPC UA Interface “IIdentifiableType”, which is not depicted in figures for space reason. All these instances feature a property “identi-

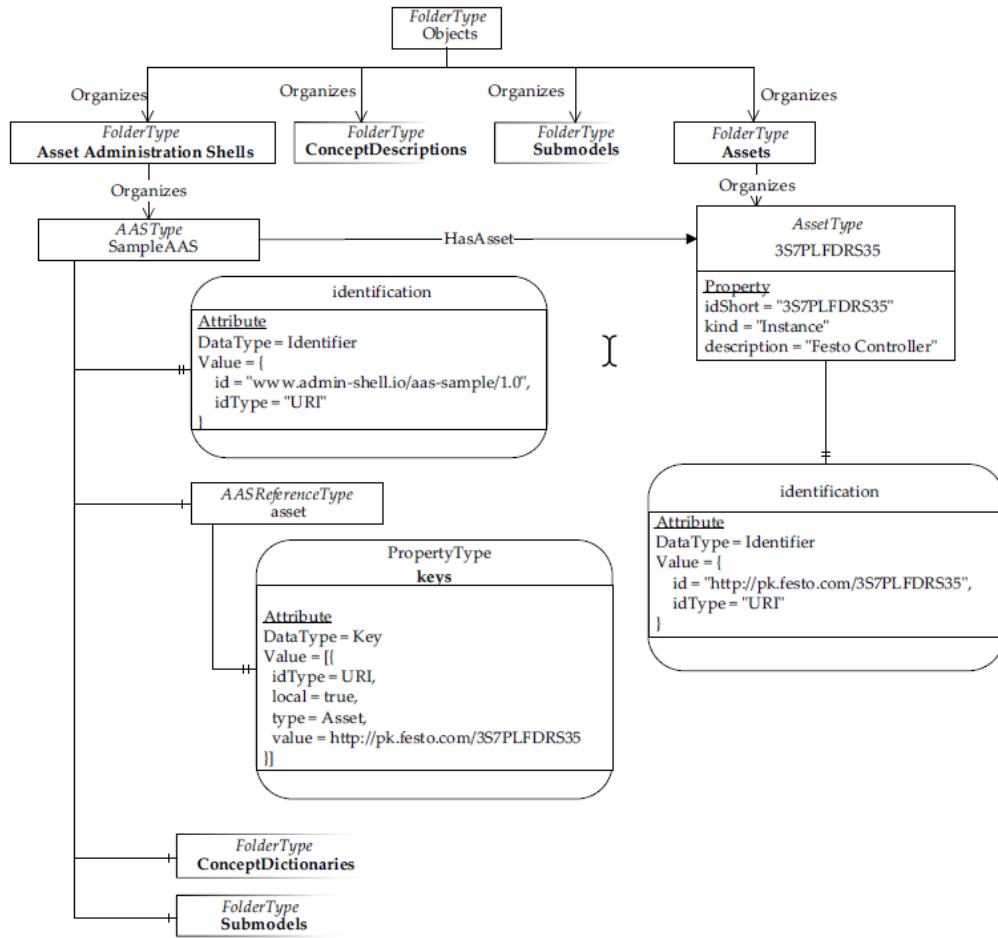


Figure 4.8: Mapping of one of the AAS considered in the case study into OPC UA information model.

fication” that contains the relevant identifier of the entity represented. All the attributes consisting in AAS References (depicted with <<ref >> in Figure 4.7) are mapped using instances of the AASReferenceType ObjectType. Furthermore, ad-hoc defined Non-Hierarchical ReferenceTypes are used to enhance the representation of AAS References in OPC UA and simplify the browsing of an OPC UA Client: HasAsset, HasSubmodel, HasSemantics, Has-ConceptDescription.

All the attributes consisting of composition are mapped as folder objects named using the plural noun of the relevant attribute. Such folders organize

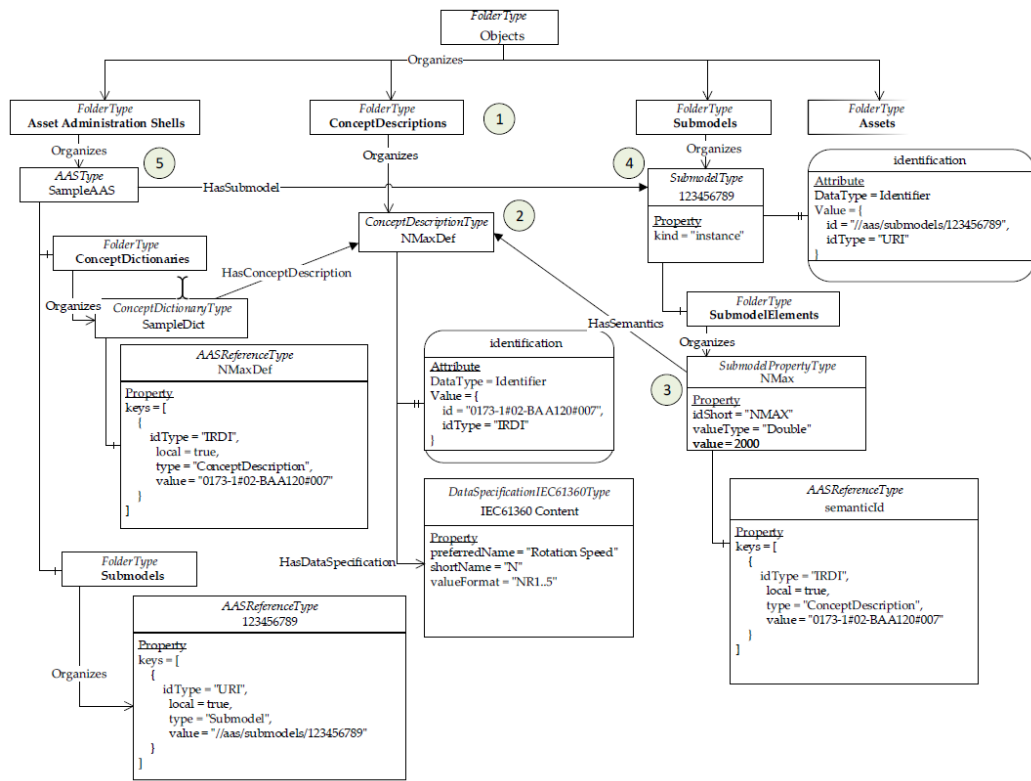


Figure 4.9: Continuation of the mapping of the AAS considered in the case study.

objects mapping the entities contained by such composition attributes. The ConceptDictionary SampleDict and the AAS Property NMax shown in Figure 4.7 are mapped using instances of ad-hoc defined ObjectTypes, i.e. ConceptDictionaryType and SubmodelPropertyType, respectively. As depicted in Figure 4.9, these instances are the Objects “SampleDict” and “NMax”. Finally, since the ConceptDescription NMaxDef features additional attributes coming from the DST for IEC 61360, an AddIn instance of the DataSpecificationIEC61360Type ObjectType (i.e., “IEC61360 Content” in Figure 4.9) is created and connected to the “NMaxDef” object by means of a HasDataSpecification Reference. Therefore, all the properties of this AddIn instances are filled accordingly to all the relevant values of the ConceptDescription.

We consider the scenario where the OSS uses an OPC UA Client to access an OPC UA Server containing all the AASs of the motor controllers, as shown

in Figure 4.10. Different instances of AASType are present, both based on "SampleAAs" shown in Figure 4.7; only two instances are depicted for space reason: Motor Controller 1 and Motor Controller 2. These instances differ for the NMAX property, hence the motor controllers represented by these AASs differ for the maximum rotation speeds supported.

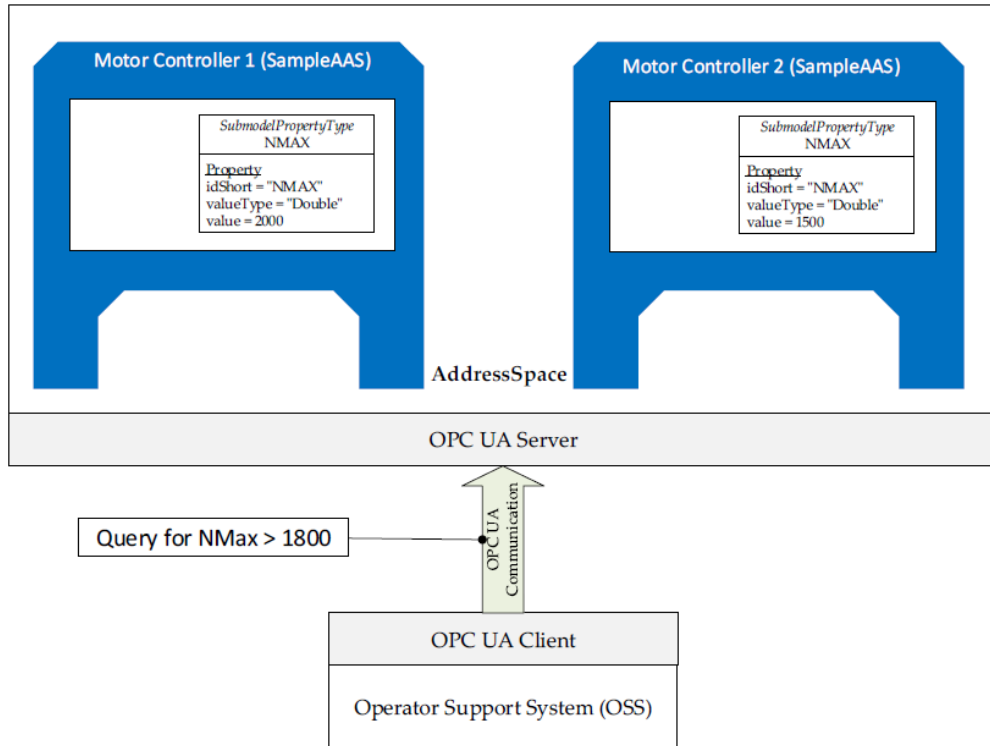


Figure 4.10: Example of OPC UA Server maintaining instances of AASType.

At a certain moment in the assembly process, the OSS suggest to an human operator a specific motor controller to be assembled on the basis of a query submitted to the OPC UA Server. Assuming that the product to be assembled requires a motor controller with maximum rotation speed supported greater than 1800, the OSS uses the OPC UA Client to browse the AddressSpace in the OPC UA Server looking for an instance of AASType featuring a Property NMAX > 1800.

Based on the AAS depicted in Figure 4.7, the OSS knows that the property

it must check (i.e. Max Rotation Speed) have a semantics identified by the id “0173-1#02-BAA120#007”. It is worth noting that the OSS does not look for a Property named “NMAX” but for a property featuring a well-known semantic identifier. This is of paramount importance, because Properties can have different names but have the same semantics, which is always identified with the same id.

Starting from folder “Concept Descriptions” (see point 1 in Figure 4.9), the object “NMaxDef” of `ConceptDescriptionType` type is selected because it features a Property “identification” containing the identification “0173-1#02-BAA120#007” (point 2 in Figure 4.9). Starting from this Object, the OPC UA `HasSemantics` Reference is followed in the opposite sense in order to look for Objects of `SubmodelPropertyType` type. The object “NMAX” is reached (Point 3 in the Figure 4.9). On the property “value” it is possible to perform the query given in input. In this case, the condition of the query is satisfied. The last step consists in returning to the OSS the id of the AAS found, which models the real motor controller featuring a rotation speed greater than 1800. Starting from object “NMAX” it is possible to reach its container, i.e., the `SubmodelType` Object “123456789” (point 4 in Figure 4.9). Finally, following the OPC UA Reference “`HasSubmodel`” in the opposite sense it is possible retrieving the id of the AAS, i.e., Motor Controller 1 (Point 5 of Figure 4.9). This information will be passed to the human operator, in order to realize the correct assembly.

4.4 Discussion

The contribution of this research activity consists in the definition of an approach in the mapping of the AAS metamodel into an OPC UA Information Model providing reasoning and rationales behind the approach adopted. The AAS metamodel is a big step forward for the implementation of I4.0 Component exposing a well-known interface and information structure. The document [15] defines the AAS metamodel as the fundamental means for the interoperable exchange of asset information between partners of the value-chain network. The representation of AAS in file format using XML or JSON is lim-

ited to the scenario of data exchange between partners and cannot be used for the realisation of an I4.0 component because there is no I4.0-compliant communication to access internal data of the AAS. For this reason, OPC UA is considered the right solution because it provides standardised communication capabilities, several standardised service sets, and a standardised information model mechanism for data representation. In [15] a preliminary work for the definition of an OPC UA mapping of the AAS metamodel is provided and, in our research, we validated the feasibility of OPC UA as the right technology to expose AASs in industrial scenario.

Our research involved the adoption of a bottom-up approach for the mapping of the AAS metamodel entities in the elements of the OPC UA Information Model. We started from the most fine-grained elements of the metamodel, i.e. entities and attributes, classifying them respect the functionalities they cover in the AAS metamodel (e.g. structuring the AAS, defining types, containing values, containing AAS References, etc.) and then specifying possible mapping solutions for each of them in comparison with the ones provided in [15]. We continued mapping the complex construct and mechanisms of the AAS metamodel, i.e. Common Classes and AAS Reference, providing several possibilities for their mapping.

In the mapping process of the AAS metamodel, we realised that direct representations of some AAS metamodel mechanisms in the OPC UA AddressSpace cannot be advantageous if they do not fit with some OPC UA analogous mechanism. The best example is the AAS referencing mechanism, which can be directly mapped inside the AddressSpace but cannot be used because OPC UA already have its own referencing mechanism. Our solution involved using OPC UA References to enhance the mapping of the AAS reference mechanism, defining new NonHierarchical reference Types. Of course, the information of the original AAS References mapped inside the AddressSpace is maintained inside their relevant Objects, but the addition of new OPC UA reference allows the utilization of the AAS Referencing mechanism in the context of OPC UA. This was demonstrated in the use case discussed in section 4.3, where the OSS uses such UA References properly defined inside the OPC UA AddressSpace to retrieve the desired information.

The mapping solutions developed in this research have been implemented by the authors as an OPC UA Information model with the project name *CoreAAS*, publicly available on GitHub². *CoreAAS* has been used for the development of an open source SDK for Node.js³, named *node-opcua-coreaas*⁴, based on *node-opcua*⁵ for the realisation of OPC UA Server using *CoreAAS* functionalities.

Iñigo et al. [55] realised an implementation of the AASs based on *CoreAAS* for an industrial scenario to facilitate the integration of grinding machines with other components or machines in the production plant.

4.5 Publications

A preliminary research carried out on OPC UA-based AAS has been presented at 45th Annual Conference of the IEEE Industrial Electronics Society (IECON), Lisbon, Portugal [56].

All the rationales and insights behind the mapping of the AAS metamodel in OPC UA has been published in the scientific journal “Computers” [57].

²<https://github.com/OPCUAUniCT/coreAAS>

³<https://nodejs.org/>

⁴<https://github.com/OPCUAUniCT/node-opcua-coreaas>

⁵<https://github.com/node-opcua/node-opcua>

Chapter 5

AAS representing PLC based on IEC 61131-3

In the previous chapters the AAS has been presented as the universal means to access all the information relevant to an asset useful during whole life cycle of a production system, from its development until its disposal. The AAS metamodel has been presented as the universal means to structure information inside an AAS and, in particular, in Chapter 4 mapping proposal for its representation inside an OPC UA Server has been discussed. This results allowed us to define an OPC UA Information Model specific for the realisation of an OPC UA-based AAS which can be used to expose asset information to clients in a standardised manner in order to facilitate common industrial scenarios.

At the lowest level of the hierarchy of a production system, automation and control programs are executed by Programmable Logic Controllers (PLC), whose programming technology is based on IEC 61131-3 standard. Usually, control programs, PLC running such programs, and the physical plants they control are strictly related. The description of such relationships should be clearly defined and accessible during the whole life cycle of a production system, so that they can be used for the definition of testing plant operations, maintenance operations at run-time and reconfiguration process of the plant.

In this chapter we discuss an approach leveraging the concept of AAS

to maintain such information, providing a model for the description of such relationships between the IEC 61131-3 programs, the PLC and the production system. The results presented in the previous chapter allowed us to realise such model using OPC UA and realise a case study where the AASs describe the relationships between a PLC, the program, and the physical parts needed to control a drilling machine.

5.1 Introduction

Nowadays, automation and control programs are executed by PLCs and, due to their massive adoption, it seems legit thinking that they will be used in the age of Industry 4.0 too, as stated in [58]. The PLC programming technology adopted nowadays is based on IEC 61131-3 [59], which is a standard defined in the past to cope with the heterogeneity of vendor-specific languages and technologies adopted for PLC programming. For this reason, it is legit imagine that I4.0 controllers will be based on this standard [60], in particular during the transaction of plants in Industry 4.0 where devices cannot be completely replaced.

There are close relationships (both logical and physical) between IEC 61131-3 programs, the hardware and software resources of the PLC where they run, and the plant. With plant are intended all the controlled machines, control devices, control applications and communication systems. An example of such relationships can be a variable of an IEC 61131-3 program mapped to a real input or output of the PLC which, in turn, is connected to a real device (e.g. sensors, actuators). Another example is a variable shared between an IEC 61131-3 program running in a PLC and a software tool running in another device and exchanging information with the PLC.

Considering the life cycle of a production system, a comprehensive description of all these relationships could help the definition of testing operations before the utilization of the plant (after its realization) and maintenance operations at run-time. Applying variations to the production system or changes to the model of manufactured products often requires adjustments in the plant configuration [61]; the reconfiguration process can be easily conducted

if a complete and standardised description of the entire system is available, involving both the PLC programs and the relations with the actual plant.

Current literature presents many research papers dealing with integrated models including IEC 61131-3 programs and the relevant plant controlled. For the implementation of manufacturing line, proper verification of a line's operational status are usually performed using approaches involving simulation techniques; since PLC programs only contain the control logic without specifying device models, such approaches require a corresponding plant model to perform simulation [62, 63]. In [64] a detailed overview on the development of a PLC simulation environment is provided, pointing out the importance of realising a corresponding virtual plant model (the counterpart system) required to interact with the inputs and outputs of the PLC. Other approaches present in the literature are based on the verification of properties of the state machine on which the PLC program is based; again these approaches are based on the use of a plant model integrated with the PLC program to be verified [65]. Park et al. [66] describe another approach based on the visual verification of PLC programs that integrates the program with a corresponding plant model, so that users can intuitively verify the PLC program in a 3D graphic environment.

Considering Cyber-Physical Systems (CPS), current literature presents several approaches pointing out the need to model the entire set of CPS functionalities, including control programs and the on-board hardware (e.g. sensors) [67].

An AAS can be used to represent the relationships between PLCs and the plant, but what is missing to realise such description is a model representing PLC programs based on IEC 61131-3 and the relevant relationships with the PLC hardware and the devices of the controlled plant. Our research involved the study of the IEC 61131-3 software model in order to represent programs based on it inside an AAS using the AAS metamodel. Therefore, in this research activity we defined a specific AAS Submodel to model whatever IEC 61131-3 program inside an AAS respecting, of course, the rules and specifications of the standard IEC 61131-3. Starting from this Submodel, the relationships between the software elements with other parts of the plant

or the PLC itself can be described inside the AAS.

5.2 Overview on IEC 61131-3

The standard IEC 61131 provides an architectural definition and a software model for industrial PLCs. In particular, IEC 61131-3 cope with the problem of the existence of different vendor-specific languages for PLCs programming [68]. IEC 61131-3 specifies syntax and semantics of a unified suite of programming languages for PLCs consisting in Instruction List (IL), Structured Text (ST) Ladder Diagram (LD), Function Block Diagram (FBD) and Sequential Function Chart (SFC).

The foundation of IEC 61131-3 is a unified software model for the PLC, shown in Figure 5.1, which provides the basic high-level language elements that are programmed using the aforementioned programming languages.

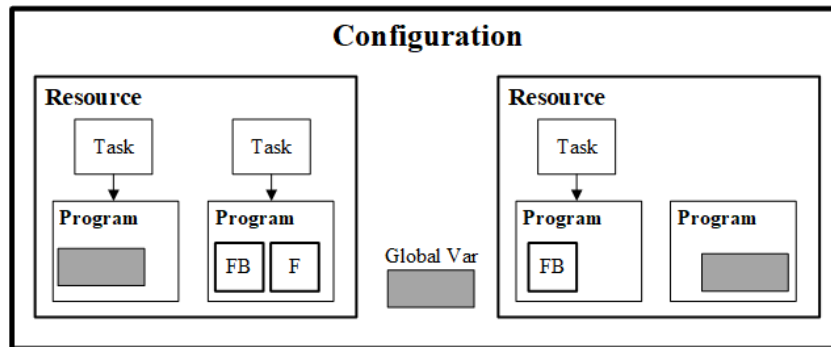


Figure 5.1: Software model of IEC 61131-3.

The main elements composing the software model of IEC 61131-3 are Configuration, Resource, Task and Program Organisation Unit (POU).

Configuration This element defines the entire software project and must include at least one Resource. A Configuration may refer to one or more PLCs involved in the project.

Resource It represents a processing facility of the PLC that can execute a program and is defined inside a Configuration. It allows the definition of several information about the PLC, like hardware features (e.g. type

of processor, memory dimension, number of physical inputs and outputs) and software features (e.g. operating system version, firmware identification).

Task This software element is used to define the desired execution mode of a program. Among the available scheduling facilities there is the cyclic execution, according to which a program can be periodically executed; in this case, an interval is assigned to a Task, specifying the period the program is executed.

POU Is a functional element which is used to decompose an IEC 61131-3 control program. A POU may be a Program, a Function Block (FB) or a Function (F).

All POUs can be defined using one of the programming languages specified in the standard. A Program typically consists of interconnected Function Blocks exchanging data and can communicate with other Programs. The execution of different parts of a Program may be controlled using Tasks. A Function Block is used to wrap an algorithm and make it reusable inside different parts of a Program. Using FBs, it is possible to create reusable parts of code for a better modularization of the program. It consists of variables for inputs, outputs, and internal storage, and it can use other FBs internally. Function is a reusable software element that generates the same output values when the same input values are provided. It differs from FB in that it has no internal state, whereas FB retains its internal values from the last execution.

Variables in an IEC 61131-3 program can be declared inside any of the aforementioned elements of the software model. Depending on where and how they are defined, variables can be global, local, input, output, external. Variables may be declared using several attributes, like AT which allows the association of a variable to a particular memory address. According to the standard IEC 61131-3, the internal memory of PLC is composed by the Input (I) and Output (Q) process images and Merker (M) memories. The I and Q process images are updated at each Program Scan [59]; inputs are sampled and copied in memory I, whilst data stored in memory Q updates the actual values of outputs at the end of each Program Scan.

The relationships between the software elements composing an IEC 61131-3 can be highlighted using an UML class diagram depicted in Figure 5.2. This diagram has been defined as a starting point, guiding us for the definition of an AAS Submodel;

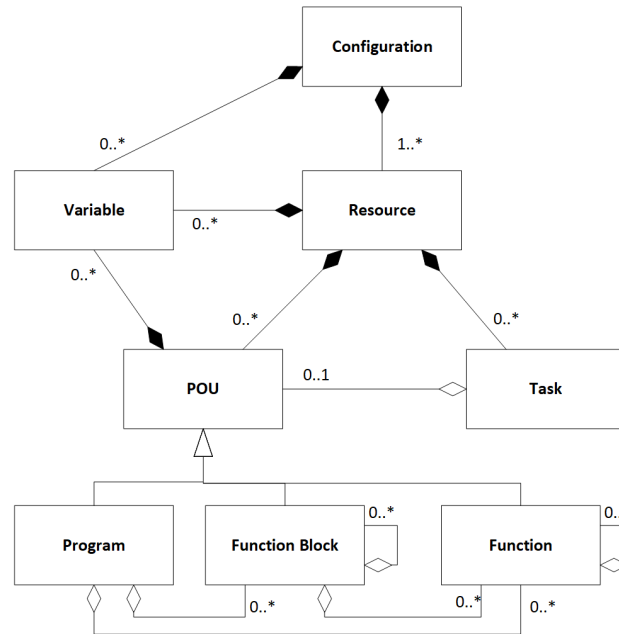


Figure 5.2: UML class diagram showing the relationships between entities of the IEC 61131-3 software model.

5.3 Submodel for IEC 61131-3

The aim of this research activity is representing each element defined in an IEC 61131-3 Configuration using the AAS metamodel. For this reason, an AAS Submodel, referred in the following as “IEC 61131-3 Submodel”, has been proposed to represent an IEC 61131-3 Configuration. Our idea is that every AAS representing a PLC contains a “IEC 61131-3 Submodel” describing the control program it runs. Of course other Submodels exist inside the PLC AAS describing all different aspects relevant the asset, as usual.

According to Figure 5.2, IEC 61131-3 Configuration is made up by several elements (e.g. Resources, POU, Tasks, etc.), thus AAS SubmodelElements

inside the IEC 61131-3 Submodel can be used to represent the IEC 61131-3 elements.

The instances of the class `SubmodelElementCollection` (SEC) can be used to organise a Submodel inside the AAS. SEC can be used in two different ways: 1) using SEC as a Folder or 2) using SEC to represent a complex element. With method 1), A SEC may be defined to contain several `SubmodelElements`, each of which represent an IEC 61131-3 element. In this case the SEC does not represent an IEC 61131-3 element, but it has only organization purpose, like a folder in a file system. This kind of organization is proposed to group `SubmodelElements` representing IEC 61131-3 elements of the same category (e.g. Variables, Tasks, POU's). The relevant advantage is an easier classification of `SubmodelElements`.

In some cases, an IEC 61131-3 element features so many characteristics that cannot be represented using the standard attributes provided by the subclasses of `SubmodelElement`. For instance, an IEC 61131-3 Variable features a lot of attributes, like AT, scope, and type which cannot be represented all together using a single `SubmodelElement` like a Property. In other words, a Variable is a complex element and a SEC can be used to represent it using the method 2). This method involves the use of SEC as a direct representation of the complex element to group its attributes that, in turn, will be mapped using proper `SubmodelElement` subclasses.

In summary, with method 1) the SEC have no meaning related to the IEC 61131-3 program, but it is just a means to organize the Submodel structure; with method 2), instead, the SEC represents a real entity of the IEC 61131-3 program, like a Variable or a Task. To differentiate SEC at a glance, it has been assumed to use the values of the attribute category (inherited by the common class `Referable`), using a value "SET" to classify a SEC representing just a container of other `SubmodelElements` (method 1) and using a value "ELEMENT" to classify a SEC representing a complex element.

In the following, will be described how each element of the IEC 61131-3 software model depicted in Figure 5.2 can be represented inside the AAS using the AAS metamodel.

5.3.1 Configuration

A Submodel named “IEC 61131-3” represents a Configuration in the context of an AAS, as depicted in Figure 5.3.

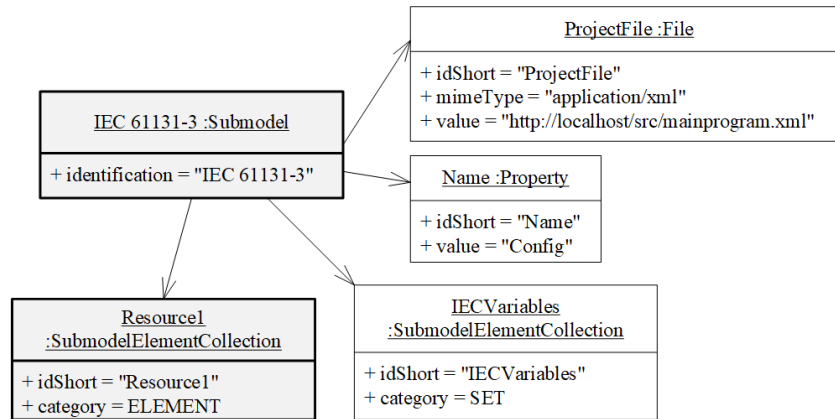


Figure 5.3: Representation of an IEC 61131-3 Configuration.

Attributes describing the Configuration can be represented as Properties and aggregated under this Submodel to describe the configuration itself. For instance a Property “Name” could provide a mnemonic name of the Configuration, whilst a File named “ProjectFile” could contain the path to the location of the project file (e.g. the path shown in the value attribute).

Since configuration contain Variables and Resources, their representations will be organized under proper SECs. A “folder” SEC named “IECVariables” is defined under the IEC 61131-3 Submodel to contain information related to Variables. How Variables are represented is described in the remainder of this section. Resources, instead, are represented using “complex object” SECs, thus featuring the value “ELEMENT” in the *category* attribute. The reason behind the difference between Resource and Variables is that Resources are numerically less than Variables, hence they require less organization in the hierarchy. For this reason, we did not create a Folder to contain them, as done for Variables, removing an unneeded level in the hierarchy of elements composing the Submodel.

5.3.2 Resource

Resource can be represented with a SEC with category set to “ELEMENT”, as depicted in Figure 5.4. It features Properties specific for the Resource, e.g. name, CPU model, Operating System version, firmware identification. Since Resource contains Variables, Tasks and POU’s, three SECs with the attribute *category* set to “SET” are created to organize specific representation of such IEC elements. For Variables, Tasks, and POU’s, the SECs “IECVariables”, “IECTasks”, and “IECPOUs” are created, respectively.

The SEC named “IECVariables” collect all the information related to the Variables defined under the Resource. The SEC “IECTasks” collect all the information related to the Tasks defined in the relevant Resource. Finally, the SEC “IECPOUs” collect all the information related to Programs, Function Blocks and Functions that are used inside the relevant Resource.

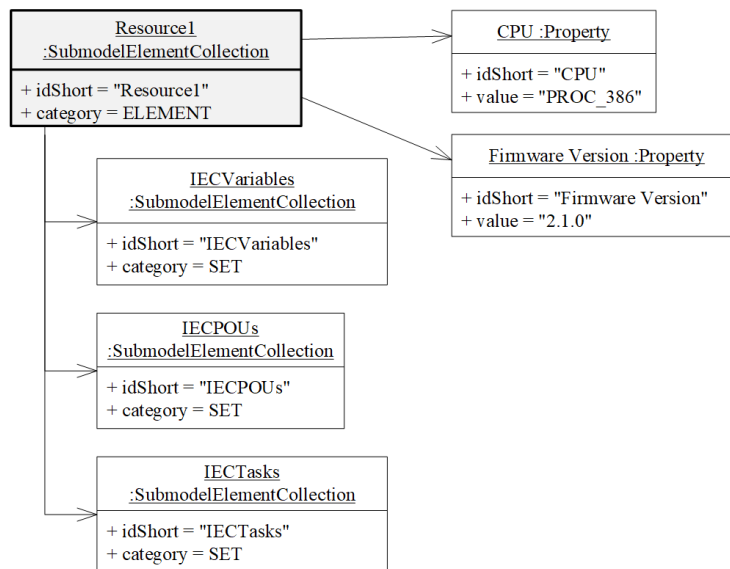


Figure 5.4: Representation of an IEC 61131-3 Resource.

5.3.3 Program

As done for Resource, Program is represented as a SEC of category “ELEMENT” inside the AAS Submodel, as shown in Figure 5.5.

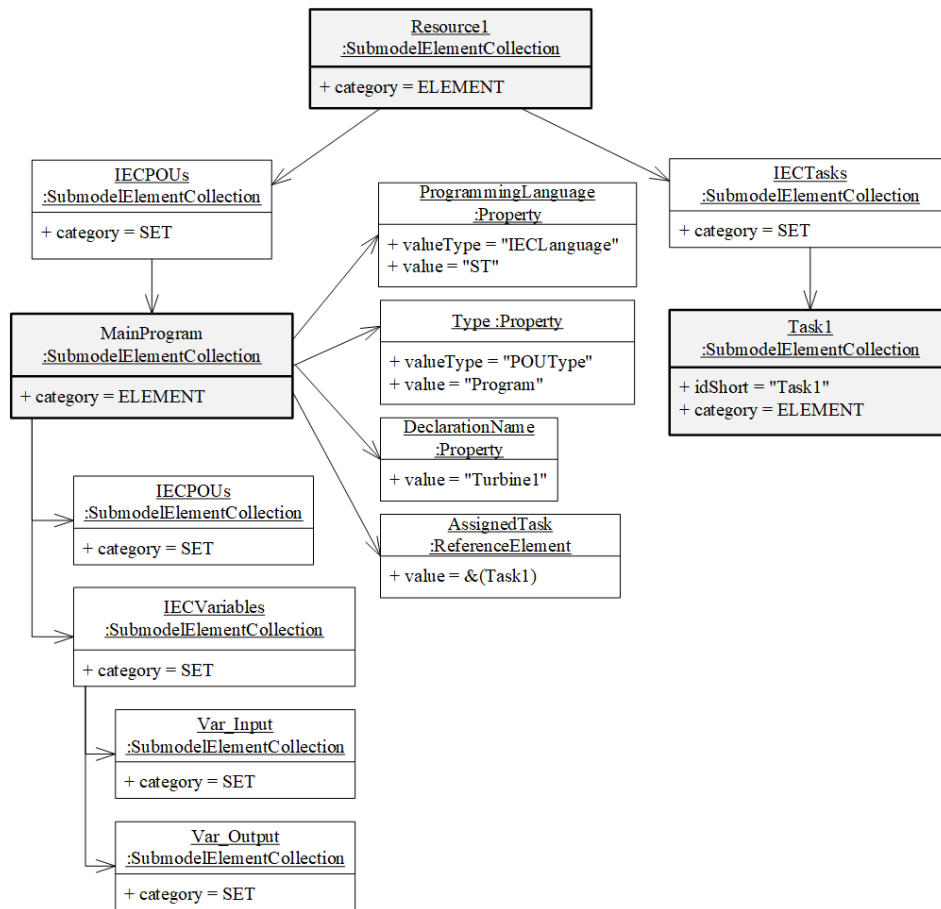


Figure 5.5: Representation of an IEC 61131-3 Program and its relationship with an IEC Task.

Such element contains Properties related to the Program description like a mnemonic name (“DeclarationName”) and the type of the POU (which is set to “Program” in this case). Another Property is the “ProgrammingLanguage” used to specify the IEC 61131-3 programming language adopted for the program; we defined the enumeration type “IECLanguage” inside the AAS Submodel, which provides all the names of the IEC 61131-3 languages as allowed values. For instance, in Figure 5.5, the value “ST” for this Property shows that the language used for the Program is Structured Text.

An instance of ReferenceElement named “AssignedTask” is defined, instead, to represent which Task the Program is associated with. It is worth

noting that the Task pointed by “AssignedTask” is one of the elements collected under the SEC “IECTasks” of the relevant Resource contained in the AAS Submodel. As done for Configuration and Resource, a SEC “IECVariables” is defined to collect information related to Variables defined inside the relevant Program.

According to the IEC 61131-3 standard, Programs may have input and output parameters, called VAR_INPUT and VAR_OUTPUT, respectively [59, 68]. VAR_INPUT represents the set of information received by a Program, whilst VAR_OUTPUT are the parameters whose value is given back by the program. For this reason, two folder-like SECs named “Var_Input” and “Var_Output” may optionally be defined inside “IECVariable” to collect eventually all the entities representing VAR_INPUT and VAR_OUTPUT Variables of Program.

Since a Program may contains POUs like instances of Function Blocks or Function calls, a SEC “IECPOUs” is used to collect all the information about such POUs, i.e. instances of Function Blocks and Functions called inside the Program.

5.3.4 Function Block

Since Function Blocks are like Programs, they can be represented like SECs of category “ELEMENT”, like shown in Figure 5.6. Such element contains Properties related to the Function Block instance like the declaration name and the type of the POU (which is set to “Function Block” in this case).

As done for Programs, an ReferenceElement “AssignedTask” is defined to show which Task the Function Block is eventually associated with.

Furthermore, the same consideration done for Programs about Variables are valid for FB too. Therefore, “IECVariables” is used to collect all the information related to the Variables, and the SECs “Var_Input” and “Var_Output” can optionally be used to collect input and output Variables of the Function Block, respectively.

A SEC of category “SET” named “IECPOUs” is used to collect all the information relevant the POUs adopted by the Function Block, i.e. instances

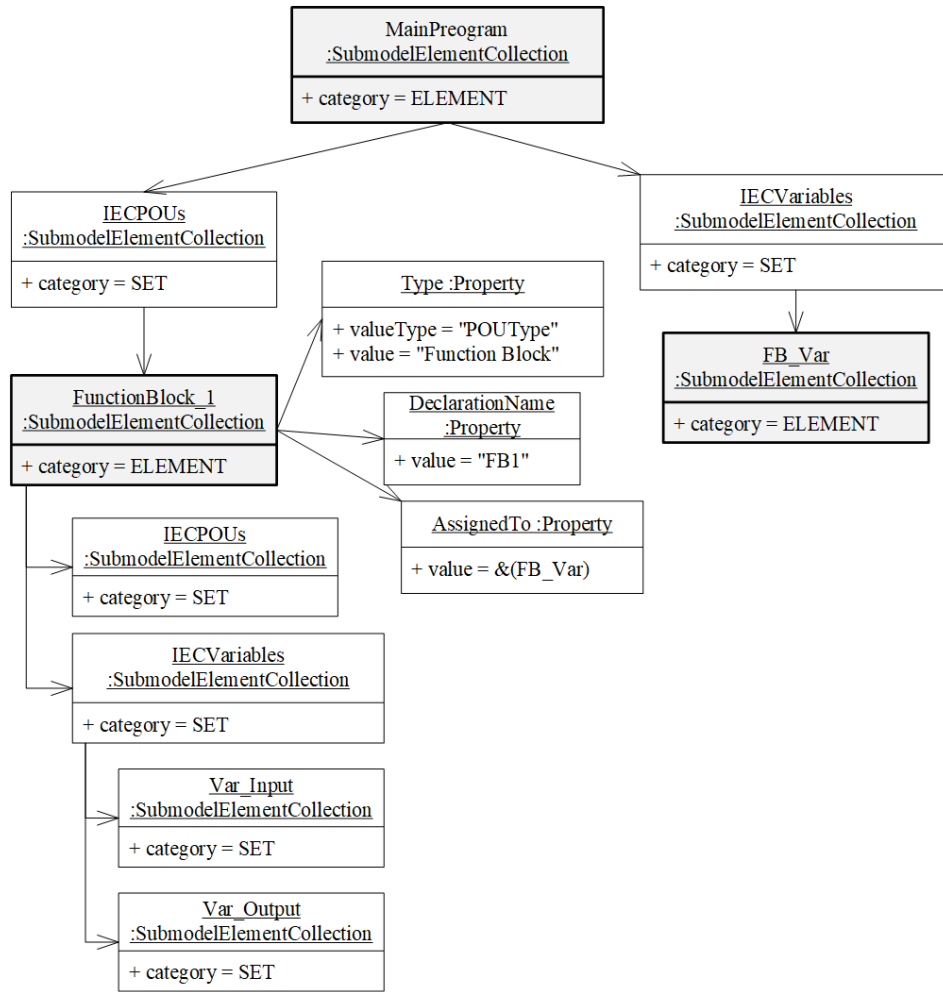


Figure 5.6: Representation of an IEC 61131-3 Function Block and its relationship with an IEC Variable.

of other Function Blocks and Functions called by the Function Block. This considerations reflect all the relationships between elements highlighted in Figure 5.2.

In IEC 61131-3, FB may be assigned to Variables, thus such behavior must be represented inside the AAS Submodel. For this reason, a ReferenceElement named “AssignedTo” points to the SEC representing the Variable containing the Function Block instance.

5.3.5 Function

A Function is represented as a SEC with category “ELEMENT”, as shown in Figure 5.7. It may contain Properties related to the Function description like the name and the POU type. A SEC “IECVariables” is used to collect

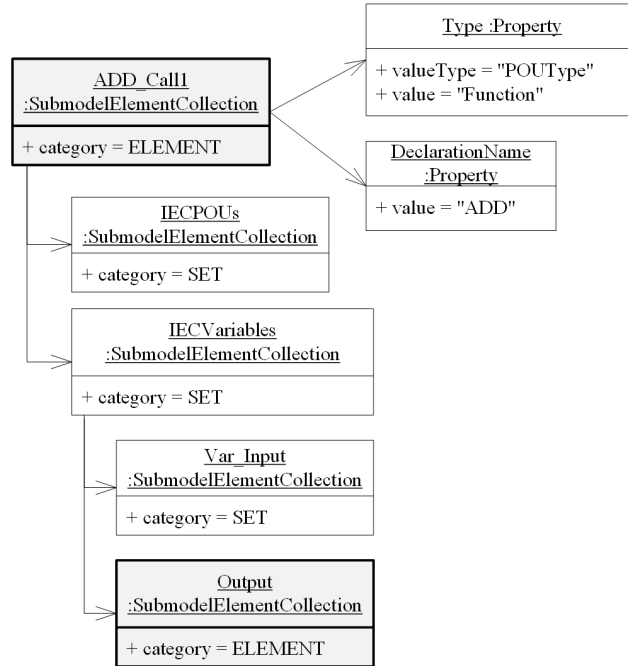


Figure 5.7: Representation of an IEC 61131-3 Function.

all the information related to the Variables of the Function. Functions may have one or more input parameters (VAR_INPUT Variables) but, differently from FBs, they do not have output parameters but return exactly one element as function “return” value. Therefore, the SECs “Var.Input” can optionally be used to collect input Variables representing the input parameters, whilst a well-known variable named “Output” is defined among the Variables to capture the value returned by the Function.

Finally, as done for the other POUs, a SEC “IECPOUs” is used to collect, if needed, all the information relevant to the Function calls present inside the Function.

5.3.6 Task

A Task is represented as a SEC of category “ELEMENT”, as depicted in Figure 5.8, collecting all Properties containing information related to the task, e.g. Interval and Priority.

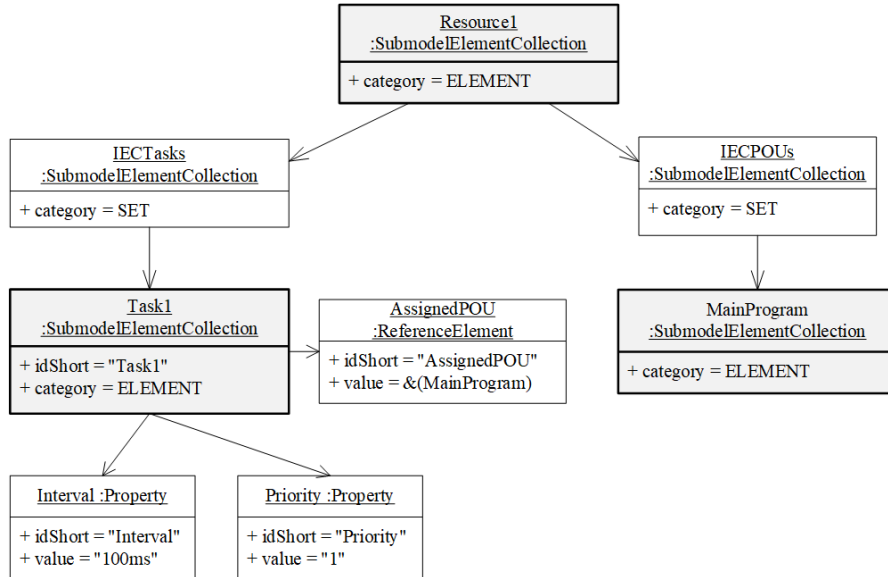


Figure 5.8: Representation of an IEC 61131-3 Task and its relationship with POU.

In order to represent which POU is currently running under the Task, the ReferenceElement “AssignedPOU” is used to point the SEC representing such POU, which is contained in the SEC “IECPOUs” of the relevant Resource.

5.3.7 Variable

A single Variable is represented as a SEC of category “ELEMENT” as shown in Figure 5.9. It contains Properties related to the description of the Variable itself, e.g. Name, Retentive, Scope, DataType Value and Address.

Inside POU, Variables can be assigned to other Variables or passed as input or output parameter/variable of other POU. Furthermore, results from Function or expressions can be assigned to Variables. To represent these relationships, two ReferenceElements named “AssignedFrom” and “AssignedTo”

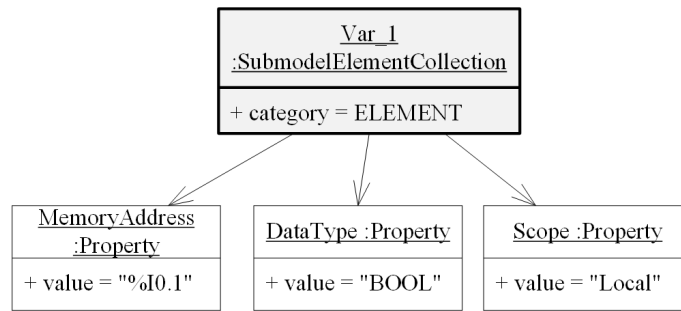


Figure 5.9: Representation of an IEC 61131-3 Variable.

are proposed to point to the element giving the value or receiving the value, respectively; the name used for this ReferenceElement depends on the direction considered for the assignment. Instead, when the content of a Variable is obtained as a result of an expression containing other Variables, one or more ReferenceElements named “DependsOn” are used to point the representations of the relevant Variables used in the expression. For instance, if the value of a Variable Z is obtained applying the formula $Z = X + 3 * Y$, where X and Y are in turn Variables, we say that Z depends on the value of both X and Y . Therefore, the SEC representing Z will expose two ReferenceElements “DependsOn” pointing to the SECs representing X and Y , respectively.

5.3.8 Semantics for IEC 61131-3 elements

When the IEC 61131-3 Submodel is explored, it must be clear what each element represents in respect to the IEC 61131-3 standard. Clients accessing the AAS Submodel “IEC 61131-3” must be able to distinguish whether an element, like a SEC, represents a Variable or a Resource. Furthermore, it must be clear what a particular Property represents for an IEC 61131-3 program. Definition of semantics played a very important role in the definition of this Submodel because a mandatory requirement for the interoperability is that the meaning of each element must be clearly understood and shared by all the partners of the value chain.

Since each element inside the IEC 61131-3 Submodel inherit from the common class **HasSemantics**, the attribute *semanticId* of each element points to

a semantic description contained in a Semantics Repository properly defined. The repository could be considered like a dictionary of IEC 61131-3 terms and concepts used by the standard. In this way, the attribute `semanticId` allows to point to the IEC 61131-3 concept defining uniquely the role of that element inside the IEC 61131-3 standard. To achieve interoperability, the Semantics Repository must be shared between all the value-chain partners using the AAS IEC 61131-3 Submodel.

5.4 Using AASs to represent PLC and Real Plant

As said so far, several relationships logically and physically exist between IEC 61131-3 programs running on PLC, the PLC physical parts, and the various parts (e.g., devices, sensors, actuators, etc.) composing the plant. Our research aim to describe all these information and make them accessible by means of the AASs that represent all the involved parts in the digital world.

The IEC 61131-3 Submodel described in the previous section is just one part of the representation and on its own cannot encompass the description of all the aforementioned relationships. As will be described in the reminder of this section, new Submodels must be defined on the relevant AASs describing both the parts of the Plant and the PLC.

Usually, a PLC features electrical connections, also referred as terminals, used to connect devices depending to the characteristics of those connections (e.g., input, output, 0-24V digital, 0..20mA analog, 4-20mA analog). Such terminals are organized into I/O modules mounted in racks. In Figure 5.10 a so-called compact PLC is shown; it features input and output connections and does not uses modular racks.

During the configuration of a PLC, some of the electrical connections are mapped into internal memory locations of the I and Q memory [68]. In turn, Variables defined in a IEC 61131-3 program may be associated to these memory locations using the standard attribute `AT`. Finally, the I/O electrical



Figure 5.10: PLC and I/O electrical connections.

connections of the PLC are connected to the terminals of the devices controlled by the PLC. For instance, a pump connected to a PLC is switched depending on the value of a Variable defined inside the control Program of the PLC. Therefore, there must be a boolean variable (e.g., "PumpTest") associated to a specific memory location (e.g., %Q0.0) which, in turn, maps the output terminal of the PLC connected to the input terminal of the pump. The description of such relationships highlights that a value *true* of the Variable switches on the pump. The IEC 61131-3 Submodel is able to describe only the mapping between the variable and the memory location (e.g. %Q0.0) and not the parts relevant the physical connection. Another example of relationship that cannot be represented by the IEC 61131-3 Submodel only is the case of control applications distributed among several PLCs and/or other computing devices. In this scenario it may happen that the applications running into the different devices need to share one or more information (e.g. variables). Let us consider the case of a Variable defined in a IEC 61131-3 program running on a certain PLC and let us assume that it must be set at run-time by a configuration tool or a SCADA application running in another device connected to the PLC. Even in this case, the description of the relationships between internal variables and the external tools cannot be represented in the

definition of the IEC 61131-3 program.

The AAS describing the PLC (or in general the computing device where the IEC 61131-3 program runs) must include the IEC 61131-3 Submodel and must be enriched with the Submodels representing the physical I/O electrical connections provided by the PLC. It is assumed that this last kind of Submodel contains Properties describing the features of the physical connections. With this assumption, instances of ReferenceElement can be used inside the AAS to connect elements inside the IEC 61131-3 Submodel modelling the Variables with the elements representing the physical terminals the Variables are referring to. Therefore, ReferenceElements connect elements of two different Submodels of the same AAS modelling the PLC.

An example of what just said is depicted in Figure 5.11, where a PLC is connected to some devices of the plant. For the sake of simplicity, the plant is described as a whole by means of a single AAS.

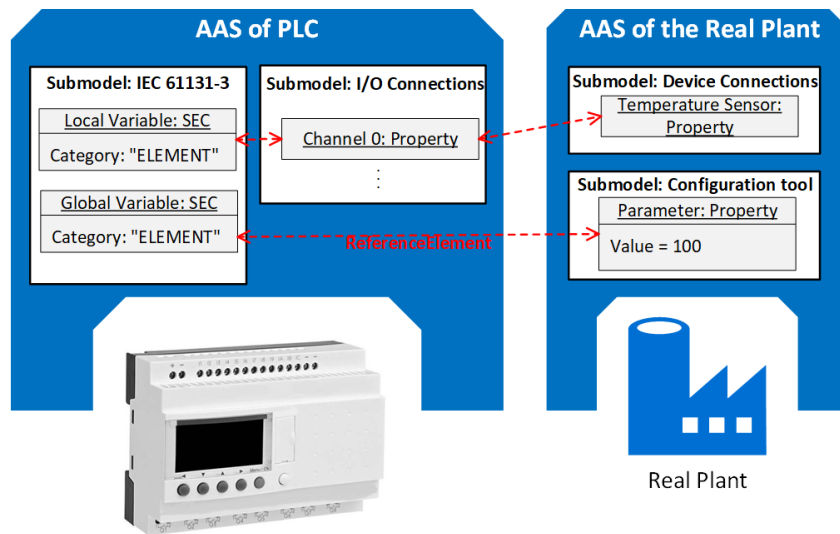


Figure 5.11: AASs representing PLC and the real plant.

The AAS of the PLC contains the IEC 61131-3 Submodel representing the IEC 61131-3 programs running on the PLC. Due to lack of space, only two SECs of category “ELEMENT” are shown representing a local Variable and

a global Variable, respectively. The local Variable is connected to the input terminal of the PLC associated at memory location %I0.0 which in turn is connected to the output terminal of a temperature sensor. This relationship is highlighted by the AASs with red arrows that represent ReferenceElements. The AAS of the PLC presents a Submodel named “I/O Connections” containing the representations of the I/O connections of the PLC, among which there is the input connection associated to the memory address %I0.0 defined, for simplicity, as a Property (i.e., Channel_I_0). Therefore, a ReferenceElement connects the SEC “Local_Variable” inside the Submodel IEC 61131-3 to the Property “Cannel_I_0” of the Submodel “I/O Connections”. Let us assume that a temperature sensor in the plant is connected with the input terminal of the PLC associated to the memory address %I0.0. The AAS representing the plant contains a Submodel named, for convention, “Device Connections” to contain properties relevant to the physical connections of each device of plant. One of these properties is named “Temperature Sensor” and represents the physical connections of the relevant sensor with the PLC. Such relationship is highlighted by a ReferenceElement connecting the Property “Temperature Sensor” with the Property “Cannel_I_0” of the PLC AAS.

As said before the real plant may feature distributed applications running on other devices (e.g. PLC, computers) and exposing configuration values that must be used by the PLC. In the example in Figure 5.11, the Submodel “Configuration Tool” contains a Property (named “Parameter”) modelling a setting value that must be used inside the PLC Programs to fill a global shared Variable. Even in this case, a ReferenceElement can be used to connect “Global_Variable” in the PLC AAS with the “Parameter” Property of the AAS of the plant to describe this kind of relationship.

This approach to represent relationships as ReferenceElements inside the AASs gives an overview of all the connections between elements composing the entire plant, including devices, applications and I/O terminals, making them both structured in a meaningful manner and accessible with a uniform format.

5.5 Case Study: Controlling a Drilling Machine

In this section, our approach defined in this research will be applied to a real industrial use case. The use case considers a PLC running a real IEC 61131-3 program written in ST that controls a drilling machine, as depicted in Figure. 5.12.

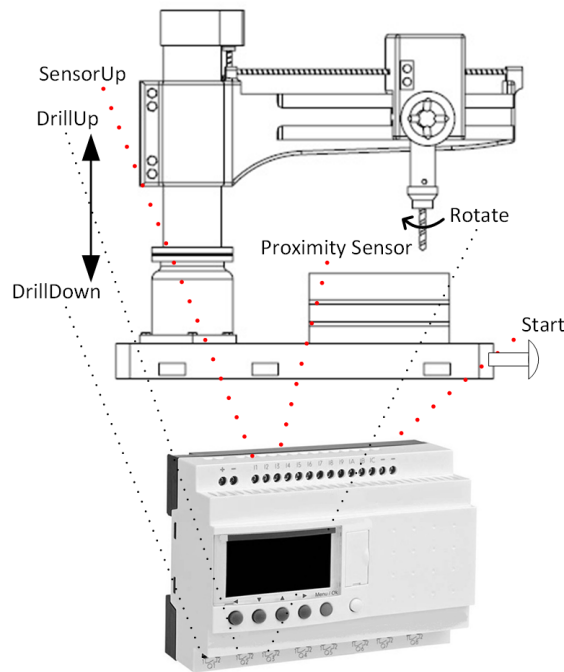


Figure 5.12: PLC and Drilling Machine.

The drilling machine can be moved up or down using a motor that receive the commands `DrillUp` and `DrillDown`, respectively. Two proximity sensors are used to track when the limit values in the range of the vertical movement are reached, i.e. `SensorUp` and `SensorDown`. To activate the rotation of the drill bit of the drilling machine, the command `Rotate` is used. The control program in the PLC is started by an operator using the `Start` command. The ST program controlling the drilling machine is reported in Listing 5.1.

```

1 PROGRAM Drill
2 VAR_EXTERNAL
3     T_PARAM: TIME;
4 END_VAR
5 VAR
6     DrillDown AT %Q0.0 : BOOL;
7     DrillUp AT %Q0.1 : BOOL;
8     DrillRotate AT %Q0.2 : BOOL;
9     Start AT %IO.0 : BOOL;
10    Sensor AT %IO.1 : BOOL;
11    SensorUp AT %IO.2 : BOOL;
12    EndDrill AT %MO.0 : BOOL:=FALSE;
13    Timer01: TON;
14 END_VAR
15 Timer01(IN:=Sensor, PT:=T_PARAM);
16 IF Start THEN
17     IF NOT EndDrill AND NOT Sensor THEN
18         DrillDown:=1;
19         DrillRotate:=1;
20     END_IF
21     IF NOT EndDrill AND Timer01.Q THEN
22         DrillDown:=0;
23         DrillRotate:=0;
24         EndDrill:=1;
25     END_IF
26     IF EndDrill THEN
27         DrillDown:=0;
28         DrillRotate:=0;
29         DrillUp:=1;
30     END_IF
31     IF EndDrill AND SensorUp THEN
32         DrillDown:=0;
33         DrillRotate:=0;
34         DrillUp:=0;
35         EndDrill:=0;
36     END_IF
37 END_IF
38 END_PROGRAM
39
40 CONFIGURATION Config
41 RESOURCE Resource1
42     VAR_GLOBAL
43         T_PARAM: TIME := T#10s;
44     END_VAR
45     TASK MainTask1(INTERVAL :=T#100ms, PRIORITY := 1);
46     PROGRAM MainInst1 WITH MainTask1: Drill;
47 END_RESOURCE
48 END_CONFIGURATION

```

Listing 5.1: IEC 61131-3 PLC Program

The program starts when the button Start is pressed. Therefore, the DrillDown signal is activated. Once the proximity Sensor assumes the value ON because the drill pit reach to the piece to be drilled, the rotation of the drill bit is activated. The drill bit rotates and the drilling machine moves down for a certain time interval specified by a user-configurable global Variable called T_PARAM. Once this time expires, the drilling machine moves up (i.e. DrillUp command is activated) until it reaches the upmost position (i.e. SensorUp is ON), thus stopping the drilling machine. It has been assumed that the control program is restarted when the operator presses the Start push-button again.

The program defines a Configuration named “Config1” including, in turn, the Resource “Resource1”. This resource features the global shared Variable T_PARAM, and a periodic task “MainTask1” with a time interval of 100ms controlling the execution of the Program “Drill”.

According to our approach, both the PLC and the drilling machine are represented with the two AASs shown in Figure 5.13.

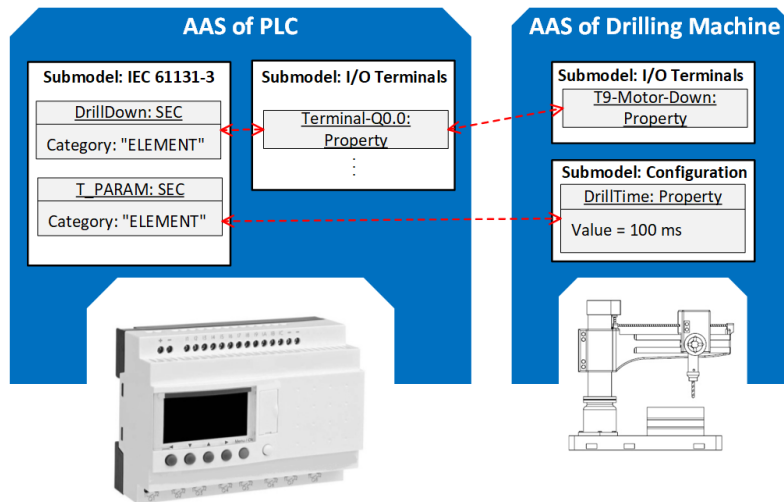


Figure 5.13: Case study showing relationships between Variables and Properties of AASs of a PLC and a drilling machine.

The AAS of the PLC contains a Submodel “IEC 61131-3” discussed in previous sections containing the representation of the elements of the program

in Listing 5.1. Furthermore a Submodel “I/O Terminals” contains the representation for all the electrical terminals featured by the PLC. The AAS of the Drilling machine, instead, contains a Submodel “I/O Terminals” exposing the representation of all the electrical terminals used to send signals controlling the drilling machine, and a Submodel “Configuration” containing all the configuration parameters of the drilling machine. In this use case, it has been assumed that a configuration tool is used to set a value in the Property “Drill-Time” contained inside the AAS of the drilling machine, which represents the duration of a drilling task. The value contained inside this Property must be used to parameterise the PLC program in Listing 5.1 setting the global shared Variable “T_PARAM”.

The program in Listing 5.1 can be represented inside the Submodel “IEC 61131-3” using the approach described in section 5.3. The representation of the program inside the AAS is depicted in Figure 5.14 using an UML instance diagram. For the sake of brevity, only the critical parts are discussed in the remainder of this section. In the following, the term “folder SEC” is used to indicate a SEC containing the value “SET” in its attribute *category*, whilst the term “complex SEC” is used to indicate a SEC containing the value “ELEMENT” in its attribute *category*.

The Resource “Resource1” is represented as a complex SEC containing three folder SECs: “IECPOUs”, “IECVariables”, and “IECTasks”. They organize all the SubmodelElements representing POU, Variables and Tasks related only to the Resource named Resource1, respectively. It is worth noting how in this approach the hierarchy of SubmodelElements reflects the contextual relationship (i.e. *scope*) in the IEC 61131-3 Program.

The Program instance “MainInst1” is represented as a complex SEC organized under IECPOUs of Resource1. Also in this case, this kind of organization reflects the fact that the program “MainInst1” is assigned to the Resource “Resource1”. This SEC features some Properties describing the Program like the programming language adopted, the kind of POU and the name used for the declaration of the Program in Listing 5.1. The ReferenceElement “AssignedTask” is used here to highlight that the program MainInst1 is executed under the Task “MainTask1”, which is represented in

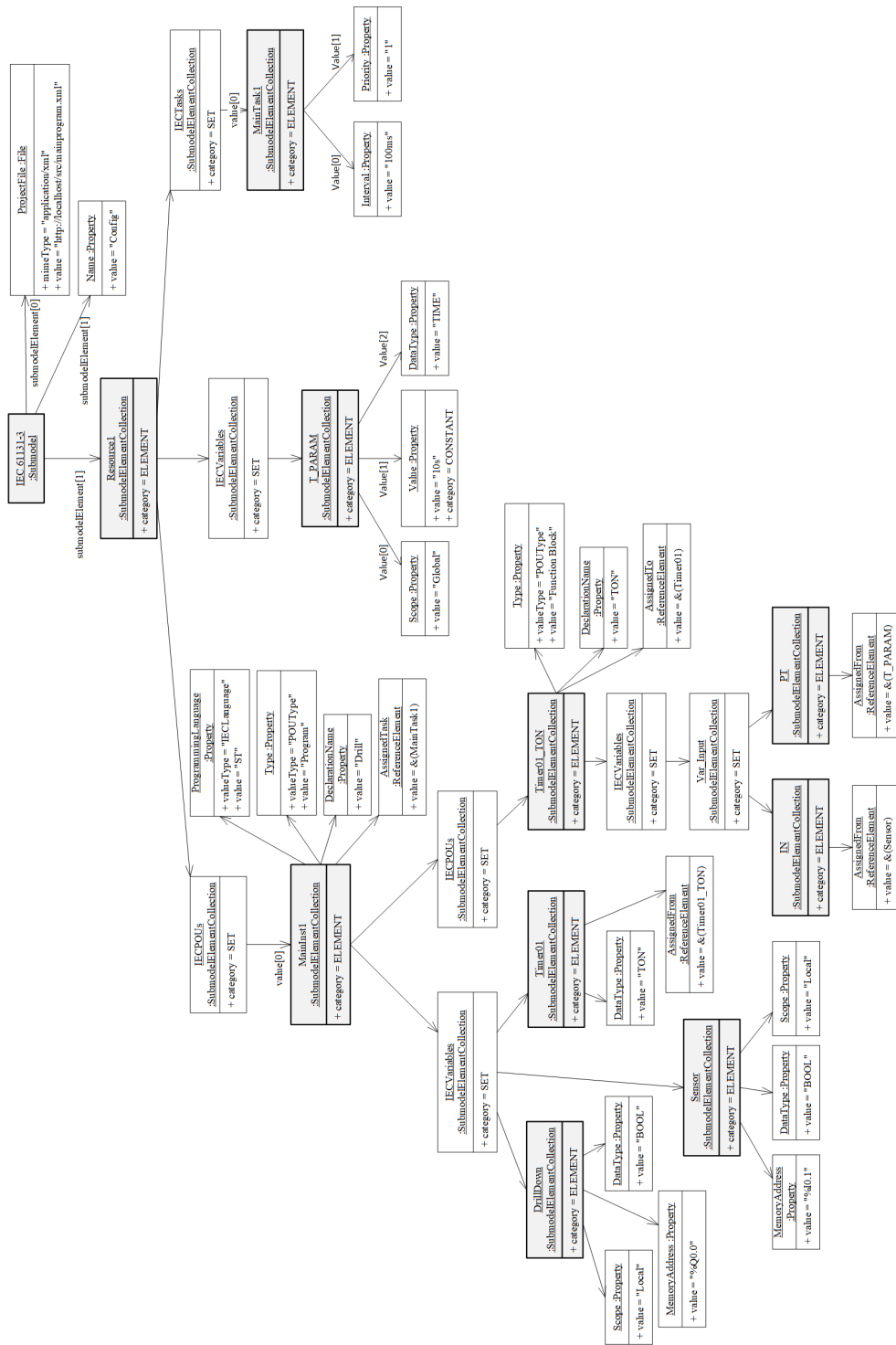


Figure 5.14: Representation of the drilling machine PLC program using a Submodel.

the Submodel as a complex SEC “MainTask1” organized under “IECTasks” of “Resource1”. The attribute *value* of “AssignedTask” contains a Reference to the SEC “MainTask1”, represented in figure with the nomenclature “&(MainTask1)”. This nomenclature is an abbreviation for the reference path used in the AAS metamodel, i.e. “&(MainTask1)” stands for “/IEC 61131-3/Resource1/IECTasks/MainTask1”.

As done for “Resource1”, “MainInst1” features a folder SEC “IECVariables” and a folder SEC “IECPOUs” organizing all the SubmodelElements representing Variables and Function Blocks/Functions defined inside the Program, respectively. Due to lack of space, only few Variables are represented, i.e. DrillDown, Sensor and Timer01, represented using complex SECs. Drill-Down features Properties describing the Variable, like scope, data type and memory address associated (by AT attribute) to the Variable. Similar considerations can be done for Sensor too.

The complex SEC “Timer01” specifies that its data type is TON, therefore it can contain an instance of a Function Block TON as value. Which Function Block instance is contained in the Variable Timer01 is shown by the ReferenceElement “AssignedFrom” that contains a Reference to the complex SEC representing the relevant Function Block, i.e. Timer01_TON contained in the folder SEC “IECPOUs” of “MainInst1”. The name “Timer01_TON” is chosen to logically differentiate in the AAS Submodel the Function Block instance from the Variable Timer01 containing it. “Timer01_TON” features Properties describing the Function Block instance like the type of POU and the Function Block type. The ReferenceElement “AssignedTo” here shows that “Timer01_TON” is assigned to the Variable Timer01. Inside the folder SEC “IECVariables” of “Timer01_TON”, the folder SEC “Var_Input” is used to organize the SubmodelElement representing the input Variables of the Function Block “Timer01_TON” (i.e. IN and PT). For both complex SECs “IN” and “PT”, a ReferenceElement “AssignedFrom” is used to point out which Variables are passed as argument to the instance “Timer01_TON”. For this reason, “AssignedFrom” of IN contains a Reference to Sensor, whilst “AssignedFrom” of PT contains a Reference to T_PARAM, in accordance with the program in Listing 5.1.

The submodel “I/O Connections” features the Property “Output Channel 0” representing the output terminal of the PLC associated to the memory location %Q0.0. Since in the program the Variable DrillDown features the AT attribute set to %Q0.0, a ReferenceElement is used to connect the SEC “Drill-Down” of the Submodel “IEC 61131-3” and the Property “Output Channel 0” of the Submodel “I/O Connections”, as depicted in Figure 5.13. Furthermore, as this terminal of the PLC is connected with the input terminal of the drilling machine to move the driller down, a ReferenceElement is used to connect, in turn, the Property “Output Channel 0” with the Property “T9-Motor-Down” in the Submodel “I/O Terminals” of the AAS of the drilling machine. The Property “T9-Motor-Down” inside the AAS of the drilling machine is the representation of the physical input terminal of the drilling machine.

As said previously, the value of the Property “DrillTime” exposed by the AAS of the drilling machine is used to parametrize the PLC program. In fact, its value must be used to set the global shared Variable “T_PARAM” of the program in Listing 5.1. For this reason, in Figure 5.13 the Property “DrillTime” exposes a relationship with the Variable T_PARAM of the PLC program represented with the ReferenceElement. For space reason this ReferenceElement is not depicted in Figure 5.14, but it is legit realising a ReferenceElement named, for instance, “DependsOn” connected to the complex SEC “T_PARAM” containing a full path to the external Property “DrillTime” of the AAS of the drilling machine.

5.6 Implementation of the approach leveraging OPC UA

The approach we discussed so far leveraging the AAS for the description of IEC 61131-3 programs and their relationships with the controlled devices has been applied for the representation of a program controlling an educational factory model. We developed a scenario where such representation can be used for an easy reconfiguration of the production plant applying some variation inside the PLC program.

The factory model is composed by different working stations, i.e. warehouse, gripper, hoven, and sorting line. To realise this implementation open source software for the PLC runtime and for the editor has been adopted. In particular, the OpenPLC¹ run-time has been installed on a Raspberry Pi to make it work as a real PLC. The IEC 61131-3 program has been developed using the OpenPLC editor and, once uploaded, executed on the Raspberry Pi connected to the factory model.

The results of the research activity discussed in Chapter 4 has been used here to realise the AASs of both the PLC and the factory model using OPC UA as implementation technology. For this reason, we developed an OPC UA using the library `node-opcua-coreaas`, based on Node.js, that we developed to implement OPC UA-based AASs. In particular, an OPC UA server hosting the AASs of the PLC and the factory has been developed on the same Raspberry Pi acting as PLC. The AASs contain all the information relevant the physical connections between the PLC and the factory; in particular, the AAS of the PLC contains the Submodel IEC 61131-3 describing the program and thus the connections between variables and the physical parts.

On a different Raspberry Pi, it has been developed a web application (back-end and front-end) to apply variations to the configuration of the factory model configuration. The frontend is developed using Angular.js² whilst the backend, based on Node.js, includes an OPC UA Client that will be used for the AAS information retrieval inside the OPC UA Server. The scenario is depicted in Figure 5.15.

The PLC program running on the OpenPLC run-time performs a certain control algorithm on the factory model, using the GPIO pins of the Raspberry Pi as I/O terminal to control the working stations. All the connections between the elements of the program and the parts of the factory are represented in the relevant AASs according to the approach discussed in this chapter.

As a test, we introduce some modifications in the production process (e.g. change configuration parameters) using the interface of the web application. The back-end of the web application uses the OPC UA client to explore the

¹<https://www.openplcproject.com/>

²<https://angularjs.org/>

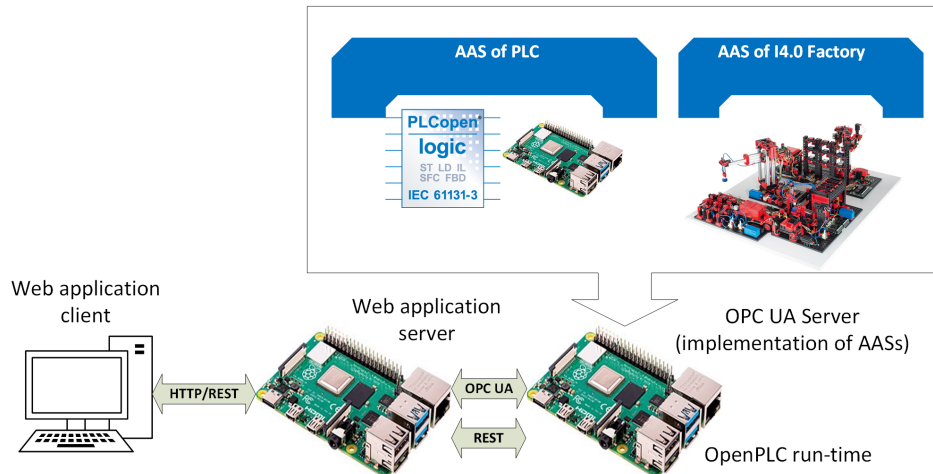


Figure 5.15: Implementation scenario.

AddressSpace of the OPC UA Server in order to retrieve all the information needed to accomplish the variation in the production. Considering a variation of a configuration parameter, such modification is applied in the relevant AAS and, in case such parameter is connected to a Variable of the PLC program, the web application downloads the PLC program file using the information contained in the PLC AAS (e.g. using a property ProjectFile of the configuration as shown in Figure 5.3) and edits the value of the variable associated at the parameter. Finally, it compiles the new program and upload it on the OpenPLC run-time using a REST interface. As a result, the factory model has been re-configured and the changing has been documented inside the AASs.

In a real scenario, the AASs representations maintained by the OPC UA Server may be used by technical teams to introduce real modification to the plant, after the reconfiguration process. The availability of a unique, standardised and complete vision of the plant greatly simplifies the work of the technical teams due to the possibility to better synchronize their works. Finally, the feature to upload the control program into the Raspberry Pi is very useful in a real scenario, allowing an automatic reconfiguration of the software once reconfiguration of the control program has been completed.

5.7 Discussion

The use of the AAS metamodel has the advantage that engineers or technicians coming from different domains can easily understand the relationships between the real plant and the relevant control programs. This was demonstrated with the conducted research where the AAS metamodel has been used as a *lingua franca* for the definition of a domain-specific Submodel, i.e. IEC 61131-3, showing how its generic structure fit for the description of more complex and detailed concepts. In particular, our research investigates how to represent the IEC 61131-3 software model using the simple building blocks of the AAS metamodel in order to represent and fully describe a PLC program and its relationships with external elements, which usually belong to different domains, hence different Submodels. According to the levels of interoperability described in the introduction of this thesis, the approach satisfies the requirement of semantic interoperability, since parts of the PLC Program, like variables, now carry information about the meaning of their content.

Considering an Industry 4.0 scenario where all assets, including PLCs, are represented in the information world with their own AASs, all the relationships between properties of different assets are tracked and available in the network, structured in a standard manner. To understand the advantage of such an approach, it must be considered that nowadays in the development of a SCADA system, usually tag-names for hardware I/O signals and internal program points are assigned and documented in spreadsheets. Such spreadsheets contain which terminal hardware is associated to each tag-name [69]; of course, this kind of documentation is error prone. With the proposed approach, maintenance of control programs of PLCs is greatly simplified because all the information relevant to variables and their connection with the physical parts of the plant are documented and semantically enriched in the AASs. If the developer of the PLC program is changed in a future iteration, the new one can easily track-down, for instance, what a variable is referring to because it points directly to the associated property of a AAS of the relevant device. Furthermore, parameters inside the IEC 61131-3 program can be configured automatically spilling the value from the information contained in the AAS

of the relevant machine or device. This was demonstrated with the use case and the implementation discussed in this chapter.

What is not considered in the research carried out is how the AAS Submodel for the IEC 61131-3 program must be created and who is in charge of its creation. Such Submodel in the AAS of a PLC contains a description of the IEC 61131-3 program currently running on that PLC and features relationships with other elements of the AAS (or different AASs) described in terms of ReferenceElements. We did not address the question of who creates this Submodel and describes such relationships because we considered it out of the scope, but still a valid point to investigate for future work though. By the way, we strongly believe that automatic creation of an AAS Submodel starting from an existent PLC program is feasible and it represents a very important topic from our point of view. According to the novel IEC 61131-10 standard [70], IEC 61131-3 programs are contained in XML-based project file and solution for the automatic creation of the Submodel IEC 61131-3 may use annotated statements in the XML-based project file consisting in additional metadata (e.g. semantic references, AAS-specific information). For instance, a variable declaration may be annotated with a reference to the AAS property it represents. By means of a suited tool, such XML project files can be parsed to retrieve both the IEC 61131-10 tags and the AAS-based tags and using them for the automatic definition of an AAS. An approach like this opens completely new scenarios; for instance, IDE for PLC programming can be extended to include new AAS-related functionalities which in turn help the developer to create the aforementioned extensions for an IEC 61131-10 XML project file. Different tools can thus interoperate each other using the AASs information as a common information source creating completely new ways to approach the production management. Furthermore, the OPC UA Companion specification provided by PLCopen [71] defines a set of IEC 61131-3 based function blocks for mapping an OPC UA Client functionalities that allow controllers to initiate communication sessions to any available OPC UA Server. Such functionalities can be adopted to create interactions between PLCs and their relevant AASs for configurations retrieval. For instance, considering the OPC UA-based implementation of the AAS proposed in this thesis, a PLC

program can take advantage of the Function Blocks defined in the companion specification to create a direct connection with the OPC UA server containing AAS with information of a certain relevance for the configuration of the program. For instance, considering the example described in Section 5.5, an OPC UA Read Function Block can be used to retrieve the value of the Variable “T_PARAM” reading the value of the Property ”DrillTime” in the AAS of the Drilling machine.

The research conducted focused mainly in the use of the description of IEC 61131-3 program in the AAS of the PLC for configuration purposes, as shown in the use case and the implementation discussed in this chapter, but leveraging such approach to achieve flexible manufacturing as intended in Industry 4.0 requires more investigations. This is due to the fact that PLC programs based on IEC 61131-3 are less prone to variations at run-time. In fact, IEC 61131-3 languages use an imperative and procedural approach which does not fit to dynamic variation. Of course, the possibility of applying variations in production can be considered during the development of PLC programs but, in general, such solutions do not scale very well. In future, AASs can cover a more prominent role in the development of more flexible control programs but this requires further investigations, especially considering the implementation of active AASs communicating with each other to take initiative in the production process.

5.8 Publications

The results conducted for this research activities has been published in the scientific journal “IEEE Access” [72].

Chapter 6

A Model for Predictive Maintenance based on AAS

In the previous chapter, the AAS has been used for the description of relationships between software elements of PLC programs and physical parts of the production plant. We discussed how such uniform description of information allows interoperability between tools of different domains enabling new forms of collaboration during the production life cycle. In particular, we stated that maintenance operations can take advantage of AAS information during the maintenance phase.

Maintenance is one of the most important aspects in industrial and production environment. In particular, the approach referred as Predictive maintenance (PdM) aims to schedule maintenance tasks based on historical data in order to avoid machine failures and reducing the costs due to unneeded maintenance actions.

Often, the implementations of maintenance solutions differ on the basis of the kind of data to be analysed and on the techniques and models adopted for the failure forecasts and maintenance decision-making. As stated in the previous chapters, Industry 4.0 introduces the concept of flexible and adaptable manufacturing in order to satisfy a market requiring an increasing demand of customization. The adoption of vendor-specific solutions for PdM and the heterogeneity of technologies adopted in the brownfield for the

condition monitoring of machinery reduce the flexibility and interoperability required by Industry 4.0. For all these reasons that, of course, will be better described in the remainder of this chapter, our research focused on the definition of generic and technology-independent model for PdM where the concept of AAS is used as means to achieve interoperability between different devices and to implement generic functionalities for PdM.

The model we defined is then applied for the description of a case study considering an Industry 4.0 Cloud-based PdM maintenance program for 100 milling machine.

6.1 Introduction

Maintenance is of paramount importance for industrial or production plants as it aims to maximize the production whilst reducing the costs as much as possible. In fact, maintenance costs are some of the main components of the total cost of a production facility as it is estimated that they represent between 15 and 60 percent of the cost of goods produced, where one-third of such maintenance costs is wasted for unneeded or improperly carried out maintenance operations [73]. Therefore, the overall cost of the plant strictly depend on the strategy adopted for maintenance management. For this reason, different maintenance strategies are used to maintain high the production efficiency [74]. One of this maintenance strategies is PdM which schedule maintenance operations when either a deterioration of the machines or a degradation in the performances is detected.

In the context of Industry 4.0, where production must be efficient to face the demand for a high level of product customization, maintenance plays an important role as avoiding breakdowns, and thus loss of money, is a key requirement in a challenging market that requires high efficiency and availability. Considering the presence in literature of different approaches for the realisation of PdM solutions and different technologies in both the brownfield area (e.g. sensors, fieldbus) and the IT area, it is difficult defining a PdM solution that can adapt to the variation of the original production configuration. This, of course, puts new constraints on the flexibility of the smart

factory. For all these reasons, new mechanisms are required for the horizontal integration hiding implementation details and guaranteeing a communication channel between devices, regardless of both their manufacturer and technologies adopted. Using such an approach, a device can be easily replaced with an equivalent one providing the same functionalities, i.e. production functionalities and PdM functionalities. Furthermore, vertical integration must be guaranteed because heterogeneous data coming from devices and directed to enterprise levels must be presented in a uniform manner to allow a collaboration between all the components of the PdM solution.

An approach for the definition of a PdM program that satisfies the requirements of flexibility and interoperability, as demanded by Industry 4.0, must address two main objectives: 1) defining generic functionalities for the description of a technology-independent PdM solution and 2) hiding the heterogeneity and complexity of the OT level. Groba et al [75] described such objectives and analysed the aspect of PdM and the challenges to face considering such an approach. In this work, authors defined a PdM framework integrating the diversity of different PdM techniques, thus addressing the objective 1), but they identified that one of the biggest challenges consists of describing the shop floor equipment and corresponding condition indicators in a uniform manner, hence addressing 2). In general, both the objectives 1) and 2) are faced either separately or partially but, at the best of our knowledge, there is no solution addressing both together in literature. Traini et al. [76] proposed a framework to define PdM solutions for a generic manufacturing tool but it is based on ML techniques only and does not consider the flexibility as a requirement. A framework defining a flexible maintenance platform is described in [77] proposing an approach that uses modularisation of maintenance functions, but such approaches is based only on AI solutions for failures forecasting and furthermore the integration of devices and theirs data is addressed partially.

As will be described in the following, our approach is strongly based on modularisation of PdM functions. A similar idea has been used by [78] to define framework based on functional blocks collecting key functionalities for the components constituting a generic PdM solution. The aim of this framework

is reducing the complexity of PdM programs management but, differently from our approach, it does not consider the interoperability of devices and data as requirements.

It is worth noting that all the issues involved to address the objective 2) are the same faced nowadays by CPS and DT in the context of Industry 4.0. For this reason our approach is strongly based in the adoption of AAS to cope with the objective 2). This choice is confirmed by [79] where, in production automation, an appropriate infrastructure consisting of components with uniform interfaces is of utmost importance for condition monitoring and PdM (see [80]).

In this research activity, we defined a logical model for PdM than encompass all the “common factors” among the PdM solutions present in literature, generalising all the aspect that a PdM solution implementation should cope with and representing them in suited logical blocks collecting the relevant functionalities. Abstraction and generalisation of PdM functionalities is the foundation of our model to address the objective 1). To address the objective 2), instead, our model leverage on AASs since they create a standardised abstraction layer above assets using different technologies that cannot interoperate each other. As confirmed in [81], the AAS model can be used to expose different data modules in a uniform manner by means of the standardised external interface of AAS, with the result of making the devices seamlessly interoperable, solving the issue concerning the massive heterogeneity of technologies and information models adopted in the industrial environment. In chapter 5, we already used the AAS to abstract and describe the complexity of plant configuration against the software model of IEC 61360 for PLC programs but in the research described in the following we propose an approach leveraging the AAS for the description of a PdM solution in terms of generic and technology-independent functionalities.

6.2 Overview on Predictive Maintenance

A maintenance program based on PdM approach aims to prevent catastrophic failure using some indicators of machinery conditions to schedule maintenance

tasks and to detect the kind of failure, hence is referred in literature also as condition-based maintenance (CBM). In fact a CBM solution uses actual operating condition of the equipment to predict the future state of the machine using a model based on historical data [74]. The foundation of predictive maintenance is the condition monitoring (CM) process [82], where sensors are applied in machinery to continuously monitor signals, or other appropriate indicators, to assess the health of the equipment [83]. For instance, the ac component of the output of pressure sensor (e.g. noise) can be used to detect blockages in the pipe of a plant [84], or vibration analysis gives an indication about reliability and safety of rotating machine [85].

PdM is a so broad subject area that literature presents several different approaches to implement a maintenance program. Often PdM solutions differs each other on the basis of the kind of signals used for the health condition assessment of the machine, the kind of machine to be maintained, the approach adopted for the retrieval of the indicators and the approach used for the failure forecasting, among others. For this reason in the remainder of this section, the main parts composing a PdM approach will be pointed out so that they can be used for the definition of the generic PdM model that is the subject of this research activity.

A CBM program can be divided in three main parts: data acquisition, data processing, and maintenance decision-making [86].

6.2.1 Data Acquisition

The first step in a PdM program consist in collecting data from the machinery that is the subject of the maintenance program. Such data, usually coming from sensors embedded or applied on the machinery itself, are used as indicators of the health condition of the devices. The kinds of data collected vary case by case depending on the machine to be maintained and hence on the sensor used. Hashemian and Bean [84] identifies three major categories for predictive maintenance depending on both the kind of information acquired and the source adopted. The authors show that some techniques may adopt signals coming from existent sensors embedded in the machinery to reveal

its current status; other techniques involves the adoption of new sensors, like accelerometers for vibration or acoustic sensors for leaks detection; finally, other techniques involves the injection of test signals into the equipment to measure the performances [87].

In the past, the adoption of a PdM approach was considered quite expensive because old equipments required the additions of new sensors [88] and because new tools for the monitoring were necessary [89, 83]. Nowadays, new technologies in the field of data acquisitions, like IoT and sensor technology, make the adoption of PdM solutions more accessible. In particular, the IoT is the foundation of condition monitoring in the fourth industrial era, utilising sensors embedded in machinery to acquire data and communicate over the Internet infrastructure with cloud-based solutions that analyse such data [90].

It is worth noting that often the brownfield contains already smart equipment providing more information than just basic data over intelligent communication protocols, but systems from different vendors may run in different parts of the production process and a way to allow them to understand the information exchanged must be provided (e.g. industrial gateways).

6.2.2 Data Processing

The next step after the data acquisition consists of a series of processes of data manipulation. Some involves data cleansing in order to remove, for instance, sensor errors that may affect the data analysis. A pre-process step involves the reduction of the volume of data (i.e. aggregation) to pass only the selected and extracted indicators (i.e. feature extraction) [88] to the forecasting and/or decision-making algorithms.

The techniques adopted to process and analyse data mainly depend on both the types of data collected and the algorithms used to reveal the condition of the machine. In [86], data are classified in three types: value type, waveform type, and multidimension type. Value type are data collected at a specific time epoch and containing a single value, like temperature, humidity, or pressure. Waveform type are data collected in time series, like vibration or

acoustic signals. Multidimension type contains multidimensional values, like images coming from infrared thermographs.

In general, data analysis and signal process techniques operate in the time-domain, frequency-domain, or time-frequency domain. As said previously, the kind of techniques depends on the type of data processed and the feature to be extracted. Examples of time-domain analysis are statistical indicators like mean, standard deviation, root-mean square, kurtosis, skewness but also peak, peak-to-peak interval, crest factor in case of time waveform signals. With frequency-domain analysis, instead, it is possible identifying and isolating certain components of interest not accessible with time-domain analysis. For instance, considering data coming from vibration analysis, in the time domain we see only the sum of the effects of all the sources of vibrations. But, with frequency-domain analysis we can isolate such sources of vibrations because they operate in different frequencies. Examples of frequency-domain analysis is the widely used spectrum analysis by means of Fast Fourier Transform (FFT). Features in frequency domains are power bandwidth, mean frequency, harmonics, peak frequencies, etc. Time-frequency analysis, instead, is common to handle non-stationary waveform signals and uses time-frequency distributions representing either the energy or the power in two-dimensional function. Some features in time-frequency domain are spectral entropy and spectral kurtosis, but more indicator exists in literature [86].

All features extracted in this step are indicators of the health condition of the equipment and can also be used for an estimation about the remaining operational time until a failure occurs.

6.2.3 Maintenance Decision-making

The techniques adopted for maintenance decision-making in CBM are classified in diagnostics and prognostics. The former deals with finding the source of a fault, whilst the latter deals with estimating when a failure may occur in future. Even though prognostics is better than diagnostics since the former try to prevent a failure whilst in the latter the failure already occurred, diagnostics techniques are often used as complementary support for prognostics

because prognostics cannot prevent all failures. Furthermore, diagnostics results can be used as feedback to improve the accuracy of prognostics solutions.

Decision strategies for machine fault diagnostics usually are procedures that create correlations between data and/or features collected in the Data Acquisition step with features of the faults, creating then a sort of classification over measurement data. This mapping process is also called pattern recognition. Usually, diagnostic solutions adopt statistical approaches or AI over the information collected with condition monitoring, but model-based solutions exist too. Some statistical approaches are hypothesis test [91], statistical process control (SPC) [92], and cluster analysis [93]. AI solutions, instead, has shown better performance compared to conventional approaches to diagnostics [86]. Common AI techniques include the adoption of artificial neural network [94], fuzzy-neural networks [95], and evolutionary algorithms [96]. Model-based solution instead uses physics or mathematical model to simulate the behaviour of the monitored machine [97].

Most of the strategies adopted for prognostics involve the prediction of the so-called Remaining Useful Lifetime (RUL), which indicates how much time is left until a failure occurs. Other solutions usually adopted in high-risk environment (where a failure is catastrophic) consist in calculating the probability that a failure occurs between two inspections of the plant. By the way, approaches considering RUL estimation constitutes the majority in literature. There are three families of RUL estimation models: similarity model, degenerate model, and survival model [98]. Similarity models are based on historical failure data that combines the RUL prediction of a test machine with the behaviour of a similar known machine. The degenerative models, also referred in literature as degradation models, estimates the RUL predicting when the condition indicator will cross a failure threshold. These models are most useful when a known value can be used as a threshold indicating a failure, which is not always available. The survival model, instead, is a statistical method used for time-to-event data. It is particularly useful when a complete history of failures is not available but only data about life span of similar components and/or some other variables (i.e. covariates) that correlate with the RUL. In particular, covariates are also referred as environmental variables

and indicates the regime in which the component operates.

Of particular interest here are the solution based on AI because, due the improvement in this area in the last years, lots of new solution for PdM based on AI have been proposed. In [99] a multiple classifier PdM system is proposed. Paolanti et al. [100] provides a new PdM methodology based on ML for a cutting machine predicting different states for the machine with an accuracy of the 95%. Koca et al. [101] trained an artificial neural network using Mean Time-To-Failure (MTTF) values and the past failures history of a robot system to predict future faults.

The adoption of AI-based solution for predictive maintenance perfectly fits the vision of the fourth industrial revolution [36]. Since the initial cost of a PdM maintenance program is quite expensive, in the context of Industry 4.0, where an high degrees of connectivity exist between partners of a value-chain network, some facilities and tools become services that can be used on demand, contracting or subcontracting activities and thus avoiding the expanses of building new infrastructures for PdM [89]. For these reasons all the biggest IT companies started to propose their PdM solution based on ML and cloud technologies: Amazon with their AWS Solutions¹, Microsoft with Azure ², and Google with Google Cloud Platform³.

6.3 AAS-based Model for Predictive Maintenance

The research carried out on PdM allowed us to identify all the aspects common to the PdM solutions regardless their actual implementations. We defined an approach for the description of a PdM solution on the basis on a PdM model leveraging on the concept of the so-called Logical Block (LB). A LB is a modular element that group functionalities relevant to a specific aspect of

¹<https://aws.amazon.com/it/solutions/implementations/predictive-maintenance-using-machine-learning/>

²<https://docs.microsoft.com/en-us/azure/iot-accelerators/iot-accelerators-predictive-walkthrough>

³<https://cloud.google.com/blog/products/ai-machine-learning/solution-implementing-industrial-predictive-maintenance-part-iii>

the PdM, like data acquisition or data manipulation. All the functionalities collected under a LB are specific to an aspect of PdM and abstract specific operations for the PdM process regardless how such operations are actually implemented. For the sake of clarity, with the term functionality here is also intended information, data structures and models, and not operations only.

LBs and their functionalities are meant to be modular and cooperating elements, so that they can be used to describe a PdM solution (entirely or part of it) in a generic manner without considering implementation details. One of the advantages of such an approach is a simpler specification of which PdM functionality should be assigned to the components of the IT and OT infrastructure; in this manner, if a component of the PdM solution is described in terms of generic LB functionalities it is possible to define a **role** for that component. Such a role identifies a sort of equivalence class between all the devices implementing the same functionalities. As a consequence, this makes the replacement of a device with an equivalent one seamless from the point of view of the PdM program. As already said in the introduction of this chapter, this strategy allows to address the objective 1) required for the definition of a flexible and interoperable PdM solution.

To address the objective 2), instead, the concept of AAS and I4.0 Component is used for some devices present in OT infrastructure since they can be exploited to face the problem of heterogeneity of technologies present at that level. The common interfaces and the semantically enriched information provided by the AAS make this last the foundation of the PdM model here presented. In fact, AAS achieves interoperability at the lowest level of the production infrastructure and thus allows a PdM program to be adapted for production reconfiguration. For brevity, in the reminder of this chapter the term “AAS-enabled device” will be used to identify all the devices exposed by an AAS.

In the following, for the description of the PdM model here discussed will be used a bottom-up approach, where the most fine-grained elements composing it are described first, completing then with the high-level view of the model where all the interactions between components are highlighted.

6.3.1 Logical Block for PdM

The most important elements defining the AAS-based PdM model are LBs since they abstract all the functionalities required for the PdM process. Such an abstraction generalises and modularizes the description of the PdM solution and allows the implementation of same functionalities and operations using different technologies and approaches. LBs functionalities can be realised following different standard specific for the aspect represented by an LB, so that the PdM model does not put any limitation on the standards or guideline to be adopted. For instance, standards like VDMA 24582 [80] or ISO 17359 [102] can be used to implement LB functionalities related to the condition monitoring. As will be described in the next subsection, for AAS-enabled devices the LBs will be implemented inside specific well-known AAS submodels.

The LBs specified for the AAS-based PdM model have been defined on the basis of the state of the art of PdM summarised in section 6.2 and are depicted in Figure 6.1 showing some examples of functionalities implemented for every LB.

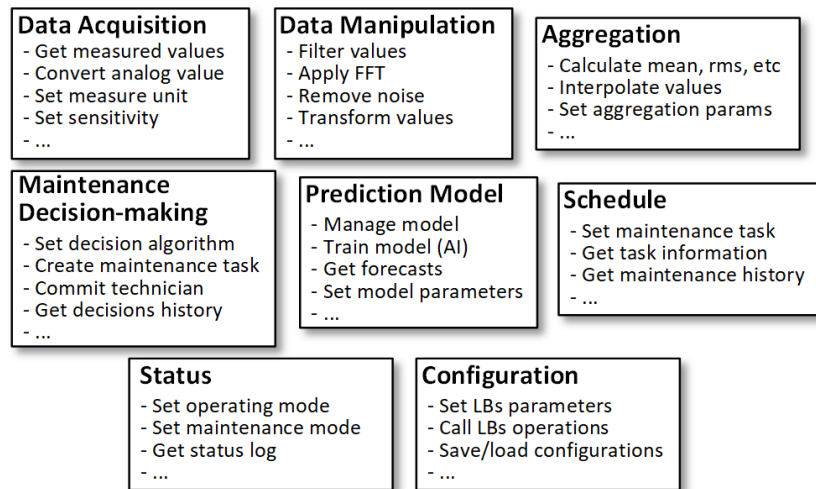


Figure 6.1: The logical blocks implementing generic functionalities related to PdM aspects.

In the following the aspects related to the LBs will be described:

Data Acquisition (DA) All the functionalities to access data coming from sensors or other sources are provided by this LB. Example of such functionalities are functions like the conversion of the output of a transducer to a digital parameter representing the physical quantity. Such digital values may be enhanced with more quality parameters, like calibration or timestamp.

Data Manipulation (DM) This LB contains operations that perform analysis of signals and computes meaningful descriptors from raw measures (usually coming from DA). It also performs transformations on signals (e.g., FFT) and applies algorithms for features extraction. Most of the approaches for data processing lies in this LB.

Configuration (Config) This LB provides an interface for the configuration of other data-processing LBs exposing parameters and management functions. It is one of the most important LBs since it is the only one providing functionalities used to parametrise the functionalities of other LBs. For instance, some configurations for DA may include the relative position of the transducers, monitoring polling rates and calibration parameters, among others. Of course, parameters and functionalities of Config may be strictly dependant from the implementation of the other LBs. For instance, considering a block DA implemented with OPC UA and using the Subscription mechanism for data retrieval, a Config block may be used to configure parameters relevant to the publishing interval or sampling interval.

Aggregation This LB provides the functionalities needed to perform data aggregation of all the different data coming from logically “underlying” devices. Such a block may include mechanisms of Sensor Data Fusion when, for example, the data monitored of a complex device come from sub-devices or sensors composing it. It is worth noting how this perfectly fits with the concept of AAS because it allows the representation of complex devices by means of composition of the AASs of their sub-devices. Therefore, the Aggregation block implemented in the AAS of

a complex device can use data coming from the DM blocks contained in the AASs of its sub-devices. Of course, there could exist aggregation of aggregations, thus input data of an Aggregation LB may come from other Aggregation blocks. Such aggregation hierarchy is needed to manage the large amounts of data coming from sensors.

Prediction Model This LB identifies all the functionalities and facilities required for the diagnostics and prognostics of the monitored machinery. For instance, considering a PdM solution based on Artificial Intelligence, this LB may consist of a neural network-based model or decision tree-model, but lot of different solutions may be adopted too. When it is possible, the models here provided are trained using historical data including health indicators and faults of machines. Such data are gathered and manipulated using functionalities of other LBs. Furthermore, the models may be constantly trained using data gathered in real time from AASs, and prediction errors can be used to improve the accuracy of the model. The output of the Prediction Model block depends on its implementation, and thus on the PdM technique adopted. Examples of output may be a type of failure, an indicator of the machine's status or the RUL. It is worth noting that technical personnel working on data analysis and the tools they use are considered entities implementing functionalities of the Prediction Model LB.

Maintenance Decision-making All the functionalities for the analysis of the information given by the Prediction Model LB to schedule appropriate maintenance actions for the predicted faulty machines are implemented in this LB. This block involves the facilities for the scheduling of maintenance tasks, the eventual commitment of available technicians for the maintenance, and is in charge to change the operational state of the machine (i.e. changing the operational state from "working" to "maintenance"). All these kinds of operations change information on the proper submodel in the AASs of devices, similarly to the approach presented in [47]. In general, most of the functionality provided by a Computer Maintenance Management System are considered being part

of the Maintenance Decision-making block. The output of this LB may be used as a feedback for the Prediction Model LB to adjust the accuracy of the model adopted or check its correctness.

Schedule This LB contains all the information relevant the maintenance tasks like the date and the duration of the maintenance and the credentials of the operator committed for the maintenance operation. This LB also includes the history log with all the maintenance operations performed on the machine and, eventually, whether a replacement with a new one occurred.

Status This LB contains all the information about the status of the machine. In particular, it highlights when the machine is in operating mode or in maintenance mode. This information may be useful to check the general status of the plant or to label eventual data still being collected from the machine even during a maintenance operation.

6.3.2 AAS Submodels supporting PdM

The AAS-based PdM model provides specific Submodels to implement some LBs functionalities for AAS-enabled devices. The structure of a submodel allows the definition of the PdM functionalities in terms of properties and operations, both semantically annotated. The definition of such well-known Submodels allows the assignment of some steps required for a PdM solution to AAS-enabled devices using a common and standardised representation. Furthermore, since AAS allows for composition, functionalities in submodel may be represented as composition of functionalities of the AASs of sub-devices or logically underlying devices. For instance, configuration functionalities of a high-level device may be represented as a composition of several configuration functionalities of underlying devices. The definition of LBs as modular components perfectly fits with the nature of AAS structure so that composition of PdM functionalities may be represented as composition of properties and operations inside the relevant AASs.

For the PdM model here described, we defined two main Submodels im-

plementing all the LB functionalities required for condition monitoring and the administrations of data relevant to maintenance operations scheduled for a device; they are named “Condition Monitoring” and “Maintenance”, respectively. These two Submodels are depicted in Figure 6.2 showing that exist relationships between them. It is worth noting that the presence of an LB inside a Submodel is not mandatory, thus the implementation of some functionalities is strictly dependent by the case in exam.

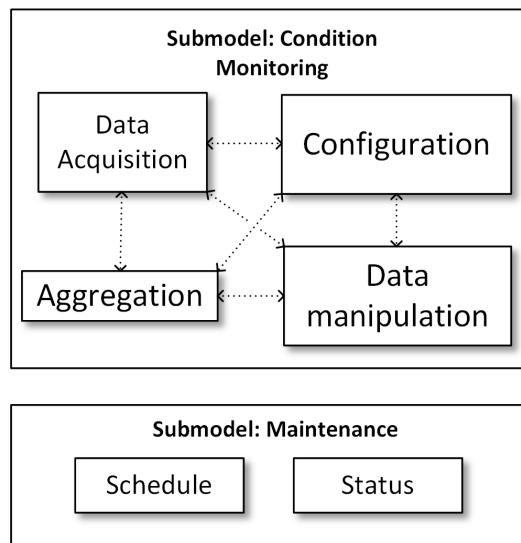


Figure 6.2: Submodel definition for Condition Monitoring and Maintenance and the LB they implement.

The Submodel “Condition Monitoring” implements the blocks DA, DM, Config and Aggregation. All the LBs in the Condition Monitoring submodel may interact to each other, as depicted in figure by means of dotted arrows. Such interactions may represent data flows, events dispatching, function calls or parameter settings.

The submodel “Maintenance” implements the functionalities of the block Schedule and Status, therefore it exposes all the information concerning the maintenance tasks and operational condition of the device and the operations to commit new maintenance tasks.

The LBs “Prediction Model” and “Maintenance Decision-making” are not considered inside the submodels definitions provided for the AAS-based PDM

model because such high-level functionalities with high-demanding computational requirement may not be easily implemented in an AAS-enabled device.

6.3.3 Description of the PdM model

The general structure of the AAS-based PdM model is depicted in Figure 6.3. From a high-level point of view, the model is divided in two main parts: the Operational Infrastructure (OI) and the Prognostics & Maintenance Management Infrastructure (PMMI). OI encompasses all the components of the PdM solution collecting and manipulating the data used for maintenance prognostics. Examples of such elements are the machines to be maintained, industrial gateways, industrial PCs, but even high-level tools like MES and ERP may be considered being part of OI. The PMMI, instead, encompasses all the components of the PdM solution using data coming from OI to forecast machine failures and schedule maintenance actions. Examples of such elements may be AI models (e.g. Recurrent Neural Network), tools for data analysis and software for the maintenance management.

The red arrows in Figure 6.3 represent the relationships between components and the data streams from low-level devices to the PMMI elements for the decision-making task. Green arrows, instead, represent the interactions between the topmost components and the maintained devices where the former set the operational status of the latter and commit maintenance tasks.

OI contains AAS-enabled devices providing both data needed for the condition monitoring and functions for data manipulation required from the first steps of the PdM process (e.g. Smart Device, Industrial PC, Gateway). In general, what is considered belonging to Operational Technology (OT) is part of the OI, thus devices like sensors, actuators, machinery, but also PLC, SCADA, DCS, may be considered part of it. It is important to point out that IT components like databases, industrial PCs, or edge devices like gateways may be part of OI too. The presence of AAS is mandatory for the devices at the lowest levels of the infrastructure (i.e. brownfield) because, as said in previous sections, such devices are the ones featuring a high degree of heterogeneity in the technologies and data representation adopted. The PdM

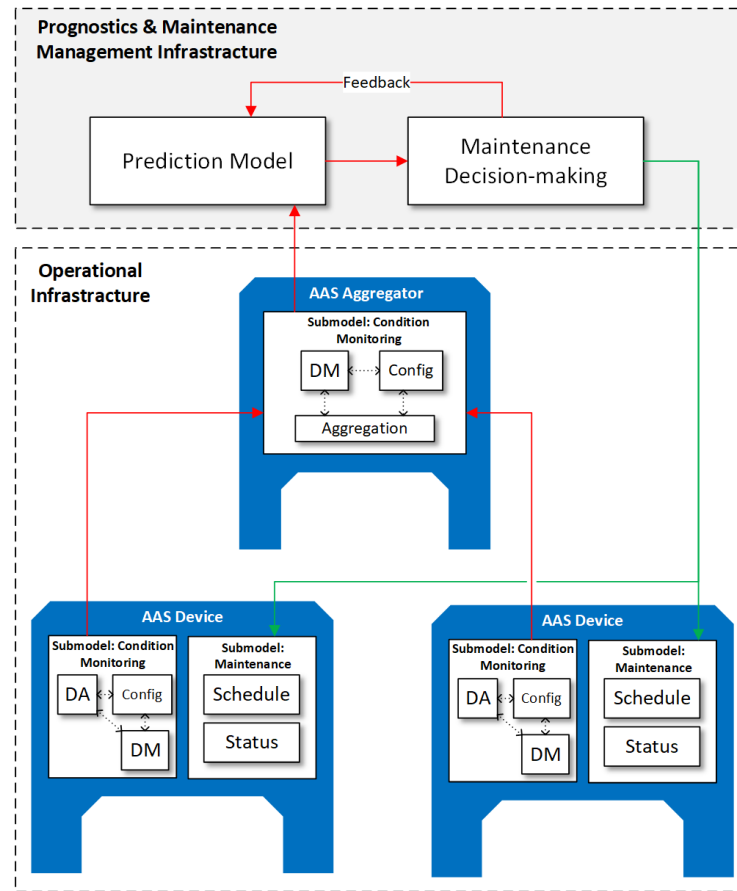


Figure 6.3: AAS-based PdM model for predictive maintenance.

model requires only that every PdM component that need to interact with an AAS-enabled device must be able to communicate with the AAS API and thus understand its semantics.

Differently from OI, the nature of entities composing the PMMI is not defined in terms of device types but in terms of which functionalities they implement. PMMI consists of IT elements and software components providing all the functionalities needed for data analysis, failures prediction, and scheduling of the maintenance tasks. The nature of such components is not specified but they are described only in terms of the functionalities they provide (i.e., their LBs). Such functionalities may be implemented on devices of the IT infrastructure and/or in the Cloud (in case of Cloud-based PdM), so that how functionalities are implemented strictly depends on the solu-

tion adopted for PdM. For instance, the prediction functionalities defined for PMMI may be implemented either by a Recurrent Neural Network (Artificial Intelligence-based solution) or by a physical person consulting a visual tool for data analysis; even if the former is a software component and the latter is a person, both of them are considered entities of the PMMI.

The AAS-enabled devices depicted in Figure 6.3 are differentiated in “Device” and “Aggregator” but such classification is not formal and is used just to clarify which role an entity plays in the OI. As said previously, the role an entity plays depends on the LB it implements. In figure, AAS Device represents a generic Device that provides condition monitoring features and need to be checked eventually for maintenance task. Similarly, AAS Aggregator identifies a device that it is not a direct subject of the maintenance process, but a component of the maintenance program. The LBs it implements suggest that the role of AAS Aggregator is that of collecting all the data coming from different AAS Devices and performing some sort of manipulation (e.g. data aggregation, sensor data fusion) before sending them to other entities.

Roles can be defined just picking specific LB functionalities and combining them properly and, viceversa, the role of an entity can be discriminated just looking at the LBs it implements. This aspect of the PdM model allows the definition of a sort of equivalence classes for PdM components because such roles are defined in terms of collection of generic PdM functionalities. The advantage of the proposed model is describing a device using a role so that it can be replaced seamlessly with another device of the same role.

The structure of the PdM model allows the description of every PdM program based on I4.0 Components at low levels of the infrastructure. In fact, their AASs allows the description of all their functionalities in terms of LBs generic functionalities, thus better defining the roles that such PdM components play in the whole PdM solution. Generalisation realised using the model allows easy reconfiguration and extensibility of the production systems, increasing the integration of all the different parts of the PdM solution. The AAS is the foundation of this abstraction mechanism for low-level devices featuring different implementation for similar functionalities, different information models or different communication protocols. The presence of

I4.0 Components allows their eventual replacement in a transparent fashion from the point of view of both the PdM solution and the production system. This important aspect is better point out in the next section.

6.4 Case study: Modeling a Cloud-based Machine Learning PdM solution

The AAS-based model will be used for the description of a use case consisting of a simplified PdM solution based on machine learning techniques and cloud technologies for the maintenance of 100 industrial milling machines. This case study shows that the model and its LBs allow the description of a PdM solution in a high-level perspective to better specify the functionalities and roles played by the components of the infrastructure.

As already said, such description is possible at the lowest level of the infrastructure only because the common standardized structure and communication interface provided by AAS hide the implementation details of the relevant asset and of the technologies used to implement its functionalities. In fact, the generic LB functionalities provided by an asset can be implemented using any suitable solution because the AAS abstracts the implementation details and provides its information in a uniform (and semantically annotated) manner exposed by its standardized API.

For the sake of simplicity, the AASs mentioned in this case study are considered embedded in their relevant assets, which provides computation and communication capabilities (e.g., microprocessor and network access). In this use case scenario Microsoft Azure is used as cloud infrastructure.

6.4.1 Description of the use case

The case study in exam consists of a multi-class classification problem, thus a ML algorithm is used to create the predictive model that learns from data collected from 100 milling machines and exposed by their AAS. For the training of the prediction model are considered four different data sources: real-time

telemetry data from sensors (time-series), error logs, maintenance history, and machine information.

In order to respect the desired information structure for the model training, telemetry data must be manipulated applying simple measurement unit conversion or filtering operations. Furthermore, the amount of telemetry data collected from each machine is aggregated every hour using an Industrial PC. This first aggregation step is done averaging the data collected every hour. An Edge Gateway, instead, collects telemetry data from all the milling machines and labels the records accordingly with the relevant machine names (i.e., serial number or machine id). Finally, data are aggregated choosing a lag window of 24 hours and using mean and standard deviation as rolling aggregate measures. All the records are then sent in the cloud where eventually other data manipulation processes may be applied to generate the data set for the training of the ML model.

Error logs are generated by the milling machines and contains only non-breaking errors that do not constitutes failures. The error timestamps are rounded to the closest hour since telemetry data are aggregated at an hourly rate during the first aggregation step, as previously said.

Maintenance history is a collection of record generated when a component is replaced during a scheduled inspection or due to a failure. For the sake of simplicity, maintenance history is considered already available because collected in the past and stored in the databases.

Information about the machine, instead, contains the model name of the milling machine and its age in terms of years of service.

6.4.2 Representing the use case using the PdM model

Considering the description of the case study previously provided and using the means specified by the PdM model, we can say that the AASs of the milling machines implement the LBs DA and DM, whereas the Industrial PC and the Industrial Gateway require the functionalities of the LB Aggregation. All of them, instead, must implement the functionalities of the LB Configuration because they must expose the proper functionalities used by an external

configuration tool to set the right parameters for data acquisition, data manipulation and aggregation, respectively. All these LBs are implemented in a proper “Condition Monitoring” Submodel of the AASs.

The telemetry data are collected using sensors either directly applied or embedded to the milling machines. Telemetry data consist in vibration, voltage, rotation, and pressure. Voltage and pressure are gathered using sensors already embedded in the milling machine. The rotation speed is obtained using a rotation speed sensor applied to the machine and connected to a PLC. Finally, the vibration data are gathered using a wireless vibration sensor. Figure 6.4 shows how telemetry data are exposed by the AAS in a uniform manner regardless how data are retrieved from sensors.

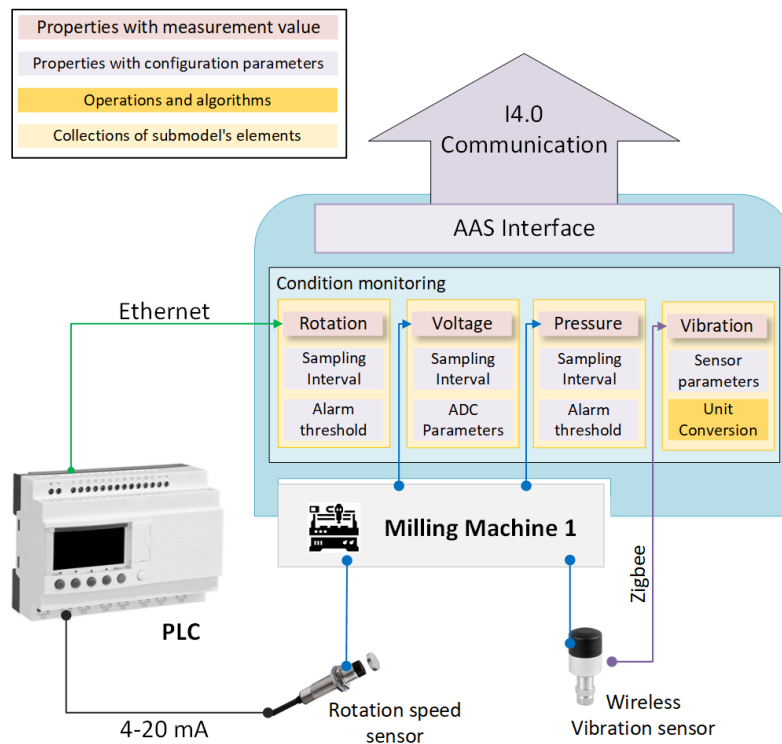


Figure 6.4: The structure of the AAS of a milling machine. The submodel “Condition Monitoring” provides measurement value, configuration parameters, alarms and operations in a uniform manner by means of the AAS interface. All the implementation details and underlying technologies adopted for data collection and manipulation are hidden by the AAS.

Telemetry data are collected under the Submodel “Condition monitoring” including also the measurement values, alarms thresholds, configuration parameters for data acquisition, and operations for data manipulation. The added value in the adoption of the AAS here is that logically different properties and operations are modelled using the same entities coming from the AAS metamodel, thus hiding the details of the technologies required to gather data or to implement the operations in the milling machine. Furthermore, the AAS allows to configure how such data are retrieved (e.g. sampling rates) or exposed by its interface. Suitable operations may use asset information and configuration parameters and be applied to data collected to expose values in a suitable manner for the ML solution of the use case, e.g. measurement unit conversion.

The complete scenario of the case study with the interaction between all the components involved in the PdM solution are depicted in Figure 6.5.

Since telemetry data exposed by the AASs of the milling machines are already well structured in a uniform manner, the AASs of the Industrial PCs are configured to collect measurement values from each relevant milling machine and saving them in a time-series database. In turn, the AAS of the Edge Gateway is configured to collect the data of all the milling machines, putting them all together and calculating the mean and the standard deviation in a window of 24 hours and saving them in a cloud database, as said in the case study description.

An external configuration tool is adopted to configure how data collection, data manipulation, and aggregation is performed by the AASs. This tool interacts with the functionalities contained in the Submodel “Configuration” of the relevant AASs. Figure 6.5 shows how the configuration of the Edge Gateway may start cascading configuration processes to the underlying Industrial PCs. This cascading function calls are possible because of the inherent modularity of LB functionalities, realised in turn by the modularity of the structure of the AASs.

Error logs are generated by the Industrial PCs collecting alarms coming from the AASs of the milling machines, whilst machine information can be directly retrieved by the Enterprise Resource Manufacturing (ERP), which

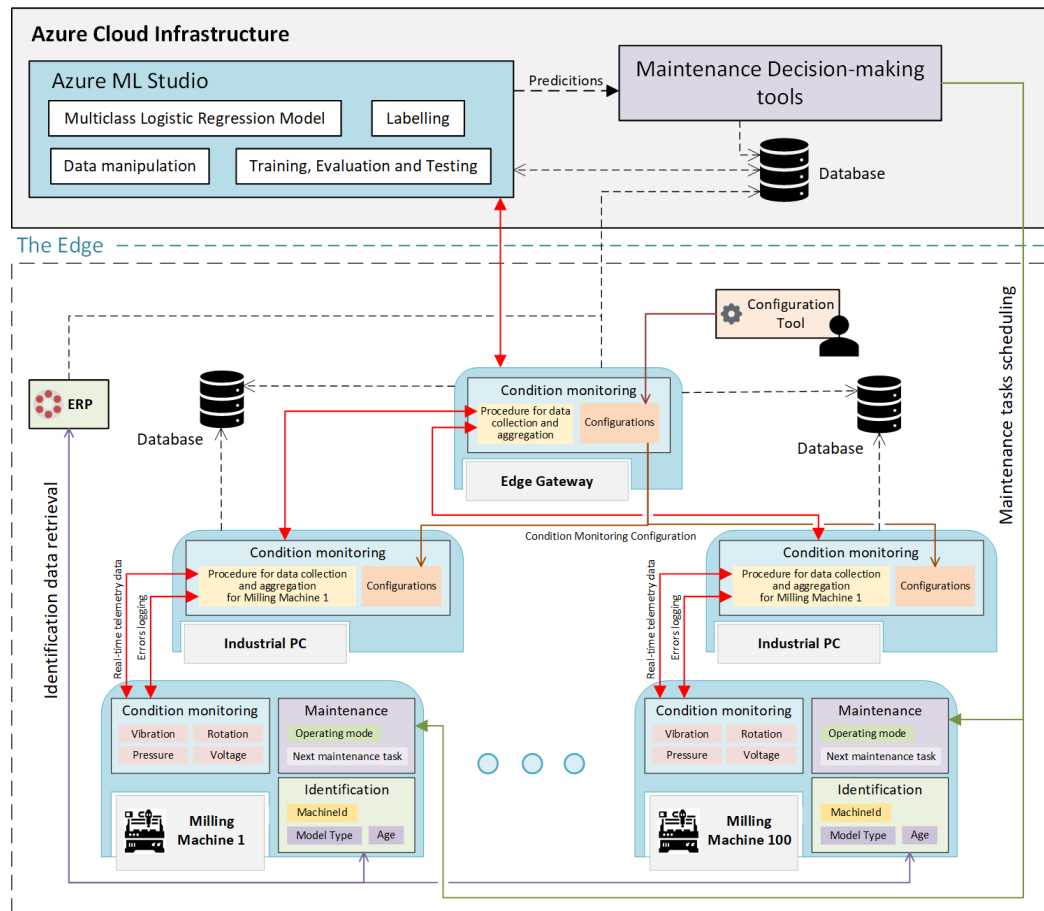


Figure 6.5: Complete overview of the scenario of the case study.

interacts with the AASs of the milling machines and save the information in the cloud. A submodel “Identification” is considered inside the AAS of the milling machines to contains data like identifiers, model type and age. Maintenance history, as already said, is considered available in the cloud database for simplicity.

For the preparation of the dataset and the training, tasting and validation of the ML model, the tools offered by Azure ML Studio are used. The functionalities of the LB Prediction Model are obviously implemented by the tools of Azure ML Studio. Some functionalities of the LB Aggregation are also required at this level to put all data together and create the complete dataset. It is worth noting that with an AAS-based infrastructure under the edge, the

creation of datasets of features is simplified since all the steps required for its construction are abstracted in LB functionalities exposed as uniform and standardised AAS functionalities.

Once the ML-based prediction model is validated, real-time data are provided to the model implementing the functionality of the LB Prediction Model in order to generate predictions. Such predictions are then used by a Maintenance decision-making tool deployed in the cloud that analyses the inputs and eventually schedules maintenance actions. This tool interacts directly with the AASs of the drilling machine to set all the maintenance information in the properties of the Submodel “Maintenance”, which implements the functionalities of the LB Schedule and Status.

The description of this scenario using LBs helps to identify the roles that IT elements must play in order to realise the PdM program. The adoption of AASs is of paramount importance because the implementation details of LBs are hidden by the standardized structure and interface of the AAS. This is a huge improvement in terms of interoperability, especially in the brownfield where lot of different vendor-specific technologies can be used for data retrieval and for communication. The harmonization of the technologies in OT level using the AASs is an evident advantage for the definition (or re-definition) of a PdM solution in the context of Industry 4.0. For instance, considering the case where the wireless vibration sensor of one of the milling machine must be changed with a new one with a different model, this reconfiguration is absolutely transparent from the point of view of the PdM program as long as the data exposed by the AAS of the milling machine are the same. This example can be extended to every component of the PdM program, so that a component can be replaced with a new one without causing disruptive effect to the PdM program as long as it respects the role of the previous one, i.e. it features the same LB functionalities.

6.5 Discussion

The main aim of this research is the harmonization of all possible approaches applied for PdM in order to achieve flexibility and interoperability in the

context of an Industry 4.0 smart factory. We assessed the state of the art about PdM and generalised the steps and the functionalities needed for the definition of a model able to describe every PdM solution in terms of combination of generic functionalities. This is of paramount importance in the context of smart manufacturing because production systems need to adapt their configuration on the basis of the enterprise needs. As a consequence of the re-configuration process, replacing machinery or adding new devices in the production configuration may harm the PdM program.

We identified two main objectives that must be achieved in order to define a model describing PdM program acting on a flexible production plant: PdM functionalities must be technology-independent in order to describe PdM programs regardless its effective implementation, and an abstraction mechanism is required to hide all the heterogeneity of technologies adopted for the PdM implementation.

Our approach is based on the definition of LBs grouping all the different aspects involved in a generic PdM solution, like data acquisition or data manipulation. All the functionalities of an LB abstract a specific operation for the PdM process, regardless how such operation is actually implemented. The advantage in the adoption of LBs is that they allow the definition of roles for the components of the maintenance program, which enable both easy replacement and changings in the PdM solution in a seamless manner. As we discussed in the chapter, since functionalities are generic and technology-independent, the PdM model allows the replacement of a device with another one featuring the same role, even though the implementation of such functionalities differs. This is possible because the LBs follow a “black box” approach where the signatures of the operations are defined in generic terms and, as long as the implementations of the functionalities respect such signatures, devices with the same role are considered equivalent even though they achieve the same goals using different implementations.

Furthermore, the concept of AAS provides an abstraction layer for the heterogeneity of devices and technologies adopted (especially in the brown-field) improving the degree of integration between PdM components and a common structure to the information and operation featured by devices.

Both LBs and AASs specified in the PdM model allow the definition of maintenance programs to improve the level of flexibility in production, thus satisfying the aforementioned main objectives for the definition of a flexible PdM model.

The case study analysed in section 6.4 demonstrated how the the AAS is a powerful Industry 4.0 concept that hides the implementation details behind PdM solutions increasing the interoperability between both devices of different manufacturer and devices using different technologies.

With this research we described a new scenario taking advantage of the AAS as a single universal source of information relevant to assets in order to achieve flexible manufacturing against a PdM maintenance program. Future works can demonstrate how new information generated by procedures of different areas (like PdM) can be used to improve production flexibility. For instance, a manufacturing system may access the AAS of a machine to retrieve its operational status; if the machine is either in maintenance or in failure mode, the manufacturing system can use such information to assign the next production step to a functionally equivalent machine. This may show how the results coming from the PdM solution may be used to improve the availability and the optimisation of the production system. According to the classification of interoperability given in the introduction of this thesis, the approach described in this chapter allows the technical, syntactic and semantic interoperability inside the definition of a PdM solution, but it is not possible speaking of organizational interoperability since the cooperation with processes out of the context of the PdM program is not guaranteed.

6.6 Publications

The results conducted for this research activities has been published in the scientific journal “Sensors” for the special issue “Industrial Internet of Things in the Industry 4.0: New Researches, Applications and Challenges” [103].

Chapter 7

Conclusions

In this thesis, we discussed the importance of interoperability in Industry 4.0 and the importance of the concept of the AAS to achieve it. AAS is defined in RAMI4.0 as the corner stone of interoperability because it is the only component that allows the representation of physical assets in the digital world, exposing them in a uniform, standardised and semantically-annotated structure. In this context, the AAS is the powerful component that “transform” the asset it represents in a CPS (I4.0 Component), providing a uniform information model and a standardised and interoperable communication interface (I4.0 Communication).

Since the very beginning of RAMI4.0, the concept of the AAS has been presented from an high-level point of view providing its generic structure and functional behaviour in a very abstract manner. Only in last few years first documents describing its concrete internal structure (from the informational level point of view) start to appear, and in the very recent future other documents providing API and architectural structure will be released.

Our research started with the study of the existent technologies to achieve interoperability; in particular, the standard IEC 62541 (OPC UA), which is proposed by RAMI4.0 as the only possible technology to implement the Communication Layer of a I4.0 Component, has been identified as a suitable solution in order to implement an AAS.

Our first research activity involved the definition of an AAS using OPC

UA and, in particular, a mapping between the AAS metamodel and the OPC UA standard information model so that the internal structure of the AAS realised with OPC UA still maintains the rules of the original metamodel. We proposed reasoning behind the adoption of the instruments provided by OPC UA to represent an AAS inside the AddressSpace of an OPC UA Server, providing pros and cons behind each solution. We followed a bottom-up approach for the mapping between the two information models, starting with the most fine-grained elements of the AAS metamodel and finishing with more complex mechanisms behind the AAS structure, like referencing and data specification. We realised an OPC UA Information Model for the realisation of AASs and described a use case where it is possible taking advantage of the OPC UA technology to define an Operator Support System accessing the AASs inside OPC UA Servers to configure the production in the context of an assembly line.

The research continued defining two new methodologies taking advantage of the concept of the AAS that highlight how new scenarios can benefit from its introduction. The first methodology we introduced was the representation of PLC programs based on IEC 61131-3 inside the AASs of PLCs. The representation of PLC programs in the digital world permits to underline relationships between software elements defined inside programs and physical parts constituting both assets and production plant. Such detailed description of the state of the plant permits not only an easier management of the plant configuration in its whole life-cycle but also a means to create more flexible production systems. We described use case scenarios where AASs can be accessed by specific tools in order to evaluate configuration parameters and generate new PLC programs according to these new parameters. Furthermore, we discussed how AAS information can be used in the future during the development process of PLC programs creating a closer relationship between the digital world and the physical world.

The second methodology leveraging the AAS consist in the definition of interoperable and flexible PdM programs for production systems. We defined a PdM model that allows the description of a PdM solution using abstract and technology-independent functionalities to address the issue faced by mainte-

nance programs in the context of a re-configurable factory in industry 4.0. The research question was “How can we change the plant configuration (changing parameters or adding or replacing new machinery) without harming the PdM program?”. Usually PdM programs fit the machinery and data they provide; replacing a machine or consider a new health indicator require a complete revision of the maintenance programs and, usually, the complete redefinition of the prediction model used for the assessment of machinery health. We assessed the state of the art on PdM and we realised that two main objectives must be achieved in order to define a flexible PdM program: 1) defining generic functionalities for the description of a technology-independent PdM solution and 2) hiding the heterogeneity and complexity of the OT level. This is in accordance to the lessons we learned from OPC UA and AAS to achieve interoperability since both define abstraction layers hiding implementation details and uniform interfaces to generic functionalities that can be used to describe detailed and domain-specific operations. Therefore, we defined a PdM model providing generic functionalities grouped in so-called LBs, which are modular elements that can be used to describe the role of a PdM component in a technology-independent manner, hence addressing the objective 1). To address 2), instead, our model considers the adoption of AASs for brownfield devices, showing how a PdM program based on the utilisation of I4.0 Component easily achieve the objective 2). It is worth noting how the definition of LBs is suited to being realised inside specific Submodels inside the AAss of machinery, as described in chapter 6. Finally, we presented a real scenario describing a PdM program for the maintenance of 100 industrial milling machine where our PdM model is used for the definition of the PdM solution. Such description abstracts the roles covered by the infrastructure components in the context of the maintenance program and hides the implementation details, so that eventual modifications to the plant do not affect the PdM program as long as replacing components respect the descriptions (according the PdM Model) of the old ones.

We found that most of the features and the philosophy behind OPC UA have lot in common with the concept of the AAS. Both provides elementary and generic building blocks (i.e. metamodel) that can be used in order to re-

alise more complex concepts. Both leverage on semantics to express concepts represented with either same or similar structures, even though OPC UA and AAS use different strategies to accomplish this.

The similarities between OPC UA and AAS highlight that interoperability requires the definition of a universal and generic metamodel to express more complex and domain-specific concepts, but this is just a necessary but non-sufficient condition. What is of paramount importance is the possibility to express a semantics over concepts defined using metamodels. OPC UA uses the so-called Information model which defined standardised objects and manners to compose elements of specific domains, whereas AAS uses external dictionaries providing standard concept definition to semantically annotate elements.

What comes out from this discussion is that, even though interoperability requires abstraction layer and technology independent interfaces to create data federation between information systems (**syntactic interoperability**) and both semantic annotations and relationships to enable knowledge inference and machine computable logic (**semantic interoperability**), the foundation of interoperability is standardisation. Elementary concepts, interface operations and semantics must be standardised in order to start the description of more complex systems that must interoperate and companies must agree with such concepts to start creating an Industry 4.0 approach.

Established this common shared foundation, technologies and approaches from the field of Internet of Things act a major role to maintain data updated in real-time and for the exchange of information. Telemetry protocols and embedded systems are of paramount importance for gathering and exchanging data from assets. In particular, future works may rely in Time-Sensitive Networking to create stronger relationships between assets and their digital twins, providing a reliable means to fill in values for properties and to guarantee real-time executions of the operations defined inside the AASs. In this sense, the OPC UA-based AAS discussed in this thesis can take advantage of the new OPC UA PubSub specification [104] and, in general, the conjunction between AAS and OPC UA with TSN [105] will guarantee interoperability between different control subsystems, improving the management and analysis of field

devices satisfying real-time communication constraints.

Bibliography

- [1] Yongxin Liao et al. “Past, present and future of Industry 4.0 - a systematic literature review and research agenda proposal”. In: *International Journal of Production Research* 55 (2017), pp. 3609–3629.
- [2] L. Xu, Eric Xu, and L. Li. “Industry 4.0: state of the art and future trends”. In: *International Journal of Production Research* 56 (2018), pp. 2941–2962.
- [3] Ateeq Khan. and Klaus Turowski. “A Perspective on Industry 4.0: From Challenges to Opportunities in Production Systems”. In: *Proceedings of the International Conference on Internet of Things and Big Data - Volume 1: IoTBD*, INSTICC. SciTePress, 2016, pp. 441–448. ISBN: 978-989-758-183-0. DOI: 10.5220/0005929704410448.
- [4] Henning Kagermann, Wolfgang Wahlster, and Johannes Helbig. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0 – Securing the Future of German Manufacturing Industry*. Final Report of the Industrie 4.0 Working Group. acatech – National Academy of Science and Engineering, 2013.
- [5] Stephan Weyer et al. “Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems”. In: *IFAC-PapersOnLine* 48.3 (2015), pp. 579–584. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2015.06.143>.
- [6] Hugh Boyes et al. “The industrial internet of things (IIoT): An analysis framework”. In: *Computers in Industry* 101 (2018), pp. 1–12. ISSN: 0166-3615. DOI: <https://doi.org/10.1016/j.compind.2018.04.015>.

- [7] D. Schulte and A. Colombo. “RAMI 4.0 based digitalization of an industrial plate extruder system: Technical and infrastructural challenges”. In: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society* (2017), pp. 3506–3511.
- [8] A. Geraci et al. “IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries”. In: 1991.
- [9] Jacob Nilsson and F. Sandin. “Semantic Interoperability in Industry 4.0: Survey of Recent Developments and Outlook”. In: *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)* (2018), pp. 127–132.
- [10] R. Rezaei, T. K. Chiew, and S. Lee. “A review of interoperability assessment models”. In: *Journal of Zhejiang University SCIENCE C* 14 (2013), pp. 663–681.
- [11] Didem Gürdür and Fredrik Asplund. “A Systematic Review to Merge Discourses : Interoperability, Integration and Cyber-Physical Systems”. In: *Journal of Industrial Information Integration* 9 (2017), pp. 14–23.
- [12] *IEC TR 62390:2005 – Common automation device - Profile guideline*. IEC.
- [13] Jonathan Fuchs et al. “I4.0-compliant integration of assets utilizing the Asset Administration Shell”. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (2019), pp. 1243–1247.
- [14] *DIN SPEC 91345:2016-04, Reference Architecture Model Industrie 4.0 (RAMI4.0)*. Berlin: DIN, Mar. 2016.
- [15] Platform Industrie 4.0 and ZVEI. *Details of the Asset Administration Shell - Part 1 - the exchange of information between partners in the value chain of Industrie 4.0*. 2020.
- [16] M. A. Iñigo et al. “Towards an Asset Administration Shell scenario: a use case for interoperability and standardization in Industry 4.0”. In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium* (2020), pp. 1–6.

- [17] Platform Industrie 4.0 and ZVEI. *Relationships between I4.0 Components - Composite Components and Smart Production*. 2017.
- [18] Platform Industrie 4.0. *Structure of the Administration Shell - Continuation of the Development of the Reference Model for the Industrie 4.0 Component*. 2016.
- [19] David Kampert and U. Epple. “Modeling asset information for interoperable software systems”. In: *IEEE 10th International Conference on Industrial Informatics* (2012), pp. 947–952.
- [20] ZVEI. *Examples of the Asset Administration Shell for Industrie 4.0 Components - Basic parts*. 2017.
- [21] Dorota Lang et al. “Utilization of the Asset Administration Shell to Support Humans During the Maintenance Process”. In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN) 1* (2019), pp. 768–773.
- [22] F. Prinz et al. “Configuration of Application Layer Protocols within Real-time I4.0 Components”. In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN) 1* (2019), pp. 971–976.
- [23] Alejandro Seif, Carlos Toro, and Humza Akhtar. “Implementing Industry 4.0 Asset Administrative Shells in Mini Factories”. In: *KES*. 2019.
- [24] *DIN SPEC 16593-1:2018-04 RM-SA, Reference Model for Industrie 4.0 Service Architectures - Part 1: Basic Concepts of an Interaction-based Architecture*. Berlin: DIN, 2018.
- [25] S. Panda et al. “Plug&Produce Integration of Components into OPC UA based data-space”. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA) 1* (2018), pp. 1095–1100.
- [26] Platform Industrie 4.0. *I4.0-Sprache - Vokabular, Nachrichtenstruktur und semantische Interaktionsprotokolle der I4.0-Sprache*. 2018.

- [27] C. Diedrich et al. “Semantic interoperability for asset communication within smart factories”. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (2017), pp. 1–8.
- [28] A. Belyaev and C. Diedrich. *Specification Testbed ”AAS networked” - Proactive AAS - interaction according to the VDI/VDE 2193*. July 2020. URL: http://www.lia.ovgu.de/lia_media/LIA_Medien/AASnetworked-p-250.pdf.
- [29] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC Unified Architecture*. Springer, 2009.
- [30] Wenbin Dai et al. “Modelling Industrial Cyber-Physical Systems using IEC 61499 and OPC UA”. In: *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)* (2018), pp. 772–777.
- [31] *Specification Amendment 7: Interfaces and AddIns, release 1.04*. OPC Foundation, 2019.
- [32] *OPC UA Part 6: Mappings, release 1.04*. OPC Foundation, 2017.
- [33] José M. Gutiérrez-Guerrero and J. A. Holgado-Terriza. “Automatic Configuration of OPC UA for Industrial Internet of Things Environments”. In: *Electronics* 8 (2019), p. 600.
- [34] P. Ferrari et al. “Impact of Quality of Service on Cloud Based Industrial IoT Applications with OPC UA”. In: *Electronics* 7 (2018), p. 109.
- [35] Isaías González et al. “A Literature Survey on Open Platform Communications (OPC) Applied to Advanced Industrial Environments”. In: *Electronics* 8 (2019), p. 510.
- [36] R. Fielding. “Architectural Styles and the Design of Network-based Software Architectures”; Doctoral dissertation”. In: 2000.
- [37] S. Grüner, J. Pfrommer, and F. Palm. “RESTful Industrial Communication With OPC UA”. In: *IEEE Transactions on Industrial Informatics* 12 (2016), pp. 1832–1841.

- [38] T. Luckenbach et al. “TinyREST – a Protocol for Integrating Sensor Networks into the Internet”. In: 2005.
- [39] D. Guinard, V. Trifa, and E. Wilde. “A resource oriented architecture for the Web of Things”. In: *2010 Internet of Things (IOT)* (2010), pp. 1–8.
- [40] R. Pribiš, L. Beňo, and P. Drahoš. “An Industrial Communication Platform for Industry 4.0 - case study”. In: *2020 Cybernetics & Informatics (K&I)* (2020), pp. 1–9.
- [41] Ovidiu Vermesan et al. *Strategy and coordination plan for IoT interoperability and standard approaches*. Tech. rep. ETSI, 2017.
- [42] S. Cavalieri et al. “A web-based platform for OPC UA integration in IIoT environment”. In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA* (2017), pp. 1–6. DOI: 10.1109/ETFA.2017.8247713.
- [43] S. Cavalieri et al. “Integration of OPC UA into a web-based platform to enhance interoperability”. In: *IEEE International Symposium on Industrial Electronics* (2017), pp. 1206–1211. DOI: 10.1109/ISIE.2017.8001417.
- [44] S. Cavalieri et al. “OPC UA integration into the web”. In: *Proceedings IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society 2017-January* (2017), pp. 3486–3491. DOI: 10.1109/IECON.2017.8216590.
- [45] S. Cavalieri, M.G. Salafia, and M.S. Scroppo. “Integrating OPC UA with web technologies to enhance interoperability”. In: *Computer Standards and Interfaces* 61 (2019), pp. 45–64. DOI: 10.1016/j.csi.2018.04.004.
- [46] S. Cavalieri, M.G. Salafia, and M.S. Scroppo. “Mapping OPC UA AddressSpace to OCF resource model”. In: *Proceedings - 2018 IEEE Industrial Cyber-Physical Systems, ICPS 2018* (2018), pp. 135–140. DOI: 10.1109/ICPHYS.2018.8387649.

- [47] S. Cavalieri et al. “Towards integration between OPC UA and OCF”. In: *ICEIS 2019 - Proceedings of the 21st International Conference on Enterprise Information Systems 1* (2019), pp. 543–550. DOI: 10.5220/0007672205550562.
- [48] S. Cavalieri, M.G. Salafia, and M.S. Scroppo. “Realising Interoperability between OPC UA and OCF”. In: *IEEE Access* 6 (2018), pp. 69342–69357. DOI: 10.1109/ACCESS.2018.2880040.
- [49] S. Cavalieri, M.G. Salafia, and M.S. Scroppo. “Towards interoperability between OPC UA and OCF”. In: *Journal of Industrial Information Integration* 15 (2019), pp. 122–137. DOI: 10.1016/j.jii.2019.01.002.
- [50] A. Lüder et al. “One step towards an industry 4.0 component”. In: *2017 13th IEEE Conference on Automation Science and Engineering (CASE)* (2017), pp. 1268–1273.
- [51] Xun Ye and S. Hong. “Toward Industry 4.0 Components: Insights Into and Implementation of Asset Administration Shells”. In: *IEEE Industrial Electronics Magazine* 13 (2019), pp. 13–25.
- [52] Irlán Grangel-González et al. “Towards a Semantic Administrative Shell for Industry 4.0 Components”. In: *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)* (2016), pp. 230–237.
- [53] *Specification Amendment 5: Dictionary Reference, release 1.04*. OPC Foundation, 2019.
- [54] Y. Cohen et al. “Assembly system configuration through Industry 4.0 principles: the expected change in the actual paradigms”. In: *IFAC-PapersOnLine* 50 (2017), pp. 14958–14963.
- [55] M. A. Iñigo et al. “Towards an Asset Administration Shell scenario: a use case for interoperability and standardization in Industry 4.0”. In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium* (2020), pp. 1–6.

- [56] S. Cavaliere, S. Mule, and M.G. Salafia. “OPC UA-based Asset Administration Shell”. In: *IECON Proceedings (Industrial Electronics Conference) 2019-October* (2019), pp. 2982–2989. DOI: 10.1109/IECON.2019.8926859.
- [57] S. Cavaliere and M.G. Salafia. “Insights into mapping solutions based on OPC UA information model applied to the industry 4.0 asset administration shell”. In: *Computers* 9.2 (2020). DOI: 10.3390/computers9020028.
- [58] R. Langmann and Leandro F. Rojas-Peña. “A PLC as an Industry 4.0 component”. In: *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)* (2016), pp. 10–15.
- [59] *IEC 61131-3:2013 - Part 3: Programming languages*. International Electrotechnical Commission (IEC), 2013.
- [60] Langmann and Stiller. “The PLC as a Smart Service in Industry 4.0 Production Systems”. In: *Applied Sciences* 9 (2019), p. 3815.
- [61] M. Wenger, Alois Zoitl, and T. Müller. “Connecting PLCs With Their Asset Administration Shell For Automatic Device Configuration”. In: *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)* (2018), pp. 74–79.
- [62] Hyeong-Tae Park et al. “Plant model generation for PLC simulation”. In: *International Journal of Production Research* 48 (2010), pp. 1517–1529.
- [63] S. Park et al. “PLCStudio: Simulation based PLC code verification”. In: *2008 Winter Simulation Conference* (2008), pp. 222–228.
- [64] S. Park, Minsuk Ko, and Minho Chang. “A reverse engineering approach to generate a virtual plant model for PLC simulation”. In: *The International Journal of Advanced Manufacturing Technology* 69 (2013), pp. 2459–2469.
- [65] J. Machado et al. “LOGIC CONTROLLERS DEPENDABILITY VERIFICATION USING A PLANT MODEL”. In: *IFAC Proceedings Volumes* 39 (2006), pp. 37–42.

- [66] S. Park, C. M. Park, and Gi-Nam Wang. “A PLC programming environment based on a virtual plant”. In: *The International Journal of Advanced Manufacturing Technology* 39 (2008), pp. 1262–1270.
- [67] Yuqi Chen, Christopher M. Poskitt, and J. Sun. “Learning from Mutants: Using Code Mutation to Learn and Monitor Invariants of a Cyber-Physical System”. In: *2018 IEEE Symposium on Security and Privacy (SP)* (2018), pp. 648–660.
- [68] R. W. Lewis. *Programming industrial control systems using IEC 61131-3*. The Institution of Electrical Engineers, 1998.
- [69] S. G. McCrady. *Designing SCADA Application Software*. Elsevier, 2013.
- [70] *IEC 61131-10:2019 - Part 10: PLC open XML exchange format*. International Electrotechnical Commission (IEC), 2019.
- [71] *OPC 30001: IEC61131-3 Client Function Blocks for OPC UA*. OPC Foundation, PLCopen, 2016.
- [72] S. Cavalieri and M.G. Salafia. “Asset Administration Shell for PLC Representation Based on IEC 61131-3”. In: *IEEE Access* 8 (2020), pp. 142606–142621. DOI: 10.1109/ACCESS.2020.3013890.
- [73] Mobley R. K. *An introduction to Predictive Maintenance*. Elsevier, 2002.
- [74] O. Motaghare, A. S. Pillai, and K. I. Ramachandran. “Predictive Maintenance Architecture”. In: *2018 IEEE International Conference on Computational Intelligence and Computing Research (ICCCIC)*. 2018, pp. 1–4. DOI: 10.1109/ICCCIC.2018.8782406.
- [75] Christin Groba et al. “Architecture of a Predictive Maintenance Framework”. In: *6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM’07)* (2007), pp. 59–64.
- [76] E. Traini et al. “Machine learning framework for predictive maintenance in milling”. In: *IFAC-PapersOnLine* 52 (2019), pp. 177–182.

- [77] Go Muan Sang et al. “Towards Predictive Maintenance for Flexible Manufacturing Using FIWARE”. In: *Advanced Information Systems Engineering Workshops* 382 (2020), pp. 17–28.
- [78] Antonio J. Guillén et al. “A framework for effective management of condition based maintenance programs in the context of industrial development of E-Maintenance strategies”. In: *Comput. Ind.* 82 (2016), pp. 170–185.
- [79] Platform Industrie 4.0. *The Standardisation Roadmap of Predictive Maintenance for Sino-German Industrie 4.0/Intelligent Manufacturing*. 2018.
- [80] *VDMA 24582:2014 – Fieldbus Neutral Reference Architecture for Condition Monitoring in Production Automation*. VDMA, 2014.
- [81] Erdal Tantik and R. Anderl. “Potentials of the Asset Administration Shell of Industrie 4.0 for Service-Oriented Business Models”. In: *Procedia CIRP* 64 (2017), pp. 363–368.
- [82] Max Birtel et al. “Requirements for a Human-Centered Condition Monitoring in Modular Production Environments”. In: *IFAC-PapersOnLine* 51 (2018), pp. 909–914.
- [83] Ahmad Rosmaini and K. Shahrul. “An overview of time-based and condition-based maintenance in industrial application”. In: *Computers & Industrial Engineering* (2012).
- [84] H. Hashemian and W. C. Bean. “State-of-the-Art Predictive Maintenance Techniques*”. In: *IEEE Transactions on Instrumentation and Measurement* 60 (2011), pp. 3480–3492.
- [85] S. Patil and J. Gaikwad. “Vibration analysis of electrical rotating machines using FFT: A method of predictive maintenance”. In: *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)* (2013), pp. 1–6.

- [86] A. K. S. Jardine, D. Lin, and D. Banjevic. “A review on machinery diagnostics and prognostics implementing condition-based maintenance”. In: *Mechanical Systems and Signal Processing* 20 (2006), pp. 1483–1510.
- [87] H. Hashemian. “Response time testing of temperature sensors using loop current step response method”. In: *International Journal of Nuclear Energy Science and Technology* 7 (2013), p. 209.
- [88] Patrick Straus et al. “Enabling of Predictive Maintenance in the Brown-field through Low-Cost Sensors, an IIoT-Architecture and Machine Learning”. In: *2018 IEEE International Conference on Big Data (Big Data)* (2018), pp. 1474–1483.
- [89] P. Poór, J. Basl, and D. Ženíšek. “Predictive Maintenance 4.0 as next evolution step in industrial maintenance development”. In: *2019 International Research Conference on Smart Computing and Systems Engineering (SCSE)* (2019), pp. 245–253.
- [90] S. March and Gary D. Scudder. “Predictive maintenance: strategic use of IT in manufacturing organizations”. In: *Information Systems Frontiers* 21 (2019), pp. 327–341.
- [91] Jun Ma and J. Li. “Detection of localised defects in rolling element bearings via composite hypothesis test”. In: *Mechanical Systems and Signal Processing* 9 (1995), pp. 63–75.
- [92] M. Fugate, H. Sohn, and C. Farrar. “VIBRATION-BASED DAMAGE DETECTION USING STATISTICAL PROCESS CONTROL”. In: *Mechanical Systems and Signal Processing* 15 (2001), pp. 707–721.
- [93] V. Skormin et al. “Applications of cluster analysis in diagnostics-related problems”. In: *1999 IEEE Aerospace Conference. Proceedings (Cat. No.99TH8403)* 3 (1999), 161–168 vol.3.
- [94] S. Orlov, R. V. Girin, and O. Y. Uyutova. “Artificial Neural Network for Technical Diagnostics of Control Systems by Thermography”. In: *2018 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)* (2018), pp. 1–4.

- [95] R. M. Tallam, T. Habetler, and R. Harley. “Self-commissioning training algorithms for neural networks with applications to electric machine fault diagnostics”. In: *IEEE Transactions on Power Electronics* 17 (2002), pp. 1089–1095.
- [96] Suresh K. Sampath et al. “Engine-fault diagnostics:an optimisation procedure”. In: *Applied Energy* 73 (2002), pp. 47–70.
- [97] W. Bartelmus. “Diagnostic information on gearbox condition for mechatronic systems”. In: *Transactions of the Institute of Measurement & Control* 25 (2003), pp. 451–465.
- [98] Yuhuang Zheng. “Predicting Remaining Useful Life Based on Hilbert-Huang Entropy with Degradation Model”. In: *J. Electr. Comput. Eng.* 2019 (2019), 3203959:1–3203959:11.
- [99] Gian Antonio Susto et al. “Machine Learning for Predictive Maintenance: A Multiple Classifier Approach”. In: *IEEE Transactions on Industrial Informatics* 11 (2015), pp. 812–820.
- [100] M. Paolanti et al. “Machine Learning approach for Predictive Maintenance in Industry 4.0”. In: *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)* (2018), pp. 1–6.
- [101] O. Koca, O. T. Kaymakci, and M. Mercimek. “Advanced Predictive Maintenance with Machine Learning Failure Estimation in Industrial Packaging Robots”. In: *2020 International Conference on Development and Application Systems (DAS)* (2020), pp. 1–6.
- [102] *ISO 17359:2018 – Condition monitoring and diagnostics of machines — General guidelines*. ISO.
- [103] S. Cavalieri and M.G. Salafia. “A model for predictive maintenance based on asset administration shell”. In: *Sensors (Switzerland)* 20.21 (2020), pp. 1–20. DOI: 10.3390/s20216028.
- [104] *OPC UA Part 14: PubSub*. OPC Foundation, 2018.

- [105] D. Bruckner et al. “An Introduction to OPC UA TSN for Industrial Communication Systems”. In: *Proceedings of the IEEE* 107 (2019), pp. 1121–1131.

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI 2014IT16M2OP005),
Fondo Sociale Europeo, Azione I.1 "Dottorati Innovativi con caratterizzazione Industriale"



UNIONE EUROPEA
Fondo Sociale Europeo



*Ministero dell'Università
e della Ricerca*

