Received July 11, 2020, accepted July 29, 2020, date of publication August 3, 2020, date of current version August 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3013890

Asset Administration Shell for PLC Representation Based on IEC 61131-3

SALVATORE CAVALIERI AND MARCO GIUSEPPE SALAFIA
Department of Electrical Electronic and Computer Engineering, University of Catania, 95125 Catania, Italy

Corresponding author: Salvatore Cavalieri (salvatore.cavalieri@unict.it)

This work was supported in part by the "Piano di incentivi per la ricerca di Ateneo 2020/2022 (PIA.CE.RI.)," University of Catania.

ABSTRACT In the Reference Architecture Model for Industrie 4.0, the concept of Asset Administration Shell is presented as the corner stone of interoperability. Asset Administration Shell is defined as a digital representation of an asset able to provide information about the asset including documents, properties, parameters, and functionalities, all organized in a consistent way. Information provided by an Asset Administration Shell can be adopted during whole life cycle of a production system, from its development until its disposal. At the lowest level of the hierarchy of a production system, usually automation and control programs are executed by Programmable Logic Controllers, whose programming technology is based on IEC 61131-3 standard. The IEC 61131-3 programs, the Programmable Logic Controllers where they run, and the real plant controlled are closely related. Considering the life cycle of a production system, the description of IEC 61131-3 programs and the relevant relationships with the plant should be clearly defined, leading to several advantages for example regarding the definition of testing plant operations, maintenance operations at run-time and reconfiguration process of the plant. What is missing for the realization of what said so far is a standard way to realize this description. For this reason, the paper presents an Asset Administration Shell model able to represent IEC 61131-3 programs and the relevant relationships with Programmable Logic Controllers and each device of the controlled plant.

INDEX TERMS Industry 4.0, Asset Administration Shell, IEC 61131-3, PLC.

I. INTRODUCTION

Standards are the pillars of interoperability in the context of Industry 4.0 because their adoption creates the basis for the interworking between partners of a value-chain network. With the progressing technological development and quick market changes, adaptability is required to achieve agility in the production systems in factories [1]. Consequently, current manufacturing and production systems demand short re-configuration time and rapid changeover [2]. Different tools in different domain areas are adopted during the development of a production systems and most of the time it is needed for these tools to cooperate. For instance, just in automotive industry more than 30 different engineering activities are involved within the process of engineering a body work production system [3]. Standards for communication and information exchange is required in this context to achieve interoperability [4] because same data can be used in different domain for different purposes.

The associate editor coordinating the review of this manuscript and approving it for publication was Remigiusz Wisniewski.

In the Reference Architecture Model for Industrie 4.0 (RAMI 4.0) [5], the concept of Asset Administration Shell (AAS) is presented as the corner stone of interoperability. AAS is defined as a digital representation of a relevant asset; it provides an interface to an Industry 4.0 network allowing the communication with other assets and the exchange of information. Assets are defined as entities owned by an organization having either a perceived or actual value for the organization itself; physical entities like machines, products or controllers are considered assets, but even software, documents or licenses can be considered assets too.

In RAMI 4.0, an AAS is given to every asset, and the composition of the AAS and the asset is referred in RAMI 4.0 as an Industry 4.0 Component (I4.0 Component). AAS consists of several submodels within which information and functionalities of a given asset are described. In last years, an AAS metamodel has been released providing the basis for the development and usage of new Industry 4.0-compliant products and tools [6].

The objective of AAS is to provide information about the asset; all the relevant information including documents, properties, parameters, and functionalities are organized in a consistent way [7]. AAS is the information carrier of an asset, therefore information provided by AAS can be adopted during whole life cycle of a production system, from its development until its disposal. In a production system every device or a whole automation island can be represented by an AAS whose information can be used, for instance, by engineers during the production system development or from operators during maintenance phase.

At the lowest level of the hierarchy of a production system, usually automation and control programs are executed by Programmable Logic Controllers (PLCs). Due to the massive adoption of PLCs nowadays, it seems legit thinking that they will be used in the age of Industry 4.0, as stated by [8]. The PLC programming technology adopted nowadays is based on IEC 61131-3 [9], and it is legit imagine I4.0 controllers based on this standard [10].

There is a close relationship between IEC 61131-3 programs, the hardware and software resources of the PLC where they run and the plant (which may include a set of controlled machines, control devices, control applications and communication systems). For example, a variable of an IEC 61131-3 program may be mapped to a real input or output of the PLC and then to a real device (e.g. sensors, actuators) connected to a specific PLC input/output. 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. A comprehensive description of IEC 61131-3 programs including the relevant mappings with the PLC where they run, and the controlled plant could be very useful. Considering the life cycle of a production system, this description could help the definition of testing operations before the utilization of the plant (after its realization) and the maintenance operations at run-time. Changing the production system or the product to produce often requires adjustments in the plant configuration [11]; the reconfiguration process may be easily conducted if a fully and standard description of the entire system involving both the PLC programs and the relationships with the real plant could be available.

What is needed is a standard way to realize this description. Considering the AAS, what is missing for the realization of what said so far is a model able to describe PLC programs-based on IEC 61131-3 and the relevant relationships with the PLC hardware and the devices of the controlled plant.

In this paper, authors propose some reasonings aimed to define an approach to represent the entire set of PLC, its internal programs, and the relationships with the real plant by means of AAS. An AAS model will be proposed in the paper, and it will be defined following the metamodel introduced in [6].

The paper is structured as follows. In Section II, the current state of the art about the subject of the paper is described. In Section III, main concepts of IEC 61131-3 are provided as background for the paper comprehension. In Section IV, the main parts of the AAS metamodel are discussed. Section V contains the definition of the model here proposed, describing how the AAS metamodel entities are used for the representation of all the main elements of IEC 61131-3. In Section VI, an approach to represent both the PLC programs based on IEC 61131-3 and the relevant relationships with the real plant will be presented. Section VII shows a case study to give an example of the approach presented in Sections V and VI. Section VIII details the implementation made the authors. Finally, Section IX will provide conclusions.

II. STATE OF THE ART

Current literature presents many research papers dealing with integrated models including IEC 61131-3 programs and the relevant plant controlled. The aim of this section is to provide an overview of the state-of-the-art pointing out the main activities that would benefit from a standardized modeling of PLC programs and plants.

Since the implementation of a manufacturing line requires heavy investment, proper verification of a line's operational status should be performed before the implementation to ensure that the highly automated manufacturing system will successfully achieve the intended benefits. Many existing approaches utilize simulation techniques for the verification. Since PLC programs only contain the control information without device models, these approaches require a corresponding plant model to perform simulation [12], [13]. In [14] a detailed overview is given about the subject to construct a PLC simulation environment, pointing out that it is necessary to build 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 approached are based on the use of a plant model integrated with the PLC program to be verified [15]. In [16] another approach for verification of PLC programs is proposed; it is based on the visual verification of PLC programs that integrates a PLC program with a corresponding plant model, so that users can intuitively verify the PLC program in a 3D graphic environment.

Another time- and resource-consuming process exists before a real plant can start its activity; it is relevant to the commissioning. In order to save time and resources, many times virtual commissioning is considered. While the real commissioning of a manufacturing system involves a real plant system and a real controller, the virtual commissioning deals with a virtual plant model and a real controller. The expected benefits of virtual commissioning are the reduction of debugging and correction efforts during the subsequent real commissioning stage. However, this approach requires a virtual plant model integrated with the PLC program. The research paper [17] gives a detailed survey on application of virtual commissioning technology for automated manufacturing systems.

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) [18].

A last consideration needed to summarize this overview is that at this moment any standard representation of the integrated model made up of PLC program and plant controlled does not exist. Each of the approaches given in this overview represents the PLC program and the plant using proprietary formalisms. For this reason, the authors would like to highlight the importance of the proposal here presented, based on the use of AAS metamodel of Industry 4.0. The proposal can give a standard representation of PLC program and the relevant plant controlled; this representation can be used in each of the above-mentioned applications, improving interoperability of the same applications. Proposal is original, as solutions aimed to propose consistent ways to structure information relevant to PLC inside AAS seem missing in the current literature.

III. IEC 61131-3

The use of PLCs for industrial control application is consolidated after the release of the standard IEC 61131-3. This last provides an architectural definition and a software model for industrial PLCs. In particular, the IEC 61131-3 cope with the problem of having different vendor-specific languages for PLCs programming [19]. 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).

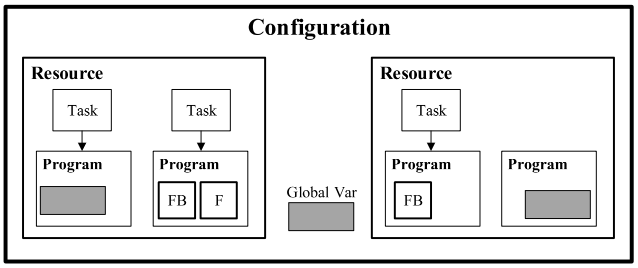

FIGURE 1. Software model of IEC 61131-3.

The foundation of IEC 61131-3 is the definition of a unified software model for the PLC, depicted in Fig. 1, which provides the basic high-level language elements that are programmed using the aforementioned programming languages.

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

Configuration defines the entire software project and must include at least one Resource (described in the following). A Configuration may refer to one or more PLCs involved in the project.

Resource represents a processing facility of the PLC that can execute a program. It is defined inside a Configuration and it allows the definition of several information about the PLC, among which 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 is the software element 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.

In IEC 61131-3 control programs are decomposed in functional elements referred as Program Organisation Units (POUs). A POU may be a Program, a Function Block (FB) or a Function (F). All of them may defined using one of the programming languages of IEC 61131-3. A Program typically consists of interconnected Function Blocks exchanging data. A program can communicate with other programs. The execution of different parts of a program may be controlled using Tasks, as explained before. Function Block is another kind of POU, and it 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. The last kind of POU is Function, that is a reusable software element that generates the same output values when the same values are provided as input. It differentiates from FB because it has no internal state whilst FB retain their internal values from latest executing.

Variables are a very important element for IEC 61131-3 programs. They 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 featured by several attributes among which there is the AT [9]; it allows the association of a variable to particular memory address. It is important to recall that according to IEC 61131-3 standard, internal memory of PLC is made up by the Input (I), Output (Q) process images and Marker (M) memories. The I and Q process images are those updated at each Program Scan [9]; inputs are sampled and copied in memory I, whilst data is stored in memory Q in order to update the actual values of outputs at the end of each Program Scan.

All the software elements composing the model in IEC 61131-3 can be related as depicted in Fig. 2; the UML representation highlights all the relationships between all the software elements showing the cardinality of every association. This diagram will be used as a starting point for the definition of a Submodel for an AAS in Section V.

IV. ASSET ADMINISTRATION SHELL METAMODEL

In the context of Industry 4.0, every asset is managed by an AAS containing all its relevant information; such information includes properties, operations, but also documentation, datasheets, CAD files or source code. This is what makes AAS so appealing for a production system life cycle management, since all this information is structured under just

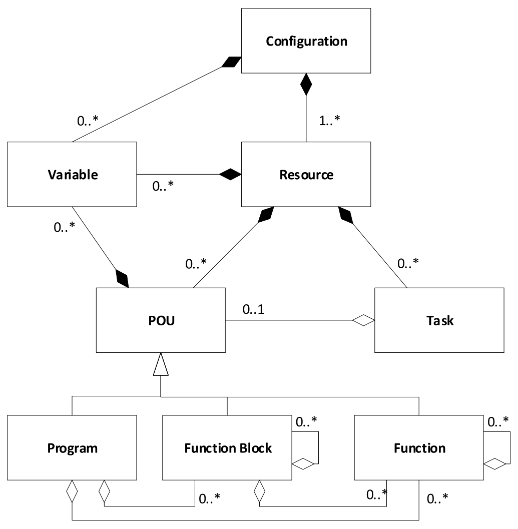

FIGURE 2. Relationships between IEC 61131-3 software model's entities represented in UML.

one entity, i.e. the AAS, and accessible by different domain-specific tools. What is needed in this scenario is a unique and consistent manner to structure the information inside the AAS; the document "Details of the Asset Administration Shell" [6] provides an AAS metamodel to meet this need. Following the metamodel, the AAS information model can be implemented using different formats, among others XML, JSON, AutomationML, RDF and OPC UA. In [6], a package format and different serialization formats for the exchange of AASs between partners are provided.

From a high-level point of view, an AAS consists of a header and a body, as depicted in Fig. 3.

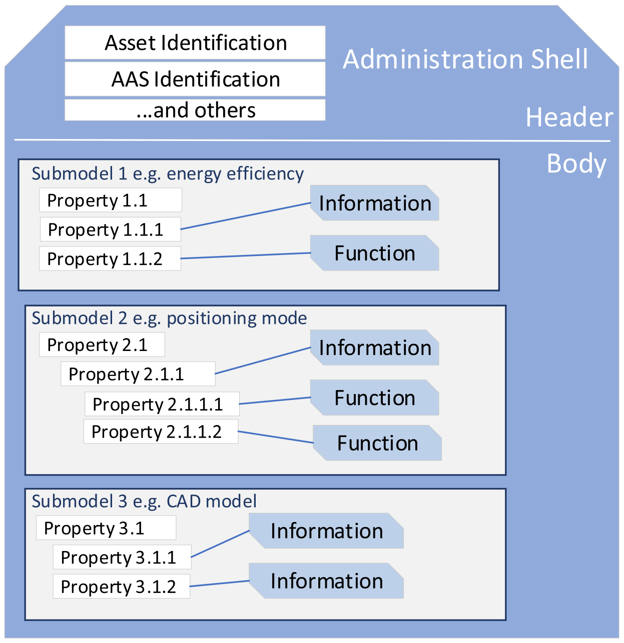

FIGURE 3. Internal structure of an AAS from a high-level point of view.

The header contains all the information relevant to the identification of both asset and AAS, whilst the body contains all the properties and operations describing the asset and organized under submodels covering specific aspects of the asset (e.g. energy efficiency, positioning, etc.). Properties shall be defined in an unambiguous manner, using hierarchically structured dictionaries, and following the standard IEC 61360 for their definition [20].

The AAS metamodel is provided as a UML class diagram describing all the main entities that shall be adopted to structure the information inside an AAS. The main classes and concepts of the metamodel will be briefly discussed in the remainder of this section. In the following, the terms class and entity are used interchangeably.

A. AAS METAMODEL COMMON CLASSES

At the base of the metamodel, some abstract classes are defined to identify aspects common to the other classes of the metamodel; these last will be referred in the following with the name common classes. From a practical point of view, common classes collect attributes that can be shared by different classes in the metamodel.

Some of the main common classes defined in the AAS metamodel are Referable, Identifiable, and HasSemantics, but more are specified in [6].

All entities in the metamodel inheriting from Referable provide a short identifier (*idShort*) that is unique only in the context of its name space. The name space for a Referable element is defined as its parent element. Another attribute inherited from referable is *category* which provides further metadata information about the class of the element.

Identifiable entities, instead, consists of all those classes whose instances can be uniquely and globally identified by means of its attribute *identification.* In other words, Identifiable entities are characterized by absolute identifiers, whilst Referable entities are characterized by relative identifiers. These identifiers will be used inside instances of the class Reference (or Reference, for brevity), as will be discussed in the next subsection.

The entities inheriting from HasSemantics identify all those classes that can be described by means of a concept. The class HasSemantics defines an attribute *semanticId* that is a Reference pointing to the semantics description. Reference will be clarified in the following.

B. REFERENCING MECHANISM OF THE METAMODEL

The AAS metamodel defines a very important referencing mechanism to establish relationships between entities composing the AAS. The foundation of the referencing mechanism consists of the class Reference. It features an attribute *key*, which is logically structured as an ordered list of keys where each key points to an entity by its identifier. The structure of this list of keys resembles an URI structure, where the first key refers to the root element and every following key spans a hierarchy until the referred element, which is identified by the last key of the list. A Reference can also be used to point to an element outside the AAS, e.g. an entry of an external dictionary entry or an element of another AAS. More details on Reference in [6]. In the remainder of the paper, the nomenclature &(<elem>) represents a Reference instance pointing to <elem>.

C. CLASS HIERARCHY IN THE AAS

All the classes defined in the AAS metamodel are used to decompose and simplify the representation of the internal information of an AAS because of [21]. The class hierarchy respects this structure, therefore the topmost class AssetAdministrationShell represent an AAS as a whole, the class Submodel represents an aspect of the asset, and the abstract class SubmodelElement represents all those elements that must be collected under a Submodel (e.g. properties, operations, files, etc.). The AAS metamodel defines several classes but only the ones adopted in the authors' proposal will be briefly described here to ease the comprehension of the paper; a full description of the metamodel is provided in [6]. Fig. 4 shows the class hierarchy relevant to the submodel of an AAS.

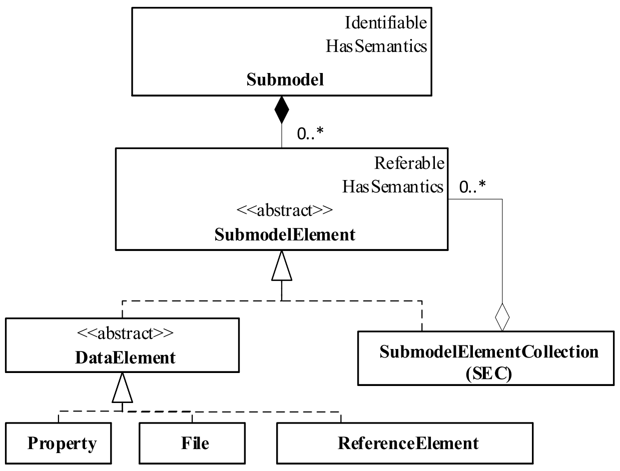

FIGURE 4. Class hierarchy relevant to submodel structure.

As shown in figure, a Submodel defines a composition with the class SubmodelElement. The terms composition here is used to indicate the existence of an attribute in Submodel that contains multiple instances of the class SubmodelElement, hence Submodel is composed by multiple SubmodelElement entities.

SubmodelElement is an abstract superclass for all those entities composing the internal structure of a Submodel, e.g. properties, files, operations. As the reader can notice, Submodel is Identifiable whilst SubmodelElement is Referable. As discussed in the previous subsection, this means that an instance of Submodel is globally and uniquely locatable by its identifier; an instance of SubmodelElement, instead, is locatable only in the context of the parent instance, i.e. a Submodel or another SubmodelElement. 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.

DataElement is an abstract class inheriting from SubmodelElement identifying all those classes that are no further composed out of other SubmodelElements. Property is a concrete class of DataElement, and it represent a property of an asset. The Property class defines attributes to contain data value (*value*) and to specify the type of such data value (*valueType*). Properties are among the most important elements of a Submodel since they constitute the main source of information regarding an asset. 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.

V. PROPOSAL OF AAS SUBMODEL FOR IEC 61131-3

As said in the Introduction, the proposal here presented aims to give a representation using AAS metamodel of each element present in an IEC 61131-3 Configuration. The aim of this section is to point out author's reasonings about how each element relevant to PLC programs based on IEC 61131-3 could be represented using AAS.

An AAS Submodel is proposed to represent an IEC 61131-3 Configuration. In the following, such a submodel will be often referenced as "IEC 61131-3 Submodel".

As IEC 61131-3 Configuration is made up by several elements (e.g. Resources, POUs, Tasks, etc.), definition of AAS SubmodelElements inside the IEC 61131-3 Submodel could be done to represent the IEC 61131-3 elements. SubmodelElements may be organized by SECs introduced in Section IV.C. Organization may be realized in different ways and the relevant choices proposed by the authors will be explained in the following.

The simplest way to organize SubmodelElements is that of using SEC as a folder. 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. This kind of organization is proposed to group SubmodelElements representing IEC 61131-3 elements of the same category (e.g. Variables, Tasks, POUs) into separate folders. The relevant advantage is an easier classification of SubmodelElements.

According to the proposal just described, a SEC acting as a folder organizes SubmodelElements representing homogeneous IEC 61131-3 elements. A problem may occur when a single SubmodelElement is not able to fully represent a particular IEC 61131-3 element. 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. Let us consider the Variable element of IEC 61131-3; as known, it may feature a lot of attributes, like AT, scope, and type. In these cases, the authors propose to use a particular strategy to represent an IEC 61131-3 elements featuring complex structures. It is always based on the use of SEC, but in this case, it represents the IEC 61131-3 element. In order to represent all the complete set of the features of the original element, it is assumed that the SEC is the container of the other SubmodelElements representing these features. Considering again the example of Variables, in this case a SEC may be defined to represent the Variable itself; it will contain SubmodelElements modelling, for example, the attribute AT (e.g. %Q0.0), the scope of the Variable (e.g. local) and the type (e.g. BOOL).

According to the proposal made by the authors, a method to differentiate SEC at a glance is needed because, as previously said, SEC may be used to represent both a folder-like container of SubmodelElements and an IEC 61131-3 element. In this paper, it has been assumed to use the values of the attribute *category* (inherited by the common class Referable) to this aim. In particular, a value "SET" is used for a SEC representing just a container of other SubmodelElements (like a folder), whilst a value "ELEMENT" is used for a SEC representing a complex element made up by several features, each represented by a SubmodelElement organized by the SEC.

The following subsections will point out details of the structure of the proposed IEC 61131-3 Submodel. In order to improve the readability of the paper, authors decided to describe the components of such Submodel by means of examples supported by UML diagrams and with description of the reasonings behind the solutions adopted to model IEC 61131-3 software components.

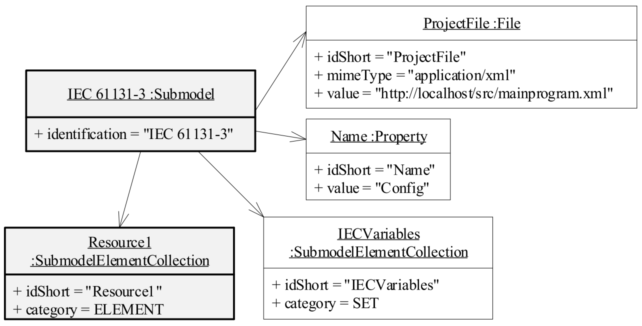
ProjectFile :File
+ idShort = "ProjectFile"
+ mimeType = "application/xml"
+ value = "http://localhost/src/mainprogram.xml"
IEC 61131-3 :Submodel
+ identification = "IEC 61131-3"
Name :Property
+ idShort = "Name"
+ value = "Config"
Resource1 :SubmodelElementCollection
+ idShort = "Resource1"
+ category = ELEMENT
IECVariables :SubmodelElementCollection
+ idShort = "IECVariables"
+ category = SET

FIGURE 5. Representation of an IEC 61131-3 configuration.

A. CONFIGURATION

As previously said, it is assumed that the IEC 61131-3 Configuration is represented in the AAS as a Submodel, as shown in Fig. 5. Properties of the Configuration can be aggregated under this Submodel describing 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).

Configurations contain Variables and Resources, as discussed in Section III.

It has been assumed to organize the representations of Variables under a folder for a better organized hierarchy. A SEC named "IECVariables" is defined under the IEC 61131-3 Submodel to contain information related to Variables. For this reason, the *category* attribute of the SEC "IECVariables" is set to "SET". Details about representation of Variables under this SEC will be given in Section V-G.

Differently, a Resource is represented as a SEC whose *category* is "ELEMENT" and its definition is provided in the Section V.B. The reason of this choice is due to the strategy adopted about the use of SEC element, as explained before.

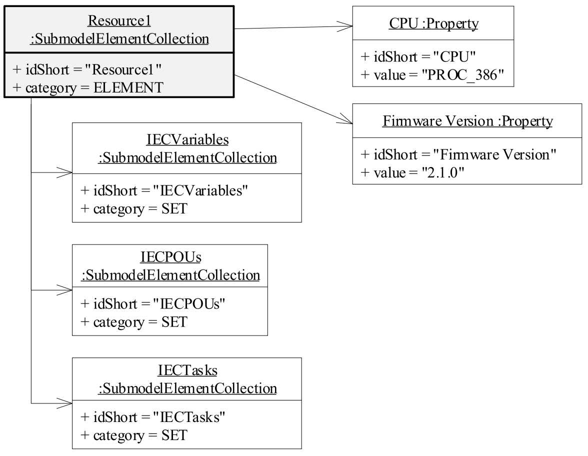
Resource1 :SubmodelElementCollection
+ idShort = "Resource1"
+ category = ELEMENT
CPU :Property
+ idShort = "CPU"
+ value = "PROC_386"
Firmware Version :Property
+ idShort = "Firmware Version"
+ value = "2.1.0"
IECVariables :SubmodelElementCollection
+ idShort = "IECVariables"
+ category = SET
IECPOUs :SubmodelElementCollection
+ idShort = "IECPOUs"
+ category = SET
IECTasks :SubmodelElementCollection
+ idShort = "IECTasks"
+ category = SET

FIGURE 6. Representation of an IEC 61131-3 resource.

B. RESOURCE

A Resource is represented using an instance of SEC with *category* set to "ELEMENT", as shown in Fig. 6. It features Properties specific for the Resource, e.g. name, CPU model, Operating System version, firmware identification.

Resources contain Variables, POUs and Tasks, as discussed in Section III.

A SEC named "IECVariables" is used to collect all the information related to the Variables defined under the Resource; for this reason, its *category* attribute is set to "SET". As said before, details about representation of Variables under this SEC will be given in Section V-G.

A SEC with *category* "SET" named "IECPOUs" is used to collect all the information related to Programs, Function Blocks and Functions that are used inside the relevant Resource. Programs, Function Blocks and Functions will be discussed in the Sections V.C, V.D and V.E, respectively.

A SEC of *category* "SET" named "IECTasks" is used to collect all the information related to the Tasks defined in the relevant Resource. The description of a Task representation is given in Section V.F.

C. PROGRAM

Similarly, to Resource, Program is represented as a SEC of *category* "ELEMENT" inside the Submodel, as shown in Fig. 7.

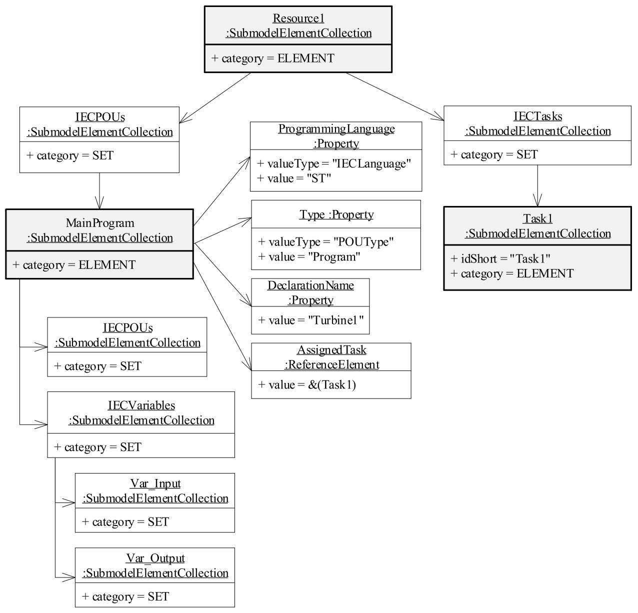
Resource1 :SubmodelElementCollection
+ category = ELEMENT
IECPOUs :SubmodelElementCollection
+ category = SET
ProgrammingLanguage :Property
+ valueType = "IECLanguage"
+ value = "ST"
IECTasks :SubmodelElementCollection
+ category = SET
MainProgram :SubmodelElementCollection
+ category = ELEMENT
Type :Property
+ valueType = "POUType"
+ value = "Program"
Task1 :SubmodelElementCollection
+ idShort = "Task1"
+ category = ELEMENT
DeclarationName :Property
+ value = "Turbine1"
IECPOUs :SubmodelElementCollection
+ category = SET
AssignedTask :ReferenceElement
+ value = &(Task1)
IECVariables :SubmodelElementCollection
+ category = SET
Var_Input :SubmodelElementCollection
+ category = SET
Var_Output :SubmodelElementCollection
+ category = SET

FIGURE 7. 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 that is shown in Fig. 7 is the "ProgrammingLanguage" which is in charge to specify the IEC 61131-3 programming language adopted for the program; the authors have defined the enum type "IECLanguage" inside the AAS Submodel, which provides all the names of the IEC 61131-3 languages as allowed values. Using this enum, it is possible specify the language for each POU modelled by AAS Submodel; Fig. 7 shows the value "ST" for this Property, belonging to the enum type "IECLanguage", meaning that the language used for the Program is Structured Text. Furthermore, an instance of ReferenceElement named "AssignedTask" is defined 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 described in the previous section. Fig. 7 shows that the program represented features a task named "Task1" whose representation is collected under the "IECTasks" SEC shown by the same figure.

As done for Configuration and Resource, a SEC "IECVariables" is defined to represent 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 [9], [19]. 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", respectively, may optionally be defined inside "IECVariable" in order to collect all the entities representing VAR_INPUT and VAR_OUTPUT Variables for the Program, if present.

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.

D. FUNCTION BLOCK

Function Blocks are like Programs; therefore, they can also be represented as a SEC of *category* "ELEMENT", as shown in Fig. 8.

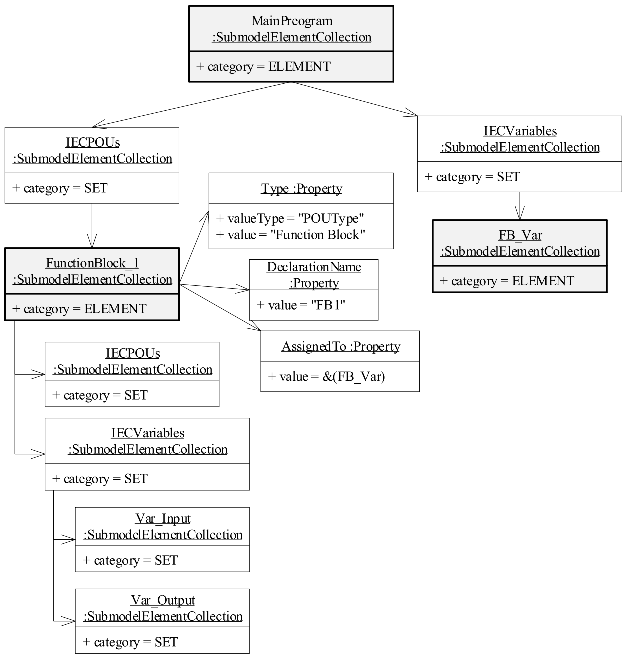
MainPreogram :SubmodelElementCollection
+ category = ELEMENT
IECPOUs :SubmodelElementCollection
+ category = SET
IECVariables :SubmodelElementCollection
+ category = SET
Type :Property
+ valueType = "POUType"
+ value = "Function Block"
FB_Var :SubmodelElementCollection
+ category = ELEMENT
FunctionBlock_1 :SubmodelElementCollection
+ category = ELEMENT
DeclarationName :Property
+ value = "FB1"
AssignedTo :Property
+ value = &(FB_Var)
IECPOUs :SubmodelElementCollection
+ category = SET
IECVariables :SubmodelElementCollection
+ category = SET
Var_Input :SubmodelElementCollection
+ category = SET
Var_Output :SubmodelElementCollection
+ category = SET

FIGURE 8. Representation of an IEC 61131-3 function block and its relationship with an IEC variable.

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). An instance of ReferenceElement named "AssignedTask" can be defined to show which Task the Function Block is eventually associated with, as done for Program in Section V.C.

In the same manner of Programs, a SEC "IECVariables" is used to collect all the information related to the Variables of the Function Block. As a Function Block may be featured by the VAR_INPUT and VAR_OUTPUT Variables, like the program, 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 of other Function Blocks and Functions called by the Function Block (as pointed out by Fig. 2).

Usually, in an IEC 61131-3 Program a Function Block instance is assigned to a Variable; this fact may be represented inside the proposed AAS Submodel with a ReferenceElement named "AssignedTo" (present in Fig. 8) pointing to the SEC representing the Variable containing the Function Block instance. The strategy adopted for this representation will be more detailed in Section V.G.

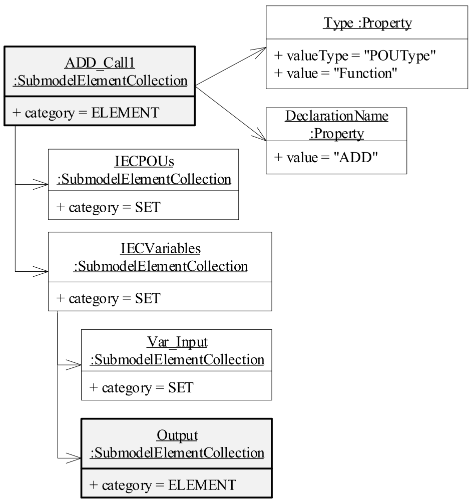
ADD_Call1 :SubmodelElementCollection
+ category = ELEMENT
Type :Property
+ valueType = "POUType"
+ value = "Function"
DeclarationName :Property
+ value = "ADD"
IECPOUs :SubmodelElementCollection
+ category = SET
IECVariables :SubmodelElementCollection
+ category = SET
Var_Input :SubmodelElementCollection
+ category = SET
Output :SubmodelElementCollection
+ category = ELEMENT

FIGURE 9. Representation of an IEC 61131-3 function.

E. FUNCTION

A Function is represented as a SEC with *category* "ELEMENT", as shown in Fig. 9. It may contain Properties related to the Function description like the name and the POU type.

A SEC "IEC Variables" is used to collect all the information related to the Variables of the Function. Functions may have one or any number of input parameters (VAR_INPUT Variables) [9], [19]. As opposed to FBs, they do not have output parameters but return exactly one element as the function (return) value. For this reason, the SECs "Var_Input" can optionally be used to collect input Variables representing the input parameters of the Function. In order to capture the value returned by the Function, a variable named "Output" is defined among the Variables of the Function.

A SEC of *category* "SET" named "IECPOUs" is used to collect, if necessary, all the information relevant to the Function calls present inside the Function.

F. TASK

A Task is represented as a SEC of *category* "ELEMENT" as depicted in Fig. 10. It collects all Properties containing information related to the task, e.g. Interval and Priority.

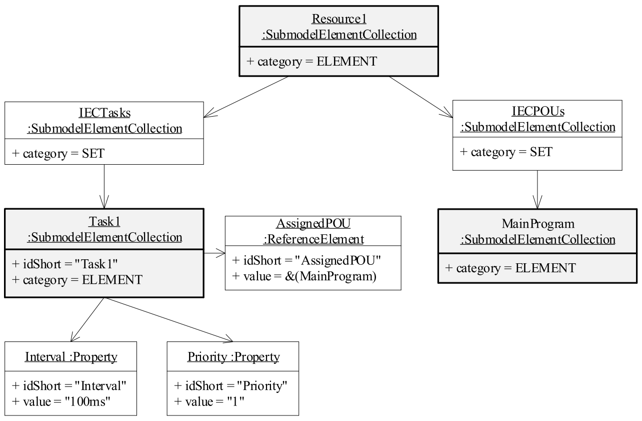

FIGURE 10. Representation of an IEC 61131-3 task and its relationship with POU.

It is worth noting that information about which POU is running under the Task can be represented by means of a ReferenceElement, in the same manner as "AssignedTask" for IEC Programs and IEC Function Blocks. Such ReferenceElement could be named "AssignedPOU". As shown in figure, the element pointed by "AssignedPOU" is a SEC representing a POU and contained in the SEC "IECPOUs" of the relevant Resource.

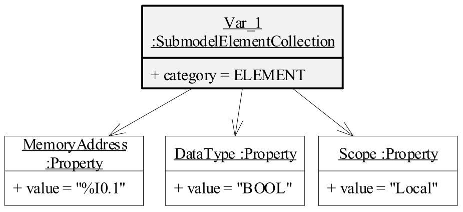

FIGURE 11. Representation of an IEC 61131-3 variable.

G. VARIABLE

A single Variable is represented as a SEC of *category* "ELEMENT" as shown in Fig. 11. It contains Properties related to the description of the Variable itself, e.g. Name, Retentive, Scope, DataType Value and Address.

Inside each single POU the value of a Variable may be assigned to another Variable; moreover, a Variable may be assigned to input/output Variable of a POU. Other possible scenarios are represented by a Variable containing the result of a Function output and by a Variable containing the instance of a Function Block. Finally, it may happen that a value of a (dependent) Variable is given by an expression applied to one or more (independent) Variables. All these examples point out the existence of relationships among Variables and other elements inside the IEC 61131-3 Submodel.

The authors propose to represent these relationships through suitable ReferenceElement instances defined between Variables and/or between Variables and SECs in the IEC 61131-3 Submodel. Two ReferenceElements named "AssignedFrom" and "AssignedTo" 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. Fig. 8 depicts an example of usage of the "AssignedTo" ReferenceElement.

When the content of a Variable in the IEC 61131-3 program is obtained from the result of an expression containing other Variables, it has been assumed that the counterpart in the AAS representing such Variable (i.e. SEC) features one or more ReferenceElements named "DependsOn" pointing to the representations of the relevant Variables used in the expression. For example, 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.

H. SEMANTICS

The previous subsections presented the main elements composing the IEC 61131-3 Submodel proposed in the paper.

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 (i.e. what each element represents) must be clearly understood by all the partners of the value chain. When the IEC 61131-3 Submodel is explored, it must be clear what each element represents regarding the IEC 61131-3 standard. For instance, value-chain partners must be able to distinguish a SEC representing an IEC 61131-3 Variable from a SEC representing an IEC 61131-3 Resource.

Since each element inside the IEC 61131-3 Submodel is defined as HasSemantics (i.e. an entity that inherits from HasSemantics common class), the relevant *SemanticId* attribute of each element references a semantic description in a Semantics Repository properly defined. This repository has been realized as a set of the official names of IEC 61131-3 components, e.g. "Configuration", "Task", "POU" and so on. The repository could be considered like a dictionary of IEC 61131-3 terms adopted by the standard. In this way when accessing a specific element of the IEC 61131-3 Submodel, the *SemanticId* attribute allows to point to the IEC 61131-3 terms which define the role of the element inside the IEC 61131-3 standard. In this way the role, and thus the relevant meaning, is univocally defined end exposed.

It has been assumed that this Semantics Repository must be shared between all the value-chain partners using the AAS IEC 61131-3 Submodel.

VI. REPRESENTING PLC AND REAL PLANT BY AAS

It is worth noting that an IEC 61131-3 program is strictly related to several details of the plant (or its subset) to be controlled; these details may include, for example, the control devices (e.g. sensors, actuators) to be connected to the PLC and the exchange of information (e.g. shared variables) with other PLCs and/or computing devices among which the control application is distributed. Representation of the IEC 61131-3 program alone by AAS (as described in Section V) cannot take into account all these details, as explained in the following.

As known, a PLC generally features a set of electrical connections to which control devices are attached depending to the characteristics of those connections (e.g., input, output, 0-24V digital, 0..20mA analog, 4-20mA analog). Sometimes, these connections are organized into I/O modules mounted in racks. Fig. 12 shows a very simple PLC featuring input and output connections; it is a so-called compact PLC as it does not feature modular racks, but just embedded electrical connections.

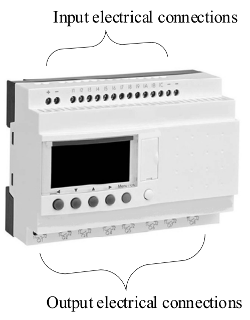

FIGURE 12. PLC and I/O electrical connections.

When configuring a PLC, electrical connections are internally mapped to memory locations of the I and Q memory blocks [19] to which internal variables may be associated through the AT attribute (see Section III). When a PLC is used to control a real plant, each I/O electrical connection is connected with the proper device that the PLC controls or receives information from. On the basis of what said, let us consider a boolean Variable named Test inside a IEC 61131-3 program; moreover, let us assume that the Variable features the attribute AT representing the mapping to the memory location %Q0.0 (i.e. output process image memory). Furthermore, let us assume that during the configuration phase of the PLC (where the IEC 61131-3 program must be executed), a 0-24V digital output connection of the PLC is associated to the memory location %Q0.0. Finally, let us assume that a pump is physically connected to this output. This means that a strict relationship between Variable Test, the output electrical PLC connection and the controlled actuator exists such that the pump is switched on every time the value of the Variable Test is true. This relationship is partially included in the AAS IEC 61131-3 Submodel described in the previous section, because this last is able to represent only the mapping between the Variable Test and the memory location (e.g. %Q0.0), as said in Section V.G (see Fig. 11). The description of the relationships between internal variables and physical devices is missing in the AAS IEC 61131-3 Submodel.

Another example of relationships between the PLC and the real plant may consist 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 have the need to share one or more information (e.g. variables). For instance, let us consider the case of a Variable defined in the IEC 61131-3 program running on a certain PLC and let us assume that it must be set at run-time by means of 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.

These examples are very simple, but they point out the limitations of the AAS IEC 61131-3 Submodel presented in the previous section. The authors propose a methodological solution to provide a more general description by means of AASs. This solution also involves the representation of the relationships between IEC 61131-3 programs running on a PLC and the "world" around it (at least, the portion of environment which has actual dependencies which the programs running in the PLC). The remainder of this section is aimed to describe this solution.

First, an AAS describing the PLC (or in general the computing device where the IEC 61131-3 program runs) must be defined. It must include the IEC 61131-3 Submodel (described in Section V) and should be enriched by the definition of other Submodels aimed to represent the actual I/O electrical connections provided by the PLC. It is assumed that such kind of Submodel contains Properties describing the features of the physical connections. On the basis of this assumption, it is worth noting that relationships between each Variable in the IEC 61131-3 program and each physical connection may be modelled in the AAS by ReferenceElements. The references may connect the elements modelling the Variables inside the IEC 61131-3 Submodel with the elements representing the physical connections the Variables are referring to. The formers are present in the IEC 61131-3 Submodel, whilst the latter ones are located in the AAS Submodel representing I/O connections. Therefore, ReferenceElements connect elements of two different Submodels of the same AAS modelling the PLC.

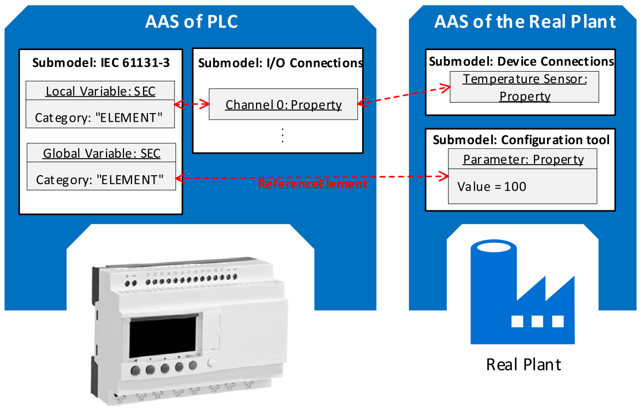

FIGURE 13. AASs representing PLC and the real plant.

Fig. 13 shows on the left an example of AAS modelling the PLC shown in Fig. 12 (and present in the same figure). This AAS includes a Submodel representing the IEC 61131-3 programs running on the PLC, defined according to the content of Section V; for the ack of space, only two SECs of *category* "ELEMENT" are shown representing a local Variable and a global Variable, respectively. According to the content of Section V, the local Variable may be represented by a SEC of *category* "ELEMENT" organized inside SEC representing POU (e.g. a Program), whilst the global Variable may be represented by another SEC of *category* "ELEMENT" placed in a SEC representing a Resource. It has been assumed that the local Variable features the AT attribute mapping it to a certain memory location; we consider the location %I0.0 in the following.

Fig. 13 shows that another Submodel is present inside the AAS modelling the PLC. This Submodel contains the representations of the I/O connections of the PLC, among which there is the input connection associated to the memory address %I0.0. In figure, this Submodel contains a Property named "Channel 0" representing the above-mentioned input connection. Due to the mapping of the local Variable to the address %I0.0 and to the assignment of "Channel 0" to the same address, a relationship exists between the local Variable and the "Channel 0" Property. This relationship is represented by a ReferenceElement (drawn as a dashed red arrow) connecting the local Variable and the "Channel 0" Property, as shown on the left of Fig. 13.

In order to represents the "world" around the PLC, the authors assume the existence of another AAS modelling the real plant (or the subset of it) controlled by the PLC. Details about the realization of this kind of AAS are not given in this paper as they are out of scope and they strictly depends on the actual features of the real plant to be represented. Such AAS may feature Submodels based on specific existing standards depending on the features of the plant to be represented. For example, the standards IEEE 1364 [22] and IEEE 1800 [23] may be used for the hardware description of particular devices present in the plant; a Submodel may be based on IEC 61804 Electronic Device Description Language (EDDL) [24] for the description of digital communication characteristics of intelligent field instrumentations and equipment parameters. A Submodel for condition monitoring may be based on VDMA 24582 [25], and a Submodel representing parameters related to energy efficiency may be based on ISO/IEC 20140-5 [26].

The only assumption made in this paper is that for each device in the plant (e.g. sensor, pump, motor) connected to the PLC, a Submodel must be present in order to represent the relevant physical connections used for the communication with the PLC. As done for the PLC, the Submodel must contain one or more Properties representing the features of the physical connections of the device.

Let us assume that in real plant, a temperature sensor is physically connected with the input terminal represented by the Property "Channel 0" shown in Fig. 13. Among the available Submodels of the AAS modelling the plant, there is one called "Device Connections" in Fig. 13, which has been assumed to contain properties relevant to the physical connections of each device present in the real plant. One of these properties is named Temperature Sensor and represents the physical connections of the relevant sensor with the PLC. In order to represent the relationship between this sensor and the input connection represented by the Property "Channel 0", a ReferenceElement may be used to connect the two Properties of the two different Submodels, as shown by Fig. 13. In this way the relationships between the local Variable, the PLC input terminal and the relevant input device is clearly represented flowing the ReferenceElement instances shown by the same figure.

The AAS modelling the real plant may contain other kinds of Submodels. 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. Again, it has been assumed that for each application sharing variables with the PLC, an AAS Submodel must be present to represent the variable. Fig. 13 for example shows a Submodel named "Configuration tool" containing a Property (named "Parameter") modelling a setting value that must be used inside the PLC Program to fill a Global Variable. The ReferenceElement shown in Fig. 13 aims to connect this Global Variable with the Property "Parameter" of the Submodel "Configuration tool" inside the AAS representing the real plant. Again, the relationships between the PLC Variables and external applications may be clearly represented flowing the ReferenceElement elements.

Considering Fig. 13, it is important to point out that the set of AASs and the relevant relationships proposed in this paper are able to give an overview of the connections between elements composing the entire plant, including devices, applications and I/O terminals.

VII. CASE STUDY: DRILLING MACHINE

In this section an application of the proposed approach presented in Sections V and VI will be provided. The use case here discussed considers a PLC controlling a drilling machine, as depicted in Fig. 14.

The drilling mechanism may be moved up and down through a vertical guide shown in the figure. The drilling machine features two sensors; SensorUp is a limit switch which gives the value ON when the drill is in the upmost upright position. The figure shows another sensor, which is a proximity sensor that gives the value ON when the drill bit is close to the piece to be drilled. The drilling machine features a motor able to move up and down the drilling mechanism. The motor receives the DrillDown and DrillUp commands to move the driller down and up, respectively. The Rotate command is used to rotate the drill bit. Fig. 14 shows the Start signal which represents the command given by an operator (e.g. pushing a button) to start the control program. The same figure shows the connections between the sensors/actuators of the drilling machine, the Start push button and the I/O physical connections of the PLC.

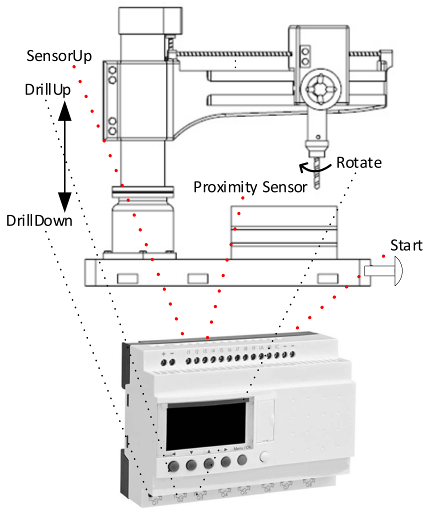

FIGURE 14. PLC and drilling machine.

The control program running inside the PLC is listed in Fig. 15; it is written using the Structured Text programming language, according to the IEC 61131-3 standard [9].

The Program Drill uses a global Variable (T_PARAM), defined at the Resource level. It is assumed that this Variable is filled using a value retrieved by a configuration tool running in another computing device not shown in the Fig. 14.

Several local binary Variables are defined as shown in Fig. 15; they are assigned to the I and Q process images and to the memory M, through the AT attribute. The list of local Variables includes a local instance of the TON timer [9]; in particular, the Variable Timer01 is used to contain the instance of the TON timer FB.

The ST code shown by Fig. 15 realizes a very simple control program; when the Start push button is pressed, the program starts, and the DrillDown command is activated. Once the Proximity Sensor assumes the value ON (due to the drill pit closeness 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 given by a user-configurable global Variable called T_PARAM. Once the time expires, the drilling machine moves up (i.e. DrillUp command is activated) until it reaches the upmost upright position (i.e. SensorUp is ON); the drilling machine stops when this condition occurs. It has been assumed that the control program is restarted when the operator presses the Start push-button again.

The Fig. 15 shows the definition of the Configuration (named "Config1"), which includes the Resource "Resource1". This resource features the global shared Variable T_PARAM, and a periodic task named "MainTask1" featuring a time interval of 100ms controlling the execution of the Program Drill.

According to the approach presented in Section VI, PLC and the real plant controlled (i.e. the drilling machine and

```
PROGRAM Drill
VAR_EXTERNAL
      T_PARAM: TIME;
END_VAR
VAR
  DrillDown AT %Q0.0 : BOOL;
  DrillUp AT %Q0.1 : BOOL;
  DrillRotate AT %Q0.2 : BOOL;
  Start AT %I0.0 : BOOL;
  Sensor AT %I0.1 : BOOL;
  SensorUp AT %I0.2 : BOOL;
  EndDrill AT %M0.0 : BOOL:=FALSE;
  Timer01: TON;
END_VAR
Timer01(IN:=Sensor, PT:=T_PARAM);
IF Start THEN
          IF NOT EndDrill AND NOT Sensor THEN
                    DrillDown:=1;
                    DrillRotate:=1;
          END_IF
          IF NOT EndDrill AND Timer01.Q THEN
                    DrillDown:=0;
                    DrillRotate:=0;
                    EndDrill:=1;
          END_IF
          IF EndDrill THEN
                    DrillDown:=0;
                    DrillRotate:=0;
                    DrillUp:=1;
          END_IF
          IF EndDrill AND SensorUp THEN
                    DrillDown:=0;
                    DrillRotate:=0;
                    DrillUp:=0;
                    EndDrill:=0;
          END_IF
END_IF
END_PROGRAM

CONFIGURATION Config
  RESOURCE Resource1
    VAR_GLOBAL
      T_PARAM: TIME := T#10s;
    END_VAR
    TASK MainTask1(INTERVAL :=T#100ms, PRIORITY := 1);
    PROGRAM MainInst1 WITH MainTask1: Drill;
  END_RESOURCE
END_CONFIGURATION
```

FIGURE 15. IEC 61131-3 PLC program.

the relevant control devices) have been modelled by the two AASs shown by Fig. 16.

The AAS of the PLC contains several Submodels, including the IEC 61131-3 Submodel, defined following the approach proposed in Section V; its internal details will be deeply discussed in the following.

Fig. 17 depicts the representation of the drilling machine program using an IEC 61131-3 Submodel created following the guidelines presented in Section V. It is a UML Object Diagram where the IEC 61131-3 Submodel and the SECs with *category* "ELEMENT" are highlighted with gray-colored objects. The semantic of each element in Fig. 17 must be deduced by the attribute *semanticId*, as explained in Section V.H; the attribute *semanticId* is not represented in figure for space reason.

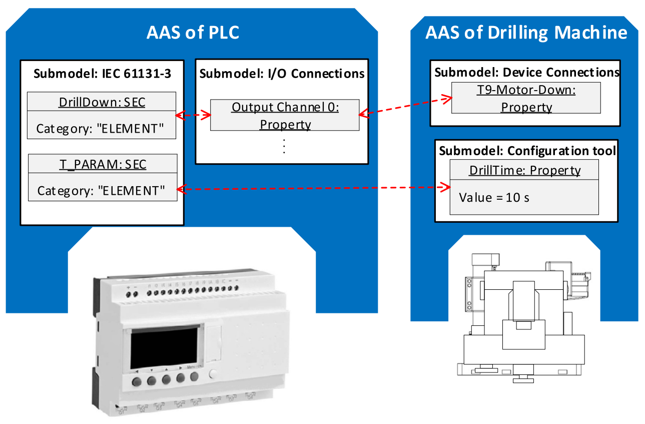

FIGURE 16. Case study scenario showing relationships between variables and properties of AASs of both a PLC and a drilling machine.

In Fig. 17 the topmost element is a Submodel named "IEC 61131-3" representing the Configuration of the program in Fig. 15; as for that, two SubmodelElements are connected to the Submodel: one is a Property providing the name of the Configuration in the program (i.e., Name), whilst the other one is a File containing the path to the project file containing the program (i.e. ProjectFile).

The SEC "Resource1" represents the Resource defined inside the program. As explained in Section V, this last contains the value "ELEMENT" in the attribute *category* because this SEC represents a real software entity (i.e. the IEC Resource instance "Resource1") and not a folder-like entity. This SEC, in turn, contains three SECs: "IECPOUs", "IECVariables", and "IECTasks". The attribute *category* of these last contains the value "SET" because they represent folder-like entities organizing SubmodelElements representing POUs, Variables and Tasks, respectively. It is worth noting that the IEC 61131-3 elements represented by these SubmodelElements are only the ones related to the Resource named Resource1. In other words, the hierarchy of SubmodelElements reflects the contextual relationship in the IEC 61131-3 Program.

The global Variable T_PARAM is mapped as SEC and organized under "IECVariables". It features some Properties reflecting the relevant scope, value, and data type; their attributes are filled according to the definition of the Variable in the program in Fig. 15.

The Task "MainTask1" is mapped as a SEC and organized under "IECTasks". It features two properties containing the interval and the priority of the Task, filled according to the declaration in the program in Fig. 15.

The Program instance "MainInst1" declared in Fig. 15 is mapped as a SEC and organized under IECPOUs of Resource1. As previously said, this organization of the Submodel highlights the relationship between "Resource1" and "MainInst1", showing which Program is assigned to the Resource. This SEC features some Properties describing the Program like the programming language adopted, the kind

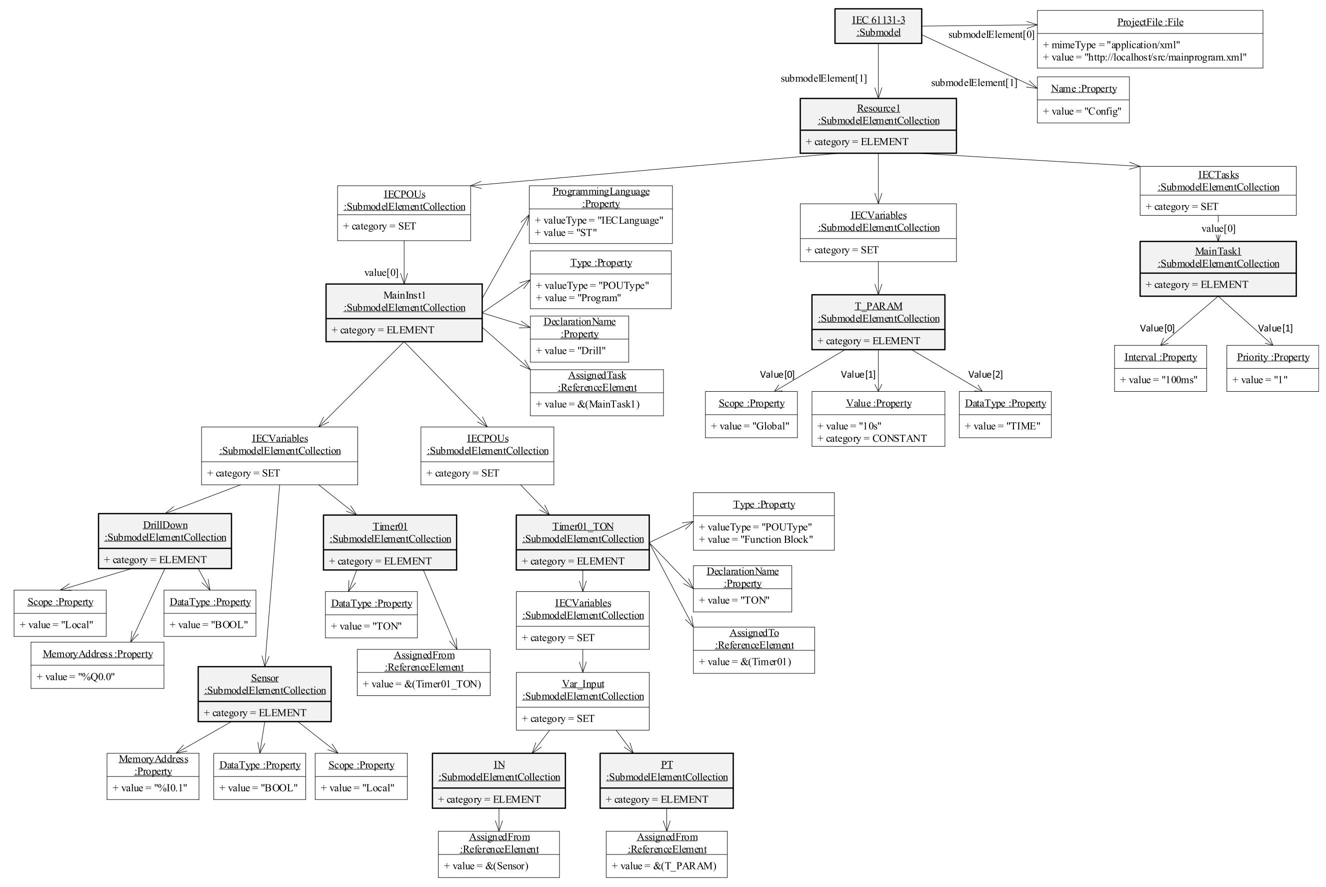

FIGURE 17. **Representation of the drilling machine PLC program using a submodel.**

of POU and the name used for the declaration of the Program in Fig. 15. It is worth noting that the ReferenceElement "AssignedTask" is used here to highlight that the program MainInst1 is executed under the Task "MainTask1"; this is reflected from the attribute *value* which contains a Reference to the SEC MainTask1.

Like "Resource1", "MainInst1" features a SEC "IECVariables" and a SEC "IECPOUs" organizing all the SubmodelElements mapping Variables and Function Blocks/Functions used inside the Program, respectively. Due to lack of space, only few Variables are represented, i.e. DrillDown, Sensor and Timer01. As usual, DrillDown features Properties describing the Variable, like scope, data type and memory address associated (by AT attribute) to the Variable. Similar consideration can be done for Sensor too. Instead, the Variable Timer01 shows 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 SEC representing the relevant Function Block, i.e. Timer01_TON. This last will be described in the following.

As shown in Fig. 15, the Program "MainInst1" use a Function Block (i.e. TON). For this reason, the instance of the Function Block is created as SEC named "Timer01_TON" and organized under "IECPOUs" of "MainInst1". The name "Timer01_TON" is chosen to logically differentiate 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, as shown in Fig. 15. As discussed in Section V.G, a SEC "IECVariables" is used to organize all the Variables used inside the Function Block. A 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 IN and PT, a ReferenceElement "AssignedFrom" is used to show which Variables are passed as argument to the "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 depicted in Fig. 15.

Coming back to the Fig. 16, it is possible to notice that the AAS representing the PLC contains a Submodel "I/O Connections". According to the approach presented in Section VI, it features properties describing the I/O physical connections of the PLC. In particular, the "Output Channel

0'' Property is shown; it refers to the output connection associated to the memory location %Q0.0. A shown by Fig. 15, DrillDown Variable features the AT attribute set to %Q0.0. For this reason, a ReferenceElement is considered between this Variable and the Property ''Output Channel 0'', as shown by Fig. 16.

Let us consider the other AAS shown by Fig. 16. This AAS represents the drilling machine; among its Submodels not shown in the figure, this AAS contains a Submodel ''I/O Device Connections'', which has been assumed to contain properties relevant to the physical connections of each device present in the drilling machine. Among these connections there is that relevant to the Motor in charge to move the drill up and down. The Property ''T9-Motor-Down'' represents the physical connection relevant to the command which moves the drill up. On the basis of the program shown by Fig. 15 this physical connection must be linked to the PLC terminal associated to the memory location %Q0.0. For this reason, the ReferenceElement shown in the Fig. 16 represents the relationship between the Property ''Output Channel 0'' and the Property ''T9-Motor-Down'' of the two AASs.

As said before, it is assumed that a configuration tool running in another computing device is present in the ''world'' around the PLC. Fig. 16 contains another Submodel named ''Configuration tool'' containing all the information relevant to the configurable parameters of the drilling machine set by a configuration tool. Such values must be used inside the PLC Program (e.g. drill time, rotation speed, etc.).

The Property ''DrillTime'' represents the Variable whose current value is downloaded into the PLC and assigned to the Variable T_PARAM. For this reason, the Property ''DrillTime'' exposes a relationship with the Variable T_PARAM of the PLC program represented with the ReferenceElement shown in Fig. 16.

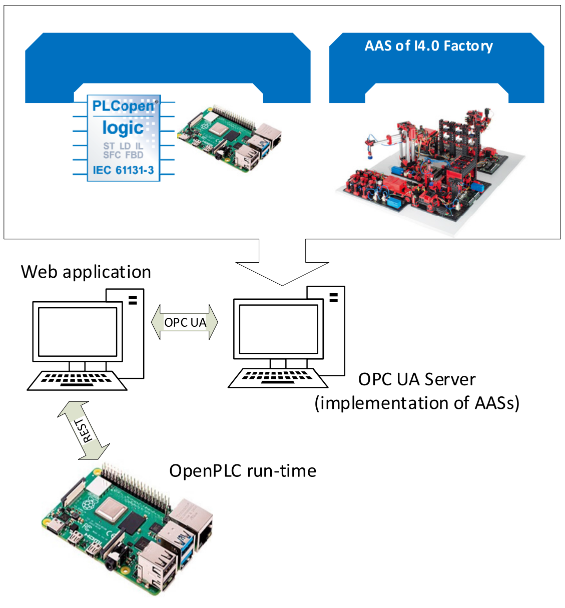

FIGURE 18. Implementation scenario taken into consideration.

VIII. IMPLEMENTATION

An implementation of the proposed approach has been realized. Fig. 18 shows the scenario considered for the implementation.

It has been assumed to realize an IEC 61131-3 program controlling an educational factory model produced by fischertechnik [27]; it is composed by different working stations as shown by Fig. 18, i.e. warehouse, gripper, hoven, and sorting line. Both software development tool and runtime system for the IEC 61131-3 program are based on OpenPLC [28]. OpenPLC has been used as it is open source allowing to avoid the use of proprietary PLC run-time and editor software. OpenPLC may be installed on different kinds of embedded systems. Among the compliant embedded systems there is the Raspberry Pi which was used in the implementation scenario. Although Raspberry Pi is not a PLC, the OpenPLC run-time allows it to behave as a PLC. The OpenPLC run-time was installed on the Raspberry Pi allowing to execute an OpenPLC project containing the control program of the factory model.

The AASs of both PLC and factory model have been defined according to the approach presented in Section V and VI. The two AASs have been implemented using OPC UA [29] which is one of the technologies suggested in [6] for the implementation of AAS metamodel. In particular, the two AASs have been realized by mapping each AAS element into OPC UA Nodes according to the proposal made by the authors in [30]. This proposal is based on the use of an OPC UA information model defined by the same authors and aimed to map each AAS element into OPC UA Nodes; the information model is called ''CoreAAS'' and it is freely available in [31]. AAS implementation into OPC UA has been realized using the OPC UA SDK [32] for Node.js. According to this implementation choice, the OPC UA Server maintains the representation in OPC UA of the two AASs describing the control device and the factory model. A generic OPC UA Client is able to explore the OPC UA Server accessing the two AASs to discover all the details about the IEC 61131-3 programs and the relationships between IEC 61131-3 Variables, I/O connections, and the factory model's control I/O devices.

A web application has been developed as it will be described in the remainder. It includes an OPC UA Client for the communication with the OPC UA Server, allowing the web application to give to a web user the description of the entire system made up by the PLC and the controlled plant. The web application is also in charge to allow the creation of an AAS representation into OPC UA, given the relevant UML description. Updates of existing AASs maintained by OPC UA Server are also possible using the web application. Fig. 18 points out that the web application communicates also with the Raspberry Pi (and with the OpenPLC run-time); communication is based on Rest services. Using these services, it is possible to upload an OpenPLC program from the web application towards the Raspberry Pi. The web application has been developed in AngularJS and deployed on a web server application based on Node.js.

Each component of the implementation here described is freely available on GitHub [33], under Apache 2.0 license.

Using the scenario depicted by Fig. 18 just described, several tests have been carried out. Among them there is one about a reconfiguration process which will be described in the following. A production process has been defined; an IEC 61131-3 program has been written to perform a certain control algorithm on the factory model, involving a certain set of the available control devices connected to certain I/O pins of the Raspberry Pi. The two AASs depicted in Fig. 18 have been defined in UML and the OPC UA representation has been created inside the OPC UA Server by the web applications. The test involved the introduction of some modification in the production process (e.g. changing the control algorithm on which the IEC 61131-3 program is based, adding/removing control devices, changing the relevant connections with the I/O pins of the Raspberry Pi). The web application was used to update the AASs' representation based on OPC UA and maintained inside the OPC UA Server. Once this update has been done, the web application was used to access the OPC UA Server to acquire the current OpenPLC-based project which was uploaded into the OpenPLC run-time running into the Raspberry Pi.

The following consideration may be done to point out the relationship between the test just described and a real scenario concerning reconfiguration process. In a real scenario the AASs representations maintained by the OPC UA Server may be used by technician teams to introduce the real modification into the plant, after the reconfiguration process. The availability of a unique and complete vision of the plant greatly simplifies the work of the technician 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 concluded.

IX. CONCLUSION

The paper presented the use of AAS metamodel to represent an IEC 61131-3 program and its relationships with the real plant controlled. 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. Considering an Industry 4.0 scenario where all assets, including PLCs, are represented in the digital world with their own AASs, every relationship between properties of different assets are tracked and available in the network and structured in a standard manner.

The proposed solution may make easier the development and the reconfiguration of the production system. The test presented in Section VIII is an example of advantages of the proposed approach in a real process reconfiguration scenario. Furthermore, maintenance of control programs is greatly simplified because all the information relevant to variables are documented and semantically enriched in the AASs. If the developer 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 device's AAS. Furthermore, parameters inside the IEC 61131-3 program can be configurated automatically spilling the value from the information contained in the AAS of the relevant machine or device. This is possible because each software component inside the PLC points to the asset's property it refers to.

The work here presented do not discuss how the AAS Submodel for the IEC 61131-3 program must be created and who is in charge of its creation. These topics are out of the scope of this paper but can be considered in future work. Authors strongly believe that automatic creation of an AAS Submodel starting from an existent PLC program is feasible and it represents a very important issue. IEC 61131-3 program are contained in XML-based project file according to the novel IEC 61131-10 standard part [34]. A solution for the automatic creation of AAS Submodel representing IEC 61131-3 programs may involve the use of annotated statement 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 file 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. 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.

The paper pointed out that an implementation of the proposed approach is maintained and freely available on GitHub repository, allowing whatever interested researcher to download and experiment its use in real scenarios, introducing improvements as needed.

ACKNOWLEDGMENT

The authors would like to express their very great appreciation to Dr. Alessandro Messina and Dr. Marco Pisasale for their valuable and constructive suggestions and contributions during the planning and development of this research work, with particular reference to the implementation of the prototype of the proposal.

REFERENCES

[1] S. Lass and N. Gronau, "A factory operating system for extending existing factories to industry 4.0," *Comput. Ind.*, vol. 115, Feb. 2020, Art. no. 103128.

[2] S. Kumar Panda, T. Schröder, L. Wisniewski, and C. Diedrich, "Plug&Produce integration of components into OPC UA based data-space," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2018, pp. 1095–1100, doi: 10.1109/ETFA.2018.8502663.

[3] A. Lüder, N. Schmidt, and R. Drath, "Standardized information exchange within production system engineering," in *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, S. Biffl, A. Lüder, D. Gerhard, eds. Cham, Switzerland: Springer, 2017, pp. 235–257.

[4] J. Fuchs, J. Schmidt, J. Franke, K. Rehman, M. Sauer, and S. Karnouskos, "I4.0-compliant integration of assets utilizing the asset administration shell," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2019, pp. 1243–1247, doi: 10.1109/ETFA.2019.8869255.

[5] *Reference Architecture Model Industrie 4.0 (RAMI4.0), DIN SPEC 91345:2016-04*, Beuth Verlag GmbH, Berlin, Germany, 2016.

[6] *Details of the Asset Administration Shell, Platform Industrie 4.0*, ZVEI, Frankfurt, Germany, Nov. 2019.

[7] M. Azarmipour, H. Elfaham, C. Gries, and U. Epple, "PLC 4.0: A control system for industry 4.0," in *Proc. 45th Annu. Conf. IEEE Ind. Electron. Soc. IECON*, vol. 1, Oct. 2019, pp. 5513–5518, doi: 10.1109/IECON.2019.8927026.

[8] R. Langmann and L. F. Rojas-Peña, "A PLC as an industry 4.0 component," in *Proc. 13th Int. Conf. Remote Eng. Virtual Instrum. (REV)*, Feb. 2016, pp. 10–15, doi: 10.1109/REV.2016.7444433.

[9] *Programming languages, International Electrotechnical Commission (IEC)*, document IEC 61131-3:2013, Programmable Controllers, 2013.

[10] Langmann and Stiller, "The PLC as a smart service in industry 4.0 production systems," *Appl. Sci.*, vol. 9, no. 18, p. 3815, Sep. 2019.

[11] M. Wenger, A. Zoitl, and T. Müller, "Connecting PLCs with their asset administration shell for automatic device configuration," in *Proc. IEEE 16th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2018, pp. 74–79, doi: 10.1109/INDIN.2018.8472022.

[12] H.-T. Park, J.-G. Kwak, G.-N. Wang, and S. C. Park, "Plant model generation for PLC simulation," *Int. J. Prod. Res.*, vol. 48, no. 5, pp. 1517–1529, 2010, doi: 10.1080/00207540802577961.

[13] S. C. Park, C. M. Park, G.-N. Wang, J. Kwak, and S. Yeo, "PLCStudio: Simulation based PLC code verification," in *Proc. Winter Simul. Conf.*, Dec. 2008, pp. 222–228.

[14] S. C. Park, M. Ko, and M. Chang, "A reverse engineering approach to generate a virtual plant model for PLC simulation," *Int. J. Adv. Manuf. Technol.*, vol. 69, pp. 2459–2469, Aug. 2013, doi: 10.1007/s00170-013-5209-1

[15] J. M. Machado, B. Denis, J.-J. Lesage, J.-M. Faure, and J. C. L. F. Da Silva, "Logic controllers dependability verification using a plant model," *IFAC Proc. Volumes*, vol. 39, no. 17, pp. 37–42, 2006.

[16] S. C. Park, C. M. Park, and G.-N. Wang, "A PLC programming environment based on a virtual plant," *Int. J. Adv. Manuf. Technol.*, vol. 39, nos. 11–12, pp. 1262–1270, Dec. 2008, doi: 10.1007/s00170-007-1306-3.

[17] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *J. Comput. Design Eng.*, vol. 1, no. 3, pp. 213–222, Jul. 2014.

[18] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 648–660.

[19] R. W. Lewis, "Programming industrial control systems using IEC 61131-3," in *IEE Control Engineering, The Institution of Electrical Engineers*. Stevenage, U.K.: Institution of Engineering and Technology, 1998.

[20] *Common Data Dictionary (CDD)*, document IEC 61360, V2.0014.0016, International Electrotechnical Commission (IEC), 2018.

[21] *Platform Industrie 4.0, Structure of the Administration Shell—Continuation of the Development of the Reference Model for the Industrie 4.0 Component, Berlin: Federal Ministry for Economic Affairs and Energy (BMWi)*, Federal Ministry Econ. Affairs Energy (BMWi), Berlin, Germany, 2016.

[22] *IEEE Standard for Verilog Hardware Description Language*, Standard 1364-2005, IEEE Computer Society, 2006.

[23] *IEEE Computer Society, IEEE Standard for System Verilog-Unified Hardware Design, Specification, and Verification Language*, Standard 1800-2017, 2017.

[24] *Function Blocks (FB) for Process Control and Electronic Device Description Language (EDDL)—Part 2: Specification of FB concept*, document IEC 61804-2:2018, 2018.

[25] *Fieldbus Neutral Reference Architecture for Condition Monitoring in Production Automation*, document VDMA 24582:2014-04, Verband Deutscher Maschinen-und Anlagenbau E.V. (VDMA), Frankfurt, Germany, 2014.

[26] *Automation Systems and Integration—Evaluating Energy Efficiency and Other Factors of Manufacturing Systems that Influence the Environment—Part 5: Environmental Performance Evaluation Data*, document ISO 20140-5, 2017

[27] *The Training Models From Fischertechnik*. Accessed: Jun. 3, 2020. [Online]. Available: https://www.fischertechnik.de/en/products/simulating/training-models

[28] *The OpenPLC Project*. Accessed: Jun. 3, 2020. [Online]. Available: https://www.openplcproject.com/

[29] W. Mahnke, S. H. Leitner, and M. Damm, *OPC Unified Architecture*. Berlin, Germany: Springer-Verlag, 2009.

[30] S. Cavalieri and M. G. Salafia, "Insights into mapping solutions based on OPC UA information model applied to the industry 4.0 asset administration shell," *Computers*, vol. 9, no. 2, p. 28, Apr. 2020, doi: 10.3390/computers9020028.

[31] *CoreAAS Repository*. Accessed: Jun. 3, 2020. [Online]. Available: https://github.com/OPCUAUniCT/coreAAS

[32] *node-Opcua-Coreaas Repository*. Accessed: Jun. 3, 2020. [Online]. Available: https://github.com/OPCUAUniCT/node-opcua-coreaas

[33] *AAS-for-PLC Repository*. Accessed; Jun. 3, 2020. [Online]. Available: https://github.com/OPCUAUniCT/AAS-for-PLC

[34] *PLC Open XML Exchange Format, International Electrotechnical Commission (IEC)*, document IEC 61131-10:2019, Programmable controllers, 2019.

• • •