IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 8, 2018, accepted October 24, 2018, date of publication November 9, 2018, date of current version December 7, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2880040

Realising Interoperability Between OPC UA and OCF

SALVATORE CAVALIERI, MARCO GIUSEPPE SALAFIA, AND MARCO STEFANO SCROPPO
Department of Electrical, Electronic and Computer Engineering, University of Catania, 95125 Catania, Italy
Corresponding author: Salvatore Cavalieri (salvatore.cavalieri@unict.it)

ABSTRACT The paper deals with the interoperability issue inside Industry 4.0. Definition and adoption of communication standards are of paramount importance to improve the interoperability of industrial applications. For this reason, during the last few years, different organisations have developed reference architectures to align standards in the context of the fourth industrial revolution. Among them, an important role is played by OPC UA international standard (IEC 62541). Application in industrial contexts of modern Information & Communication Technology (ICT) concepts, such as Internet of Things (IoT), is a key-point in Industry 4.0. For this reason, the current literature presents several proposals aimed to improve interoperability between reference standards in Industry 4.0 and IoT ecosystems. The paper proposes a solution towards interoperability between OPC UA and IoT, based on the mapping of OPC UA and Open Connectivity Foundation (OCF) specifications defined very recently to enable the interoperability inside the IoT ecosystem. Although other examples of integration of OPC UA with the IoT are present in the current literature, the proposal is original as interoperability between OPC UA and OCF has not been treated until now. The proposal may also have a potential impact on the current definition of OCF specifications, as it will be pointed out in the paper.

INDEX TERMS Interoperability, Industry 4.0, OPC UA, OCF, IoT.

I. INTRODUCTION

Since few years, Industry has been featured by a revolution, the fourth one, which has been coined with different names in different countries; the most known is Industry 4.0. It features the application of modern Information & Communication Technology (ICT) concepts, such as Internet of Things (IoT) [1], in industrial contexts to create more flexible and innovative products and services leading to new business models and added value [2], [3].

Realisation of this novel vision may be achieved only if a big effort is really put to make interoperable the interchange of information between different industrial applications [4]. In order to provide for interoperability, definition and adoption of communication standards are of paramount importance [5]. For this reason, during the last few years, different organisations have developed reference architectures to align standards in the context of the fourth industrial revolution. One of the main examples is the ''Reference Architecture Model for Industry 4.0 (RAMI 4.0)'' [6]. Among the standards taken into consideration, RAMI 4.0 indicates for the OPC UA international standard (IEC 62541) [7] the role to standardise machine-to-machine communication. OPC UA is mainly based on a Client/Server communication model, although an extension of OPC UA has been very recently released based on Publish/Subscribe pattern [8]. Another example of effort in developing standardisation for Industry 4.0 is the introduction of the Industrial Internet Reference Architecture (IIRA), which is a standards-based open architecture defined by the Industrial Internet Consortium (IIC) [9]. The main goal of IIRA is to create a capability to manage interoperability, map applicable technologies, and guide technology and standards development. Also in this case, OPC UA plays a strategic role as it is one of the core connectivity standards taken into account by IIRA.

Interoperability is an imperative requirement also in the IoT, as no universal language exists for the Internet of Things. To overcome this, industry players have come together to form associations/foundations and consortiums of standards around the various IoT components, including connected buildings, connected home and industry IoT. Open Connectivity Foundation (OCF) is one of the biggest industrial connectivity standards organizations for IoT [10]. Very recently, OCF has defined a set of specifications which leverage existing industry standards and technologies, provide connection mechanisms between devices and between devices and the

cloud, and manage the flow of information among devices, regardless of their form factors, operating systems, service providers or transports. At this moment OCF specifications are under standardisation by ISO/IEC JTC1 Information Technology committee, with the code FDIS 30118.

As said before, one of the main goal of Industry 4.0 is the interoperability of industrial applications and the use of enabling ICT technologies, mainly those based on the IoT. This goal may be reached through integration of communication standards currently present in Industry 4.0 and IoT scenarios. As an example, during these last years, several solutions dealing with the integration of OPC UA and IoT ecosystems appeared in the literature, due to important role played by OPC UA inside the current Industry 4.0 reference models, as pointed out before. Among them, [11] describes a solution for enabling interoperability between OPC UA and DPWS; [12] proposes an OPC UA translator between OPC UA and HTTP, CoAP and MQTT. Another most recent example is represented by the draft version of the mapping between OPC UA and DDS (which is another core standard of the IIRA), defined by [13].

In this paper, the authors proposes a novel solution to make interoperable OPC UA and IoT ecosystems. Among the current IoT ecosystems, OCF has been chosen for the integration with OPC UA, as it seems a promising solution to standardise the exchange of information into IoT. The solution mainly aims to realise a mapping between OPC UA and OCF information models. Through this mapping, information maintained by an OPC UA Server may be used to populate a device compliant to OCF specifications which acts as a server, allowing it to expose this information to whatever client devices in the OCF ecosystem. Vice versa, information maintained by an OCF device may be published by an OPC UA Server allowing to make this information available to whatever OPC UA-compliant device (acting as OPC UA Client and/or OPC UA Subscriber). As previously pointed out, no other solutions of interoperability between OPC UA and OCF are present in the current literature; only papers [14], [15] written by the same authors have been presented at conferences to introduce preliminary and partial results about the research here present. The authors believe that the contribution of the paper is remarkable as it fills the existing lack of proposals about integration between OPC UA and the emerging OCF specifications. Furthermore, the proposal may have a potential impact on the current definition of OCF specifications, as it will be explained in the Section IV.

The paper is organised as it follows. At the beginning some of the basic concepts needed to the reader will be introduced, including an overview of the JSON data format, OPC UA and OCF information models. Then, the proposed mapping between OPC UA and OCF information models will be described in great details. Finally, some remarks will point out future activities and the position of the proposal against the current OCF standardisation activity.

II. TECHNICAL BACKGROUND

The aim of this section is to give to the reader the basic technical concepts needed to understand the content of the paper. In particular, an overview of JSON, OPC UA and OCF will be provided for.

A. OVERVIEW OF JSON

In the last few years, JSON (JavaScript Object Notation) [16], [17] has achieved remarkable popularity as the main format for the representation and the exchange of information over the modern web. JSON defines the following base types:

- *string* is a sequence of Unicode character included between double quotes.
- *number* represents a numerical value.
- *literal names* can assume only the following values: *true*, *false* and *null*. This JSON base type is used to represent both boolean (using *true* and *false*) and null (using *null*) values.
- *object* is a sequence of key/value pairs between curly brackets where key and value are separated by colon and pairs are separated by commas; a key must be a string whilst a value must be of a JSON base type, including object base type itself.
- *array* is an ordered collection of values between square brackets where values are separated by commas. As for JSON object, a value must be of a JSON base type including array base type itself.

Let us consider the JSON document shown by Figure 1. It contains a realistic example of a JSON payload that could be published by an IoT device; in the example, it has been assumed that the device features temperature and humidity sensors.

```
{
  "deviceid" : "myIoT123",
  "temp" : 24.18,
  "humidity" : 72.41,
  "coords" : [37.565262, 15.1063888],
  "lowbattery": true
}
```

FIGURE 1. Example of JSON document.

The JSON document is made up by an object with several key/value pairs. The key ''deviceid'' features a value of JSON string base type containing the identifier of the IoT device. The keys ''temp'' and ''humidity'' feature values represented by JSON number base type relevant to the actual measures of temperature and humidity done by the IoT device. The key ''coords'' contains a value of JSON array base type; the array contains a couple of two values both represented by JSON number base type, giving the coordinates of the current position of the IoT device. Finally, the ''lowbattery'' key features a boolean value represented by a JSON literal names base type indicating a low level of the power supplied by the batteries on board.

B. OVERVIEW OF OPC UA

The aim of this section is to deepen some features of the OPC UA international standard IEC 62541 needed to understand the content of the paper.

OPC UA is mainly based on a Client/Server communication model allowing distribution to OPC UA Clients of information maintained by an OPC UA Server inside an OPC UA AddressSpace. Very recently, an extension of OPC UA called PubSub [8] has been released, enabling distribution of this information also on the basis of on Publish/Subscribe pattern [18].

In the current automation systems, devices from many different manufacturers must be integrated resulting in effort for installation, version management and device operation. This challenge can be faced best with an open and standardised device model. For this reason, *OPC UA for Devices - Companion Specification* [19] defines the *Device Model* intended to provide a unified view of devices irrespective of the underlying device protocols.

In the following, the overview of OPC UA will be limited to the OPC UA AddressSpace and to the Device Model as the relevant knowledge is needed to understand the proposal here presented.

1) OPC UA AddressSpace

Inside an OPC UA Server, OPC UA *Nodes* are used to represent any kind of information. The set of OPC UA Nodes inside an OPC UA Server is called *AddressSpace* [20]. OPC UA Nodes may be organised into different subsets, called *Information Models* [21]; each of them is identified by a unique *Namespace URI*. An array named *NamespaceArray* contains the URIs relevant to the Information Models of an OPC UA AddressSpace; each URI in a NamespaceArray is accessed through an integer index, called *NamespaceIndex (ns)*. Each Node inside an Information Model is univocally identified by an *Identifier (i)*. On the basis of what said so far, a Node inside an OPC UA AddressSpace features an attribute named *NodeId*, which is a couple composed by the NamespaceIndex relevant to the URI of the Information Model to which the Node belongs and by the Identifier of the Node inside the Information Model.

An OPC UA Node belongs to a *NodeClass* which defines the attributes of the OPC UA Node. Each NodeClass is derived from the *Base NodeClass* which defines the common attributes of OPC UA Nodes, among which: *NodeId*, *Description* (which is a textual description of the OPC UA Node), *BrowseName* (used to identify the OPC UA Node when browsing the OPC UA AddressSpace) and *DisplayName* (which contains the name of the OPC UA Node that should be displayed in a user interface). Only for a Node defining types, the boolean attribute *isAbstract* is present. A "true" value means that no instances of the type can be created and the type is called *abstract*; instances may exist only for the relevant subtypes. A type with a "false" value for this attribute is called *concrete*; instances of concrete types can be realised.

OPC UA defines the following NodeClasses:

- *Variable NodeClass* is used to model values of the system. Two types of Variable are defined: *Property* and *DataVariable*. A Property contains information describing particular features of other OPC UA Nodes. A DataVariable represents the data of an OPC UA Object and it may be made up by a collection of other OPC UA DataVariable Nodes. A Variable features, among others, the *Value* and the *DataType* attributes providing for its current value and the relevant type definition, respectively.
- *VariableType NodeClass* is used to provide for type definition of Variables. OPC UA standard defines the *BaseVariableType* which all the VariableTypes must be extended from. OPC UA already defines several standard VariableTypes derived from BaseVariableType. Among them there are the *BaseDataVariableType* and the *PropertyType*. The former is used to define a DataVariable Node, whilst the latter defines a Property Node.
- *DataType NodeClass* is used to provide for type definition of the Value attribute of a Variable Node, as said before.
- *Object NodeClass* is used to represent real-world entities like system components, hardware and software components, or even a whole system. An OPC UA Object is a container for other OPC UA Objects, DataVariables and Methods. As the Object Node does not provide for a value, an OPC UA DataVariable Node can be used to represent the data of an Object.
- *ObjectType NodeClass* is used to hold type definition for OPC UA Objects. OPC UA defines the *BaseObjectType* which all the ObjectTypes must be extended from. Several standard ObjectTypes derived from BaseObjectType already exist inside OPC UA specifications. Among them there is the *FolderType ObjectType* to model hierarchy among OPC UA Nodes. Instances of FolderType ObjectType are used to organise the AddressSpace into a hierarchy of OPC UA Nodes.
- *Method NodeClass* allows to model callable functions that initiate actions within an OPC UA Server. A Method features two boolean attributes named *Executable* and *UserExecutable*, which indicate if the Method is currently executable. Furthermore, a Method features two OPC UA Properties named *InputArguments* and *OutputArguments*, used to specify the input and output arguments of the Method.
- *ReferenceType NodeClass* is used to define different types of References. In the following, description of References will be given.

A relationship may be defined between two OPC UA Nodes and is called *Reference* [7], [20], [21]. References may be classified into: *Hierarchical* and *NonHierarchical*.

The semantic of a Hierarchical Reference is that it spans a hierarchy. It does not forbid loops and does not allow

self-references. OPC UA foresees several Hierarchical References among which:

- *HasComponent:* If the source OPC UA Node is an Object, the target Nodes may be OPC UA Objects, DataVariables and Methods; the meaning is that the source Object is made up by the target OPC UA Nodes. If the source OPC UA Node is a DataVariable, the target Nodes must be other OPC UA DataVariables; the meaning is that the source variable is made up by a set of other variables.
- *HasProperty:* This Reference may connect a source OPC UA Node to an OPC UA Property; the meaning is that the source Node features a property described by the target Node.
- *Organizes:* This Reference may connect a source OPC UA Object of FolderType ObjectType to other OPC UA Objects and/or Variables; the meaning is that the source Node organises (i.e. acts like a folder) the target Nodes.
- *Aggregates:* This is an abstract ReferenceType. The semantic is to indicate that the target Node belongs to the source Node.
- *HasSubtype:* It expresses a subtype relationship of types.

NonHierarchical References do not span a hierarchy and should not be followed when trying to present a hierarchy. Among the NonHierarchical References, there is the *HasTypeDefinition* which is used to bind an OPC UA Object or Variable to its ObjectType or VariableType, respectively.

Another NonHierarchical Reference used in the paper is the *HasModellingRule* Reference which is used to describe how instances of types should be created. The source of this Reference is named *InstanceDeclaration*, whilst its target is a *ModellingRule* Object. An InstanceDeclaration is an Object, Variable or Method that is the target of a Hierarchical Reference starting from an ObjectType or VariableType Node (each of which will be called OPC UA type in the following). A ModellingRule Object specifies what happens to the InstanceDeclaration with respect to instances of the relevant OPC UA type. Several ModellingRules are defined in OPC UA. A *Mandatory* ModellingRule for a specific InstanceDeclaration specifies that instances of the OPC UA type referencing the InstanceDeclaration must have a counterpart of that InstanceDeclaration. This means that each instance must hold an OPC UA Node of the same NodeClass of the InstanceDeclaration; furthermore, the Reference targeting this OPC UA Node must be of the same ReferenceType of the one pointing the InstanceDeclaration. An *Optional* ModellingRule for a specific InstanceDeclaration, instead, specifies that instances of the OPC UA type may have a counterpart of that InstanceDeclaration but it is not required. Mandatory and Optional ModellingRules require that the counterpart of the InstanceDeclaration has the same BrowseName of the InstanceDeclaration. Other two ModellingRule Objects exist named *MandatoryPlaceholder* and *OptionalPlaceholder*. The difference with the previous ones is that the counterparts of InstanceDeclaration may be more than one, regardless of the BrowseName of the InstanceDeclaration.

OPC UA specifications define graphical symbols to represent Nodes and References [20]. Some of them are shown by Table 1 and Table 2, respectively.

TABLE 1. Graphical representation of some OPC UA NodeClasses.

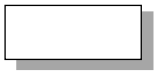
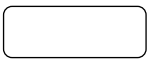
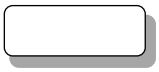
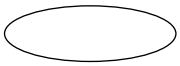
OPC UA Node	Standard graphical representation
ObjectType	
Object	
Variable (DataVariable/Property)	
VariableType	
Method	
ReferenceType	

TABLE 2. Graphical representation of some OPC UA References.

OPC UA Reference	Standard graphical representation
HasTypeDefinition	
Organizes	
HasComponent	
HasProperty	
HasSubtype	

Each graphical representation of OPC UA Node shown in Table 1, contains the relevant DisplayName displayed in the center of the graphical object.

OPC UA allows simplifications in the graphical representation of Nodes and References. For example, let us consider a HasTypeDefinition Reference from an Object or Variable Node towards the relevant OPC UA type. Graphical representation of both HasTypeDefinition Reference and the target OPC UA type may be optionally avoided, adding the name of the OPC UA type to the DisplayName used to label the Object or Variable Node. In this case, the name shall be put in italic on the top of the DisplayName.

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 instead of the DisplayName. The same happens for each counterpart of an InstanceDeclaration relevant to an instance of OPC UA type. In case of 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 2 shows, on the right, an example of the graphical notation just described.

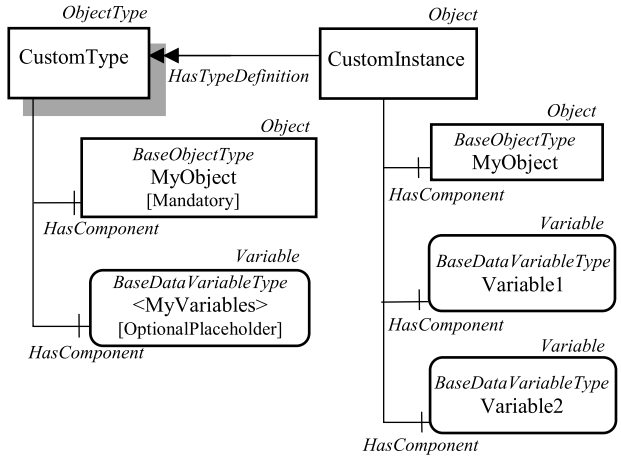


FIGURE 2. Example of graphical representation of InstanceDeclarations, ModellingRule and HasModellingRule References.

An OPC UA ObjectType named *CustomType* is present; it is the source of two HasComponent References targeting the InstanceDeclarations *MyObject* and *<MyVariables>*. In turn, they target a Mandatory and an OptionalPlaceHolder ModellingRule Object, respectively. As said before, these ModellingRule Objects and the relevant HasModellingRule References are not represented, limiting the graphical representation to the texts [Mandatory] and [OptionalPlaceholder] inside the InstanceDeclarations.

In order to better understand the role of the InstanceDeclaration and ModellingRule Objects, Figure 2 shows, on the right, a possible instance of the CustomType ObjectType, called *CustomInstance*. It features a HasComponent Reference targeting an Object which is the counterpart of the InstanceDeclaration with BrowseName MyObject. For this reason the Object is the target of a Reference of the same ReferenceType of the relevant InstanceDeclaration (i.e. HasComponent). Moreover, the graphical representation of the Object features the BrowseName which is the same of the relevant InstanceDeclaration, as required by OPC UA specifications in the case of Mandatory and Optional InstanceDelaration Objects.

CustomInstance Object features two others HasComponent References, used to target counterparts of the InstanceDeclaration <MyVariables>. As this Object points to an OptionalPlaceholder ModellingRule Object, the BrowseNames of the counterparts are not required to be the same of that hold by the InstanceDeclaration; for this reason two different BrowseNames (i.e. Variable1 and Variable2) are shown by the Figure 2.

As said before, the Value attribute of an OPC UA Variable belongs to a certain DataType, defined inside the Variable itself. A DataType may be *Built-in*, *Enumeration* or *Structured*. Array of elements belonging to these DataTypes are allowed for the Value attribute of OPC UA Variable Node.

Built-in provides base types like Int32, Boolean and Double; see [22] for the complete list of the Built-in DataTypes available.

Enumeration represents a discrete set of named values. The Value attribute is of Built-in Int32, i.e. an integer, which allows to identify the enumeration value. Enumeration values (i.e. integer representation of an enumeration) and DisplayName (i.e. human-readable representation of the value of the enumeration) are maintained by the Enumeration DataType.

Structured DataTypes represent structured data allowing to specify user-defined (i.e. vendor-specific) complex types. For each of them, an OPC UA Server provides its structure definition. OPC UA Clients can retrieve this information and use it to decode values belonging to the specific Structured DataType and to encode structured data if they want to write values to the Server. Variables of DataTypeDictionaryType VariableType are present in an OPC UA AddressSpace to contain the descriptions of Structured DataTypes. A Variable of this type contains one or more entries called *StructuredType*. A StructuredType refers to a Structured DataType and contains in turn several *Field* elements. A Field refers to a single component of the Structured DataType and features a *FieldName* and a *TypeName* attributes describing the relevant component.

Figure 3 shows an example of a user-defined Structured DataType named MyType, whose description is maintained inside the MyDictionary Variable of DataTypeDictionaryType. As shown, MyType is a structure having two elements named ''var1'' and ''var2''. The first element is of Built-in DataType (i.e. Int32), and the second one is of a Structured DataType, named ''VarType'', defined again in the same MyDictionary Variable as shown by Figure 3. As it is possible to see, it is made up by two elements: ''var3'' of Int32 Built-in DataType and ''var4'' of String Built-in DataType.

2) OPC UA DEVICE MODEL

The Device Model defines several elements as subtypes of OPC UA BaseObjectType [19]. In the following, only those necessary for the proper understanding of the paper will be deepened.

TopologyElementType ObjectType is abstract and defines the basic information components for all configurable

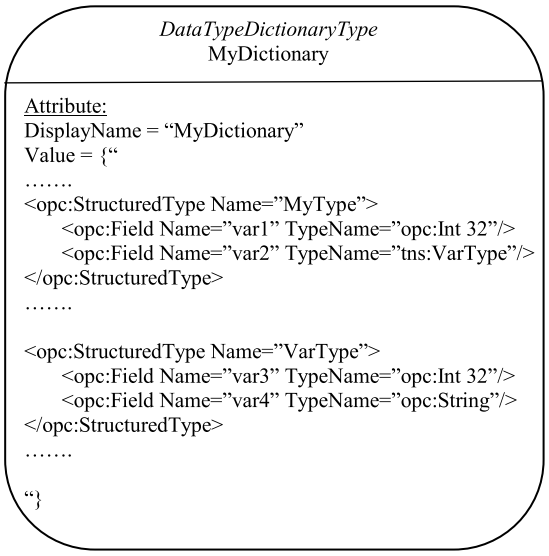


FIGURE 3. Example of a user-defined Structured DataType.

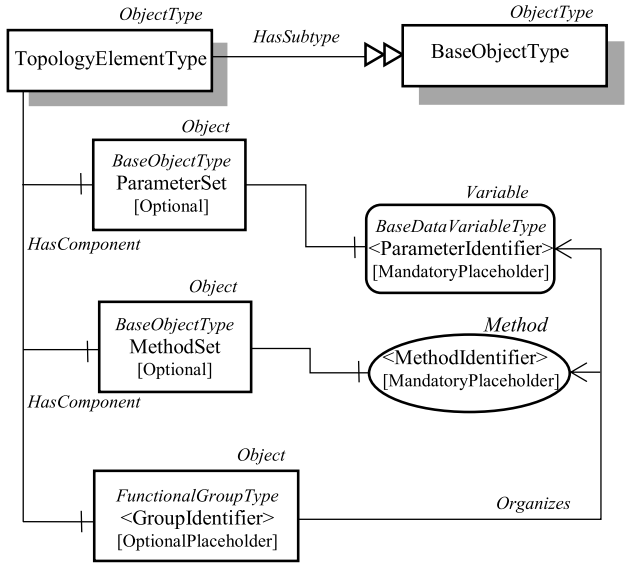


FIGURE 4. TopologyElementType ObjectType.

elements in a device topology. Figure 4 shows some of the TopologyElementType components.

All elements in a topology may have *Parameters* and *Methods*. Parameters are modelled by OPC UA DataVariable Nodes (represented by the InstanceDeclaration *<ParameterIdentifier>*) and are kept as components of an Object called *ParameterSet* as a flat list. Methods (represented by the InstanceDeclaration *<MethodIdentifier>*) are maintained in the same way as components of an Object called *MethodSet*.

FunctionalGroups can be used to organise the Parameters and Methods. A FunctionalGroup Node is an instance of the FunctionalGroupType ObjectType, a subtype of FolderType ObjectType. An instance of TopologyElementType may have an arbitrary number of FunctionalGroups to organise Parameters and Methods.

The abstract *DeviceType* ObjectType provides a general type definition for any Device. It is a subtype of TopologyElementType ObjectType. A Device Node (i.e. an instance of a concrete subtype of DeviceType) may have Parameters, Methods, and FunctionalGroups as defined for the TopologyElementType. DeviceType defines several OPC UA Properties, providing a way for a Client to get common Device information, among which: *SoftwareRevision* (which provides the revision level of the software/firmware of the device), *Model* (which provides the model name of the Device) and *Manufacturer* (which provides the name of the company that manufactured the device).

ConfigurableObjectType is used as a general means to create modular topology units. OPC UA specification [19] defines a generic pattern to expose and configure components, named *Configurable Component* pattern. This pattern is based on the following principles. A ConfigurableObject shall contain a folder called *SupportedTypes* that references the list of subtypes of BaseObjectType available for configuring components. The types are referenced using Organizes References. The instances of the available types shall be components of the ConfigurableObject (through HasComponent References).

Figure 5 shows the ConfigurableObjectType ObjectType.

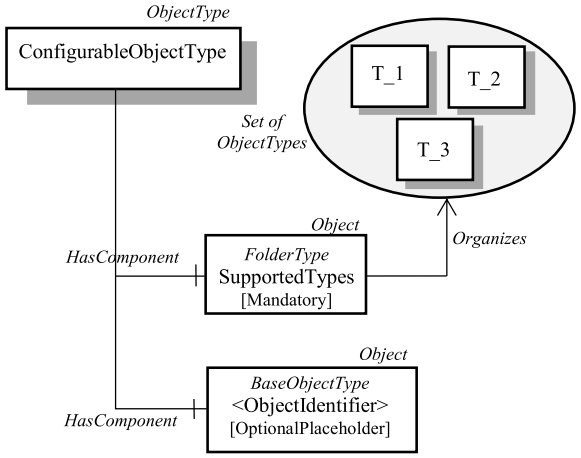


FIGURE 5. ConfigurableObjectType ObjectType.

It points out that ConfigurableObjectType has a Folder named SupportedTypes as a component. This folder organises all the BaseObjectTypes whose instances are the only ones allowed to be components of the ConfigurableObjectType subtype instance. In figure, these instances are modelled by the InstanceDeclaration *<ObjectIdentifier>* with OptionalPlaceholder ModellingRule. Each instance exactly belongs to one of the BaseObjectType organised by the SupportedType Folder (e.g., T_1, T_2 and T_3 shown by the figure).

C. OVERVIEW OF OCF

OCF specifications [23] are based on the REpresentational State Transfer (REST) architecture style [24]. The OCF specifications enable interoperability between heterogeneous devices acting as OCF Clients and Servers. An OCF Server exposes hosted resources, whilst an OCF Client accesses resources on a server through RESTful operations. Data exchange between OCF Clients and Servers occurs in JSON data format.

OCF defines the *OCF Resource Model* to enable the interoperability and to provide consistency between devices in OCF ecosystem [23]. The OCF Resource Model is based on the concepts of *Device* and *Resource*. A Device models a logical entity (e.g., corresponding to a real device) whilst a Resource is the representation of a component of a Device (e.g., a sensor in a smartphone).

An OCF Resource is addressed using URI and is an instance of one or more OCF Resource Types. Each Resource Type defines a set of properties exposed by the Resource. Properties are represented as key/value pairs of a JSON object and are defined using OCF data types derived from JSON base types. According to [25], OCF adopts the same JSON base types described in Section II-A, with the exception of ''true'' and ''false'' values that in OCF are mapped into OCF *boolean* data type (values ''true'' and ''false'' are left unaltered).

The properties of a Resource represent the state of the Resource itself. OCF specification defines several common properties which must be present in a Resource; they are specified by the ''oic.core'' Resource Type. Among them, there is an unique identifier for the Resource in the context of a Device (''id''), the name of the Resource (''n'') and the Resource Types (''rt''). The properties of Resource Type ''oic.core'' must be present in every JSON object representing an OCF Resource.

OCF specifies that a Resource can be related to another Resource through an OCF *Link* [25]. A Link consists of a set of parameters, among which there is the ''href'' parameter which specifies the target URI of the OCF Resource pointed by the Link.

A Resource with properties and Links is named Collection. Properties of an OCF Collection are defined by the Resource Type ''oic.wk.col''. Among these properties, ''links'' is used to gather every Link having the Collection itself as source.

About the Device, OCF mandates a list of core Resources that must be supported and exposed by a Device. Specifically, OCF defines three well-known Resources in an OCF Device. These Resources are addressed in the context of the OCF Device using the predefined URIs ''/oic/p'', ''/oic/d'' and ''/oic/res'' and belongs to the OCF Resource Types named ''oic.wk.p'', ''oic.wk.d'' and ''oic.wk.res'', respectively.

The Resource addressed by ''/oic/p'' URI represents the physical platform hosting the physical device. It is used to expose information about platform like vendor name or software version.

The Resource addressed by ''/oic/d'' URI represents the device and its properties. Properties of this Resource are defined by the Resource Type ''oic.wk.d''. These properties provide information about the devices, among which: a Localized Description of the device in one or more language, the Software Version, the Manufacturer Name and the Model Number.

The Resource addressed by ''/oic/res'' URI is the entry point for all the Resources exposed by the OCF Device. It contains OCF Links to each Resource owned by the Device.

According to OCF specifications, a Device belongs to a *Device Type*. A Device Type is identified by a string; a Device Name is associated to a Device Type for informative purpose too. A Device Type specifies a list of minimum OCF Resources that a Device of this type must expose; a Device may include other Resources, but those ones specified by its Device Type specification shall be present. Device Type does not include the three core resources described before, as they must be always present as said.

An OCF Device can represent a device made up by subdevices. In this case, an OCF Device can expose Resources representing the subdevices. A Resource of this kind belongs to a Device Type and shall expose the properties defined by ''oic.wk.d'' Resource Type. Furthermore, if the Resource representing a subdevice is also a Collection, it shall link mandatory Resources specified by the Device Type.

III. PROPOSAL OF MAPPING BETWEEN OPC UA AND OCF

Integration of applications belonging to different ecosystems (e.g., featuring different information models, communication protocols and services), may be realised in several ways achieving different levels of interoperability according to the main features of the integration itself [26].

This paper present an interoperability proposal between OPC UA and OCF, mainly based on the definition of rules allowing to map each element of the OPC UA AddressSpace in a corresponding element of the OCF Resource Model and vice versa. Definition of mapping rules involved the definition of novel models inside OPC UA AddressSpace and OCF Resource Model in order to allow the realisation of the exact correspondence of each element of an information model into the other, in case the native structure did not allow it.

Considering the interoperability from OPC UA to OCF, the proposal is based on the ad-hoc definition of a new OCF Device Type, called in the paper ''x.opc.device'', containing OCF Resources of novel ad-hoc defined OCF Resource Types. Information maintained by the AddressSpace of an OPC UA Server may be mapped into corresponding elements of an OCF Device of ''x.opc.device'' type, which may expose them to whatever client device in the OCF ecosystem. In this way, information maintained by an OPC UA Server may be made available to IoT devices compliant with OCF specifications.

Considering the interoperability from OCF to OPC UA, the solution proposed aims to realise a mapping from OCF Resource Model to OPC UA AddressSpace. 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. In order to realise this mapping, an extension of the existing OPC UA Information Model has been done. Through the proposal, information maintained by a generic OCF Device may be published by an OPC UA Server making this information available to whatever OPC UA Client connected with the OPC UA Server. According to the recent PubSub specification [8], the OPC UA Server may act also as a Publisher allowing each OPC UA Subscriber to receive information coming from OCF ecosystem.

Figure 6 gives a graphical representation of the proposed solution.

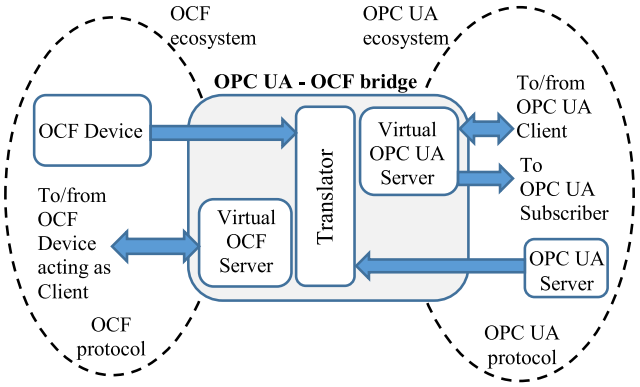


FIGURE 6. Mapping between OPC UA and OCF ecosystems.

As it can be seen, the figure shows the OPC UA and OCF ecosystems. Between them there is an element called *OPC UA-OCF bridge*. It offers a Server for each of the two ecosystems.

The *Virtual OCF Server* is an OCF Device of "x.opc.device" type, described before; as told, it aims to expose information coming from OPC UA Server to client devices in the OCF ecosystem. Mapping from OPC UA may involve the entire OPC UA AddressSpace of a specific OPC UA Server or its subset.

The *Virtual OPC UA Server* is an OPC UA Server exposing the information model defined in the research here presented. As said, this novel information model allows to map elements of the OCF Resource Model of a generic OCF Device in the corresponding elements of the OPC UA AddressSpace of the Virtual OPC UA Server. This last exposes information coming from OCF to both OPC UA Clients and Subscribers in the OPC UA ecosystem.

An important component of the OPC UA-OCF bridge is the *Translator*, shown in the figure. It is in charge to realise the mapping rules defined in the research carried out by the authors and presented in the remainder of this paper. The mapping rules specify how each element of the OPC UA AddressSpace is mapped in the corresponding element of the OCF Resource Model inside the Virtual OCF Sever; in the opposite direction, the mapping rules specify how each element of the OCF Resource Model of a generic OCF Device is mapped in the corresponding element of the information model of the Virtual OPC UA Server.

It is important to point out that the proposal here presented involves only the definition of the mapping rules between OPC UA and OCF and the relevant information models needed for the mapping. Implementations of the mapping (i.e. the Translator, Virtual OPC UA Server and Virtual OCF Server) has been considered outside the scope of the paper.

In order to better understand the advantages of the proposed interoperability solution, the following realistic application scenarios may be considered.

A typical application in an OPC UA ecosystem is a SCADA (Supervisory Control and Data Acquisition) system; it is generally based on an OPC UA Client which exchanges information with one or more OPC UA Servers. According to the proposed mapping solution and considering Figure 6, a SCADA application may be the client of the Virtual OPC UA Server; in this way, the SCADA system may collect information coming from the OCF ecosystem (e.g., sensors, actuators present in home and factory automation scenarios). Moreover, the SCADA could also send commands to OCF devices via the Virtual OCF Server.

A typical application of factory, home and building automation present in an OCF ecosystem is represented by an OCF Device that performs functions of controller and/or data analysis. Based on the mapping here presented, it can receive information maintained by an OPC UA Server through the Virtual OCF Server, as shown in the figure. This information may be used both for the control and the local data analytics together with the information coming from the OCF ecosystem. Finally, an OCF Device acting as controller may send commands to OPC UA ecosystem through the Virtual OPC UA Server.

The following two subsections will describe in details the information models defined in OPC UA and OCF to support the proposed mapping. Description of the information models will include the mapping rules defined in the research carried out.

A. MODELS IN OCF TO SUPPORT THE MAPPING

The proposal is based on the ad-hoc definition of the "x.opc.device" Device Type and three OCF Resource Types, called "x.opc.object", "x.opc.datavariable" and "x.opc.method", described in the following subsections.

1) "X.OPC.DEVICE" DEVICE TYPE

The "x.opc.device" Device Type has been defined to represent the environment where the elements of OPC UA AddressSpace (or its subset) are mapped. A Device of this type shall expose a mandatory OCF Resource of the "x.opc.object" Resource Type named "AddressSpaceSubset". It aims to aggregate the OCF Resources described in the following subsections, mapping the OPC UA Nodes.

2) "X.OPC.OBJECT" RESOURCE TYPE

The "x.opc.object" Resource Type has been defined to map OPC UA Object Nodes of any ObjectType (including FolderType). As explained in Section II-C, every OCF Resource shall implement the common properties defined by the "oic.core" Resource Type. In the case of the "x.opc.object" Resource Type, the property "n" is filled using the OPC UA attribute DisplayName; the Resource Type ("rt") is filled with the name of the Resource Type used to represent the Resource (i.e. "x.opc.object"). The identifier of the Resource ("id") is filled using a string representation of the OPC UA NodeId of the mapped Object Node, made up by the concatenation of the values ns (NamespaceIndex) and i (Identifier) separated by a dash; for example, the "id" relevant to the NodeId made up by ns=2 and i=12, becomes "2-12".

In addition to the OCF common properties, a Resource belonging to the "x.opc.object" Resource Type exposes several properties, described in the following.

A property named "OPCNodeType" is defined as a string and it can assume the values "Object" or "Folder", according if the relevant OPC UA Object Node belongs to the ObjectType or to the FolderType, respectively.

A string property named "OPCDescription" is defined in order to map the Description attribute of the OPC UA Object Node represented.

The OPC UA Object Node represented by the "x.opc.object" Resource Type may be the source of OPC UA Organizes and HasComponent References targeting other OPC UA Nodes. For this reason, another property, named "links", has been considered for the "x.opc.object" Resource Type. This property is an array containing OCF Links mapping OPC UA Organizes and HasComponent References starting from the OPC UA Object Node represented by the OCF Resource. For each Link, the "href" property is filled with the URI of the OCF Resource modelling the OPC UA Node pointed by the OPC UA Reference.

The OPC UA Object Node represented by the "x.opc.object" Resource Type may be the source of OPC UA HasProperty Reference targeting OPC UA Property Nodes. A property named "OPCProperties" is defined in order to represent the OPC UA Property Nodes which may be connected to the represented OPC UA Object Node by HasProperty References. The property is defined as an array of JSON objects. For each HasProperty Reference, a JSON object mapping the target OPC UA Property Node is created and inserted in the JSON array. In the following such a JSON object will be referred as *OPCPropertyObject*.

OPCPropertyObject is made up by several properties. The first mandatory one is named "opc-property-name" and is filled using the BrowseName attribute of the OPC UA Property Node, since this attribute is very suitable to define the semantic of the property. The second mandatory property "value-type" specifies the OCF data type used to represent the Value attribute of OPC UA Property Node; subsection III-A.5) will point out how this representation is realised. The last mandatory property is "opc-property-value", which contains the representation of the Value attribute of the OPC UA Property Node; subsection III-A.5) will point out how this representation is realised. If the Value attribute of the OPC UA Property Node is of array type, the OCF data type of each element is specified by the content of the optional "innermost-type" property; another optional property, "num-dimensions", specifies the relevant dimensions. If the Value attribute of the OPC UA Property Node is an enumeration, the optional property "enum-values" is present and is filled with an array of JSON objects each containing the enumeration values; representation of each value of this array will be described in subsection III-A.5).

3) "X.OPC.DATAVARIABLE" RESOURCE TYPE

OPC UA DataVariable Node is mapped as an instance of the "x.opc.datavariable" OCF Resource Type, which is defined in this proposal.

Common properties of the "oic.core" Resource Type are filled as explained for "x.opc.object". The property "links" is also present, to represent the HasComponent References starting from the modelled OPC UA DataVariable Node.

The OCF data type used for the representation of the Value attribute of the OPC UA DataVariable Node is specified in the "value-type" property. A property named "OPCValue" will contain the representation of the Value attribute according to the OCF data type said before. As said before, subsection III-A.5) will point out how the mapping of data type is realised and how the representation of the Value attribute is realised.

The same properties "innermost-type", "num-dimension", "enum-values" and "OPCProperties" described for "x.opc.object" Resource Type, have been defined also for the "x.opc.datavariable" Resource Type.

4) "X.OPC.METHODS" RESOURCE TYPE

The concept of method does not exist in OCF. In OCF everything is intended as an interaction with the state of an OCF Resource, performing both reading and/or writing operations. For this reason, mapping of OPC UA Method Nodes has been limited to information about their existence and their relevant properties (among which input/output parameters). The authors have defined the "x.opc.method" Resource Type to represent a method as an OCF Collection Resource.

Common properties and "links" are filled as explained for "x.opc.object". Properties named "OPCInputArg" and "OPCOutputArg" are defined in order to represent the InputArgument and OutputArgument Property Nodes relevant to the OPC UA Method Node, respectively.

A boolean property named "Executable" is used to specify whether the OPC UA Method can be invoked by an OPC UA Client; the value assumed by this property is based on the current value of the Executable and UserExecutable attributes of the OPC UA Method Node. As said at the beginning of this subsection, this property does not mean that the method is

executable in the OCF ecosystem; it allows only to highlight if the Method is currently callable inside OPC UA environment.

As done for "x.opc.object" and "x.opc.datavariable", a property named "OPCProperties" is defined in order to represent the OPC UA Property Nodes linked to the OPC UA Method Node by HasProperty References.

5) MAPPING OPC UA VARIABLE NODES

Two important issues were left unsolved in the previous subsections. The first one is about how the OPC UA DataType relevant to the Value attribute of an OPC UA Variable Node is mapped into an OCF data type. The second issue is the representation of the Value attribute according to the OCF data type just found.

It has been assumed that mapping of OPC UA Data Type is done by a two-steps process. At the first step, OPC UA DataType is converted into a JSON base type, according to the *JSON DataEncoding* defined by [22]. The two leftmost columns of Table 3 summarise this mapping; for each OPC UA DataType the relevant the JSON base type is given.

TABLE 3. Mappings OPC UA DataTypes into JSON Base and OCF data types.

OPC UA DataType[(1)]	JSON base type	OCF data type
Built-in: *Integer, Float, Double, StatusCode*	number	number
Enumeration	string	string
Built-in: *Boolean*	literal names[(2)]	boolean
Built-in: NodeId, ExpandedNodeId, DiagnosticInfo DataValue, Variant, ExtentionObject, LocalizedText, QualifiedName	object	object
Structured	object	object
Array	array	array

Notes:
(1): for the definition of OPC UA DataTypes found in the Table please refer to [20].
(2): mapping involves only 'true' and 'false' values.

At the second step, the JSON base types shown in Table 3 are mapped to the relevant OCF base types according to [25]. The rightmost column of Table 3 shows the final mappings into OCF data types. The data type so achieved is used to fill the "opc-property-name" property of OPCPropertyObject or the "value-type" property "x.opc.datavariable" OCF Resource Type, as said in the previous subsections.

The current value of the Value attribute of OPC UA Variable Node (both Property and DataVariable) is encoded according to the OCF data type, found as just explained. The value so obtained is assigned to the property "opc-property-value" in the case of OPC UA Property Node or to the property "OPCValue" in the case of OPC UA DataVariable Node. Particular cases occur when the Value attribute of OPC UA Variable belongs to OPC UA Enumeration or Structured DataTypes.

In case of Enumeration DataType, the property "enum-values" of OPCPropertyObject and "x.opc.datavariable" is filled with an array of JSON objects containing the enumeration values. Each JSON object in the array is made up by two properties named "enumeration-index" and "enumeration-value"; the first contains the integer representation of the enumeration and the second is the relevant value of the enumeration.

In the case of Structured DataType, an ad-hoc JSON object has been defined to represent the original OPC UA Structured DataType. It is made up by only one property named "fields". This property is an array of JSON objects, each of which models a single OPC UA Field of an OPC UA Structured DataType. In order to clarify better this representation, let us assume to consider an OPC UA Variable Node whose Value attribute is of MyType DataType shown by Figure 3. Moreover, let us assume that the current values of the fields "var1", "var3" and "var4" are 10, 12 and "hello", respectively. In this case, the representation of the Value attribute is shown by Figure 7. The figure points out that the property named "fields" is an array with two elements, one for each of the two OPC UA Fields of the MyData Structured DataType (i.e. "var1" and "var2").

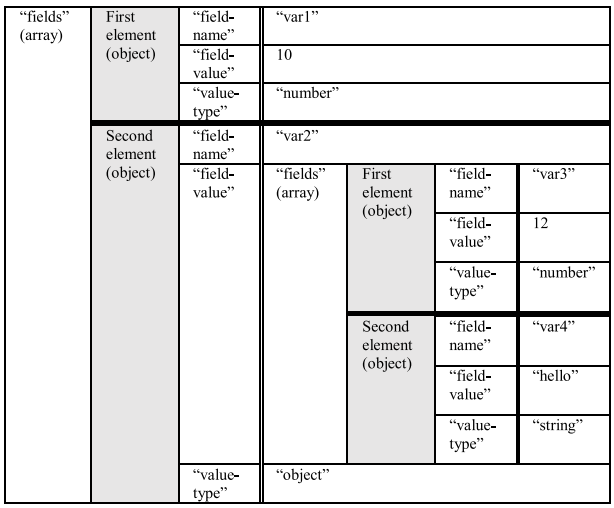


FIGURE 7. Example of representation of Value attribute of MyType Structured DataType.

Each Field is represented as a JSON object containing several properties, some of which are shown by the same figure. For example, considering the OPC UA Field "var1", the property "field-name" assumes the name of the Field. The other property "value-type" is filled with the OCF data type mapping the OPC UA DataType of the Field "var1"; according to Table 3, OPC UA Int32 is mapped into "number". Finally the property "field-value" contains the encoding of the current value of the Field "var1" according to the previous data type; in this case, the value 10 of Int32 type is encoded into the same value according to the number OCF data type.

Considering the Field "var2", its representation is a little bit more complex as "var2" belongs to a Structured Data Type (i.e. VarType, as shown by Figure 3). The Field "var2"

is mapped filling "field-name" with the name of the field (i.e. "var2"), and the "value-type" assumes the value "object" as Table 3 states that an OPC UA Structured Data Type is mapped into an OCF object. The value contained in "field-value" is achieved encoding the current value of the Field "var2" according to the JSON object made up by the only property "fields". As seen before, this property is an array of JSON object, each of which represents a Field of the Structured Data Type "var2". Figure 7 shows the properties of the two JSON objects relevant to the Fields "var3" and "var4".

6) EXAMPLE

The aim of this subsection is to give to the reader an example of the proposed mapping. On the left of Figure 8, a simple subset of an OPC UA AddressSpace is shown.

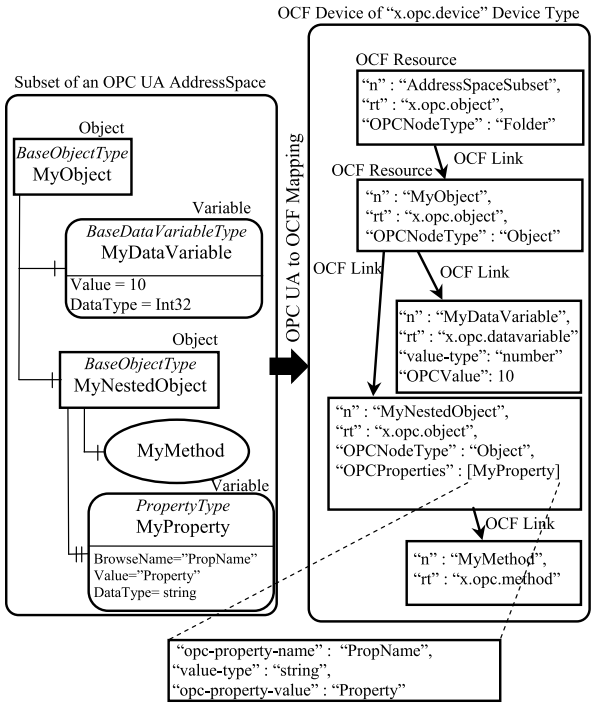


FIGURE 8. Example of the proposed mapping.

It is made up by an OPC UA Object (*MyObject*), which is in turn made up by an OPC UA DataVariable (*MyDataVariable*) and by an OPC UA Object (*MyNestedObject*). This last OPC UA Node features an OPC UA Property Node (*MyProperty*) and is made up by an OPC UA Method (*MyMethod*). Only for space reason, the figure shows a limited number of properties for some OPC UA Nodes.

The subset of OPC UA AddressSpace shown on the left of Figure 8 is mapped into OCF by the OCF Device of "x.opc.device" Device Type present on the right side. Inside the OCF Device, the OCF Resources and OCF Links used for the mapping are shown. Only some of the properties of OCF Resources are shown using JSON formalism.

The OCF Device has a mandatory Resource, named "AddressSpaceSubset" of "x.opc.object" type. The figure shows the relevant values for the properties "n", "rt" and "OPCNodeType". It has been assumed to set the "OPCNodeType" property to "Folder", as this Resource behaves like an OPC UA Node of FolderType organising several Nodes.

The figure points out the mapping of OPC UA Nodes of BaseObjectType and Method NodeClasses.

Mapping of OPC UA Nodes of Variable NodeClass depends on the kind of Variable. An OPC UA DataVariable Node is mapped into OCF Resource of "x.opc.datavariable" Resource Type, thus its "rt" property is set to "x.opc.datavariable". The figure shows the mapping of MyDataVariable Node, pointing out the representation of the attributes Value and DataType. An OPC UA Property Node is not directly mapped into an OCF Resource but it is represented through a property named "OPCProperties" inside the OCF Resource representing the OPC UA Node to which the OPC UA Property belongs. Let us consider the OCF Resource modelling the OPC UA MyNestedObject (which features the property described by MyProperty Node). As it is possible to see, the "OPCProperties" property contains an array made up by only one object called MyProperty relevant to the OPC UA Property Node said before. Some of the relevant properties of this object are shown in the same figure.

B. MODELS IN OPC UA TO SUPPORT THE MAPPING

As said in Section II-C, the OCF Resource Model is based on the concept of OCF Device. Representation of physical entities like devices is present also in OPC UA as defined in the OPC UA Device Model specification [19]. For this reason, this last has been considered as the starting point for the definition of a novel information model, called *OCF OPC UA Information Model*, to represent OCF Resource Model. It is built on top of standard OPC UA Information Model and OPC UA Device Model as graphically represented by Figure 9.

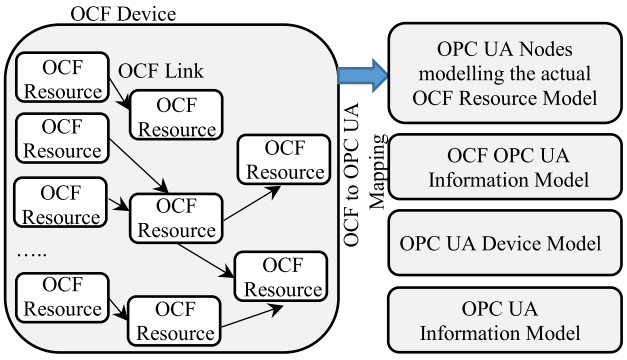


FIGURE 9. OCF OPC UA Information Model.

The proposed OCF OPC UA Information Model offers several novel OPC UA ObjectTypes defined to map elements of OCF Resource Model (e.g., Device Type, Resource Type, Resource). Most of the novel ObjectTypes inherit from the

types of the OPC UA Device Model. The highest level represented in Figure 9 represents the set of OPC UA Nodes, instances of the novel ObjectTypes, used to map the actual OCF elements of the OCF Resource Model, e.g., the OCF Device and the OCF Resources there contained.

The novel ObjectTypes here defined are: *OCFResourceType*, *OCFResourceInstanceType* and *OCFDeviceType*, described in the next subsections.

1) OCFResourceType ObjectType

The *OCFResourceType* ObjectType has been defined with the aim to represent an OCF Resource Type in OPC UA. OCFResourceType is abstract: this means that an ObjectType extending it shall be created for each OCF Resource Type to be represented.

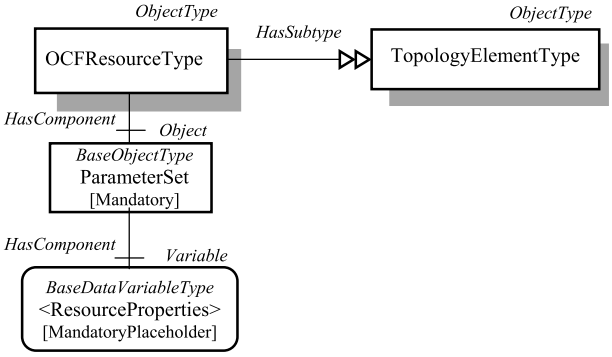


FIGURE 10. Structure of OCFResourceType.

Figure 10 shows the structure of the OCFResourceType. As it can be seen, it is a subtype of TopologyElementType ObjectType and, for this reason, it inherits each component of this last type, among which *ParameterSet* Object. This last is defined as an InstanceDeclaration with a Mandatory ModellingRule Object as shown by the figure. It has been assumed that all the properties defined by the OCF Resource Type to be represented, are mapped as Parameters and grouped by the ParameterSet Object, using OPC UA DataVariable Nodes. Figure 10 shows the InstanceDeclaration relevant to these Parameters; the ModellingRule Object associated to the InstanceDeclaration shall be Mandatory or Optional according on whether the property in the Resource Type specification is mandatory or not, respectively.

For example, let us consider the OCF Resource Type ''oic.r.light.brightness'' featuring only one mandatory property of integer type, named ''brightness''. This OCF Resource Type is mapped in OPC UA using a subtype of the OCFResourceType ObjectType called, in this example, *Light.BrightenessType*.

Figure 11 shows in details this ObjectType. As it can be seen, the ParameterSet Object contains a Parameter aimed to represent the ''brightness'' property defined by the ''oic.r.light.brightness'' OCF Resource Type. This Parameter is realised by the InstanceDeclaration *brightness* featuring

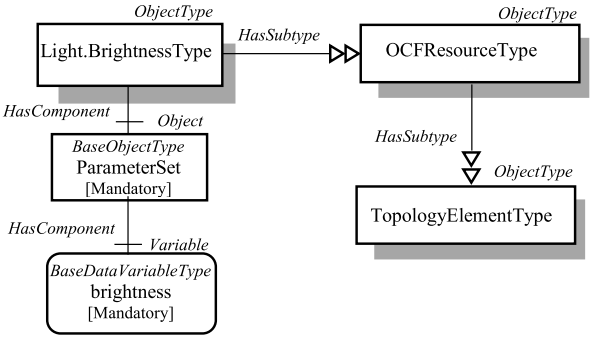


FIGURE 11. Light.BrightnessType ObjectType.

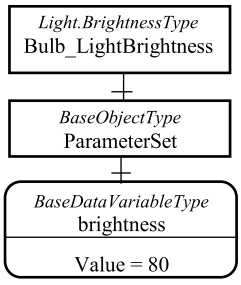


FIGURE 12. Instance of Light.BrightnessType.

a Mandatory ModellingRule Object as the ''brightness'' property is mandatory, as said before.

An OCF Resource features the properties relevant to its OCF Resource Types. In order to represent the actual values of these properties for a specific OCF Resource, an instance of each OCF Resource Type is needed. Figure 12 shows an example of instance of the Light.BrightnessType ObjectType, called Bulb_LightBrightness. As shown, the Value attribute of the brighteness DataVariable Node contains the actual value of the ''brightness'' property related to the OCF Resource to be represented.

2) OCFResourceInstanceType ObjectType

The aim of this subsection is to point out how an OCF Resource is modelled into OPC UA.

As an OCF Resource can be an instance of one or more OCF Resource Types, the obvious mapping would be consisting of defining an OPC UA ObjectType subtype of OCFResourceType for each of its OCF Resource Type and create a unique OPC UA Object instance of all these ObjectTypes, modelling the OCF Resource. Unfortunately, this solution cannot be realised as in OPC UA multiple inheritance is forbidden. For this reason, a different solution has been defined.

The solution adopted in this proposal is the definition of a concrete OPC UA ObjectType, called *OCFResourceInstanceType*. For each OCF Resource, an instance of OCFResourceInstanceType is created to model the OCF Resource. This instance must be able to realise a double aggregation, as explained in the following.

The first aggregation involves all the OCFResourceType subtypes modelling the OCF Resource Types relevant to the OCF Resource. In this way, information of the full set of OCF Resource Types from which the OCF Resource inherits, can be maintained in OPC UA.

In the previous subsection, it has been pointed out that the actual values of the properties relevant to an OCF Resource may be represented using instances of each OCFResourceType subtype modelling the OCF Resource Types from which the OCF Resource inherits (see the example shown in Figure 12). For this reason, an aggregation of all these instances is also needed to represent the actual values of the entire set of properties of an OCF Resource.

The required double aggregation just pointed out, is realised using the Configurable Component pattern defined in [19] as explained in the following. The instance of OCFResourceInstanceType created for each OCF Resource, contains an OPC UA Object of ConfigurableObjectType ObjectType, named in this paper *Aspects*. In turn, Aspects contains an instance of each OCFResourceType subtypes modelling the OCF Resource Types from which the OCF Resource inherits. Finally, due to the features of the ConfigurableObjectType, Aspects owns a folder named *SupportedTypes*; it is used to organise the subtypes of OCFResourceTypes allowed as component of the Aspects Object. Figure 13 shows the details of the OCFResourceInstanceType ObjectType.

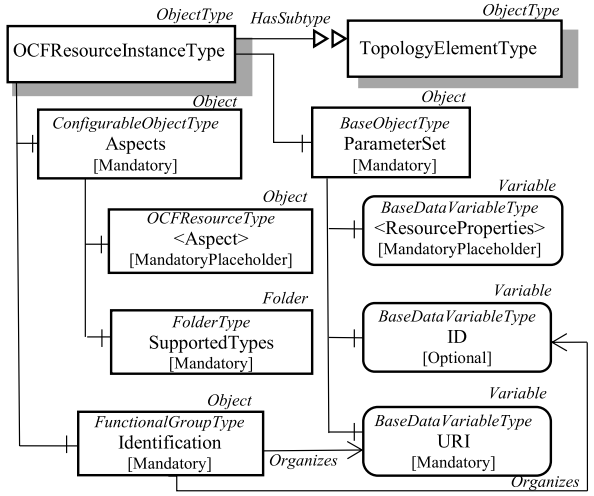


FIGURE 13. Details of OCFResourceInstanceType ObjectType.

The figure points out that an instance of this ObjectType is made up by several components. One of them is the Aspects Object. Aspects has a folder named SupportedTypes as component; it is used to organise the subtypes of OCFResourceTypes relevant to the OCF Resource to be represented. Another component of OCFResourceInstanceType is ParameterSet Object, inherited from TopologyElementType. All the Parameters of every component of Aspects will be grouped by the ParameterSet Object. This grouping allows to easily access all the OCF properties featured by the OCF Resource. ParameterSet groups also other Parameters, among which Figure 13 shows URI (that is mandatory and is used to map the URI of the OCF Resource represented) and ID (that is optional and is used to map the "id" common property of the OCF Resource state). Since URI and ID identify the OCF Resource, they shall be grouped by the FunctionalGroup called *Identification* as explained in Section II-B).

```
{
  "n": "bulb",
  "rt": ["oic.r.switch.binary","oic.r.light.brightness"],
  "value": true,
  "brightness": 80
}
```

FIGURE 14. State of an OCF Resource representing a bulb.

In the following, a simple example will be presented to better understand the mapping just explained. Let us consider an OCF Resource representing a real bulb, addressed by the URI "/a/bulb". The state of this OCF Resource is shown by Figure 14 by JSON formalism. As specified by the "rt" property, the OCF Resource is instance of two OCF Resource Types called "oic.r.switch.binary" and "oic.r.light.brightness". The former features a mandatory boolean property named "value" indicating whether the bulb is on or off. The latter, as already described before, features only one mandatory property of integer type, named "brightness". Both "value" and "brightness" properties are part of the state of the OCF Resource shown by Figure 14 which shows their actual values.

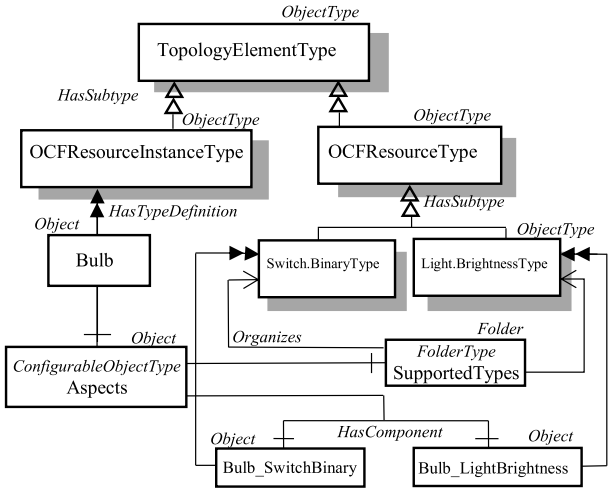


FIGURE 15. Example of OCFResourceInstanceType ObjectType.

Figure 15 shows the mapping of the OCF Resource described by Figure 14. For the sake of clarity, this mapping is limited to some of the components shown by Figure 13. The OPC UA Object Bulb is an instance of OCFResourceInstanceType and models the OCF Resource representing the bulb. Furthermore, the Switch.BinaryType and Light.BrightnessType ObjectType are subtypes of

OCFResourceType and model the OCF Resource Types ''oic.r.switch.binary'' and ''oic.r.light.brightness'', respectively.

The OPC UA Object Bulb contains the OPC UA Object Aspects. In turn this last Object has the Folder Node SupportedTypes as component. It is able to aggregate the OPC UA ObjectTypes modelling the OCF Resource Type, i.e. Switch.BinaryType and Light.BrightnessType ObjectType, which are target of Organizes References starting from SupportedTypes Node. Aspects contains also an instance of each OCFResourceType subtypes modelling the OCF Resource Types relevant to the OCF Resource, i.e. Bulb_SwitchBinary and Bulb_LightBrightness. These instances contain the actual values of the properties of the OCF Resource (i.e. ''value'': true, ''brightness'': 80).

3) OCFDeviceType ObjectType

OCFDeviceType ObjectType is an abstract ObjectType subtype of OPC UA DeviceType. A subtype of OCFDeviceType ObjectType shall be created for each OCF Device Type; an instance of such subtype maps an OCF Device and the information it gathers. OCFDeviceType is graphically described by Figure 16.a.

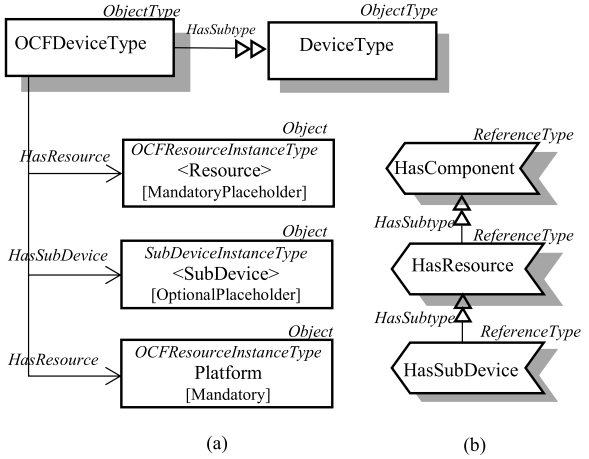


FIGURE 16. (a) OCFDeviceType ObjectType, (b) HasResource and HasSubDevice ResourceTypes details.

As explained in Section II-C, an OCF Device must expose OCF Resources and, optionally, subdevices. It has been assumed that mapping of the relationships between an OCF Device and its Resources is achieved through the use of an ad-hoc defined OPC UA ReferenceType, named *HasResource*. Relationship with the subdevices is modelled by another ad-hoc defined Reference called *HasSubDevices*. HasSubDevice is subtype of HasResource which in turn is subtype of HasComponent ReferenceType. The definition of these ReferenceTypes, made by the authors, is shown by Figure 16.b.

As shown by Figure 16.a, OCFDeviceType is the source of a HasResource Reference targeting the InstanceDeclaration named *<Resource>*. It features a MandatoryPlaceholder ModellingRule Object. This InstanceDeclaration is needed in order to represent OCF Resources contained in an OCF Device.

OCFDeviceType is also the source of a HasSubDevice Reference targeting an InstanceDeclaration named <SubDevice>; as shown, it is linked to an OptionalPlaceholder ModellingRule Object. InstanceDeclaration is an Object of an ad-hoc defined ObjectType, named SubDeviceInstanceType; it allows to model subdevices of the OCF Device.

Among the Resources exposed by an OCF Device, the three ones addressed by the URIs ''/oic/p'', ''/oic/res'' and ''/oic/d'' are mandatory, as said in Section II-C.

The OCF Resource addressed by ''/oic/p'' is mapped as an instance of OCFResourceInstanceType, named *Platform*; this explains the presence in Figure 16.a of the InstanceDeclaration name *Platform* to which a Mandatory ModellingRule Object is associated. Platform Object is made up by components able to map the properties of the OCF Resource addressed by ''/oic/p''.

The OCF Resource addressed by ''/oic/res'' provides the list of OCF Links pointing the OCF Resources exposed by an OCF Device. It has been assumed to avoid the use of an OPC UA Node to represent the OCF Resource addressed by ''/oic/res'' and to map only the OCF Resources linked. These Resources are mapped by the InstanceDeclaration named <Resource> in Figure 16.a, as said before.

OCF Resource addressed by the URI ''/oic/d'' is used to provide information about the relevant OCF Device through its properties (defined by ''oic.wk.d'' Resource Type). Also in this case, mapping by means of an OPC UA Node has been avoided. Instead, the properties of this OCF Resource are mapped by both OPC UA Properties (inherited by OPC UA DeviceType) and OPC UA Node Attributes of the instance of an OCFDeviceType subtype, as specified by Table 4.

TABLE 4. Mapping rules for ''oic.wk.d'' resource type.

''/oic/d'' properties defined by ''oic.wk.d''	OPC UA Property of OCFDeviceType	OPC UA Attribute of OCFDeviceType
Name	-	DisplayName
Localized Description	-	Description
Software Version	SoftwareRevision	-
Manufacturer Name	Manufacturer	-
Model Number	Model	-

4) EXAMPLE

The aim of this subsection is to provide an example of the mapping from OCF to OPC UA. The example is based on the mapping of an OCF Device belonging to the OCF Device Type ''x.customdevice'' containing the OCF Resource specified by Figure 14.

A new subtype of OCFDeviceType ObjectType, named *CustomDeviceType*, has been defined to represent the OCF Device Type ''x.customdevice''. An instance of CustomDeviceType is then created to model the OCF Device. Figure 17 shows the OPC UA representation of this OCF Device, named *CustomDevice*.

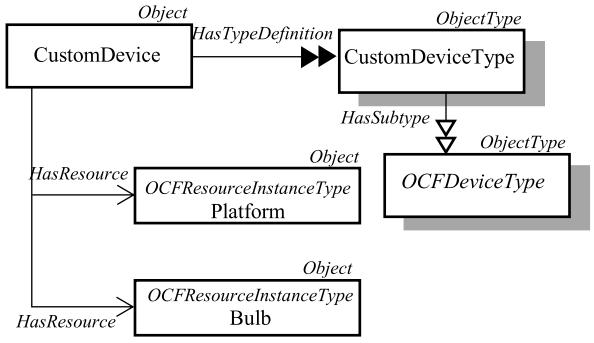


FIGURE 17. Mapping of the OCF Device considered in the example.

The CustomDevice Object features two HasResource References: the first points to the Platform Object, representing the OCF Resource addressed by ''/oic/p''. The second Reference points to the representation of the OCF Resource of Figure 14, which is shown by Figure 18.

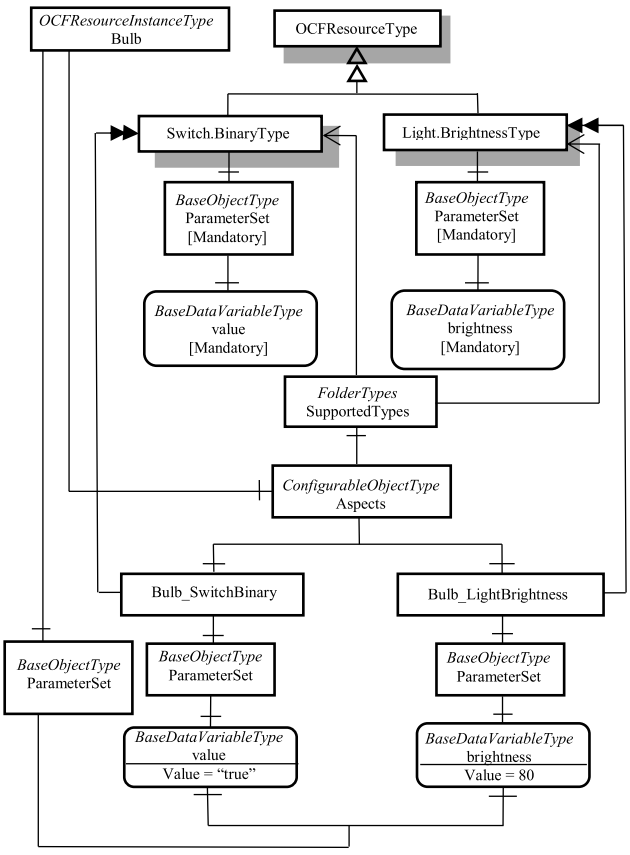


FIGURE 18. Mapping of the OCF Resource shown by Figure 14.

IV. FINAL REMARKS

The paper has presented a mapping solution between the OPC UA and the OCF information models. The main advantage of the proposal is the capability to enhance the interoperability of OPC UA in the IoT domain. The paper is original as the issue has not been treated until now, due to the very recent definition of the OCF specifications.

As said in the paper, the mapping solution has been limited to the definition of novel information models in both OPC UA and OCF standards and rules able to map each information maintained in one ecosystem into the other and vice versa. Implementation of these mapping rules has been considered out of the scope of the paper. In a next future, the authors plan to realise a software platform able to realise the interoperability between the OPC UA and OCF ecosystems, including the implementation of the OPC UA-OCF bridge components shown by Figure 6. In the meantime, a GitHub repository has been realised by the authors at the address *https://github.com/OPCUAUniCT* with the name *OPCUA-OCF-Information-Models-Mapping*; the definition of the entire set of information models presented in the paper is there available.

Beside the capability of the proposed research to enable interoperability between OPC UA and OCF, the authors believe that it may have an impact also on the current standardisation activities of OCF, as explained in the following. Among the current specifications, [27] defines a particular OCF device (called OCF Bridge Device) aimed to realise interoperability between OCF devices and devices using protocols different from OCF. At the moment, the bridge device does not allow interoperability between OCF and OPC UA. The general architecture of the OCF Bridge Device is quite similar to that featured by the OPC UA-OCF bridge shown by Figure 6, including all the relevant elements (i.e. Virtual Servers and Translator). For this reason, the mapping ideas exposed in the paper may be used as a valid contribution to support the OCF standardisation activity concerning the definition of the bridging between OPC UA and OCF in both directions.

REFERENCES

[1] T. Guarda *et al.*, "Internet of Things challenges," in *Proc. 12th Iberian Conf. Inf. Syst. Technol. (CISTI)*, Jun. 2017, pp. 628–631.
[2] Y. Liao, F. Deschamps, E. F. R. Loures, and L. F. P. Ramos, "Past, present and future of industry 4.0-a systematic literature review and research agenda proposal," *Int. J. Prod. Res.*, vol. 55, no. 12, pp. 3609–3629, 2017.
[3] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: State of the art and future trends," *Int. J. Prod. Res.*, vol. 56, no. 8, pp. 2941–2962, 2018.
[4] S. Weyer, M. Schmitt, M. Ohmer, and D. Gorecky, "Towards industry 4.0–standardization as the crucial challenge for highly modular, multi-vendor production systems," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 579–584, 2015.
[5] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1234–1249, Aug. 2013.
[6] VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik. (2015). *Status Report-Reference Architecture Model Industrie 4.0 (RAMI4.0)*. Accessed: Nov. 9, 2018. [Online]. Available: https://www.zvei.org/en/press-media/publications/gma-status-report-reference-archtitecture-model-industrie-40-rami-40/
[7] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. New York, NY, USA: Springer-Verlag, 2009.
[8] OPCFoundation. (2018). *OPC UA Part 14: PubSub Specification Release 1.04*. Accessed: Nov. 9, 2018. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture
[9] Industrial Internet Consortium. (2017). *The Industrial Internet of Things Volume G1: Reference Architecture (Version 1.80)*. [Online]. Available: https://www.iiconsortium.org/IIC_PUB_G1_V1.80_2017-01-31.pdf

[10] *Open Connectivity Foundation Web Site*. Accessed: Nov. 9, 2018. [Online]. Available: https://openconnectivity.org/
[11] M. J. A. G. Izaguirre, A. Lobov, and J. L. M. Lastra, "OPC-UA and DPWS interoperability for factory floor monitoring using complex event processing," in *Proc. 9th IEEE Int. Conf. Ind. Inform.*, Jul. 2011, pp. 205–211.
[12] H. Derhamy, J. Rönnholm, J. Delsing, J. Eliasson, and J. van Deventer, "Protocol interoperability of OPC UA in service oriented architectures," in *Proc. 15th IEEE Int. Conf. Ind. Inform. (INDIN)*, Jul. 2017, pp. 44–50, doi: 10.1109/INDIN.2017.8104744.
[13] Object Management Group (OMG). *OPC UA/DDS Gateway*. Accessed: Nov. 9, 2018. [Online]. Available: https://www.omg.org/spec/DDS-OPCUA/1.0/Beta1/PDF
[14] S. Cavalieri and M. S. Scroppo, "A proposal to make OCF and OPC UA interoperable," in *Proc. ICIT*, Feb. 2018, pp. 1551–1556, doi: 10.1109/ICIT.2018.8352412.
[15] S. Cavalieri, M. G. Salafia, and M. S. Scroppo, "Mapping OPC UA addressspace to OCF resource model," in *Proc. ICPS*, St. Petersburg, Russia, May 2018, pp. 135–140, doi: 10.1109/ICPHYS.2018.8387649.
[16] Internet Engineering Task Force (IETF). (Mar. 2014). *The JavaScript Object Notation (JSON) Data Interchange Format*. Accessed: Nov. 9, 2018. [Online]. Available: https://tools.ietf.org/html/rfc7159
[17] L. Bassett, *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON*. Newton, MA, USA: O'Reilly Media, 2015.
[18] R. Baldoni, M. Contenti, and A. Virgillito, "The evolution of publish/subscribe communication systems," in *Future Directions in Distributed Computing* (Lecture Notes in Computer Science), vol. 2584. New York, NY, USA: Springer-Verlag, 2003, pp. 137–141.
[19] OPCFoundation. (2013). *OPC UA for Device Companion Specification Release 1.01*. Accessed: Nov. 9, 2018. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture/
[20] OPCFoundation. (2017). *OPC UA Part 3: Address Space Model Specification Release 1.04*. Accessed: Nov. 9, 2018. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture
[21] OPCFoundation. (2017). *OPC UA Part 5: Information Model Release 1.04*. Accessed: Nov. 9, 2018. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture
[22] OPCFoundation. (2017). *OPC UA Part 6: Mappings Release 1.04*. Accessed: Nov. 9, 2018. https://opcfoundation.org/developer-tools/specifications-unified-architecture
[23] Open Connectivity Foundation. *OCF Specifications*. Accessed: Nov. 9, 2018. [Online]. Available: https://openconnectivity.org/developer/specifications
[24] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Inf. Comput. Sci., Univ. California, Irvine, Irvine, CA, USA, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
[25] Open Connectivity Foundation. (Jun. 2018). *OCF Core Specification Version 2.0*. Accessed: Nov. 9, 2018. [Online]. Available: https://openconnectivity.org/specs/OCF_Core_Specification_v2.0.0.pdf
[26] ETSI. *Strategy and Coordination Plan for IoT Interoperability and Standard Approaches, H2020-CREATE-IoT Project, Deliverable 06.01, Revision: 1.00*. Accessed: Nov. 9, 2018. [Online]. Available: https://european-iot-pilots.eu/wp-content/uploads/2017/10/D06_01_WP06_H2020_CREATE-IoT_Final.pdf
[27] Open Connectivity Foundation. (Nov. 2017). *OCF Bridging Specification Version 1.3*. Accessed: Nov. 9, 2018. [Online]. Available: https://openconnectivity.org/specs/OCF_Bridging_Specification_v1.3.0.pdf

SALVATORE CAVALIERI was born in Catania, Italy, in 1965. He received the Laurea degree in electronic engineering, the Ph.D. degree in electronic and computer engineering, and the post-Ph.D. degree in electrical engineering from the University of Catania in 1989, 1993, and 1995, respectively. He is currently a Full Professor in computer engineering with the Department of Electrical Electronic and Computer Engineering, University of Catania. His main research areas are in distributed systems, real-time scheduling, factory automation, and industrial informatics. The main results of the research activities carried out have been presented on over 150 papers published on international journals and conference proceedings. He served as a member of the IEC SC65C WG6 Fieldbus Standardization Committee. He was an Honorary Member of the Fieldbus Foundation Italy and Profibus Network Italy Consortium. Since 2007, he was a Scientific Partner of KNX Organization (www.knx.org). Since 2009, he was a member of OPC Foundation (www.opcfoundation.org).

MARCO GIUSEPPE SALAFIA was born in Catania, Italy, in 1989. He received the B.S. and M.S. degrees in computer engineering from the University of Catania, where he is currently pursuing the Ph.D. degree in systems, energy, computer and telecommunication engineering. From 2016 to 2017, he was a Software Analyst at Micron Technology. His research interest includes the development of interoperability solution for Industry 4.0 and IIoT. In 2016, he was a recipient of a scholarship at Leonardo/Finmeccanica for the Project–Cyber Security Simulation Environment for analysis and implementation of cyber-attack and defense models on infrastractureless communication networks.

MARCO STEFANO SCROPPO was born in Catania, Italy, in 1989. He received the B.S. degree in computer engineering from the University of Catania, Italy, in 2013, and the M.S. degree in computer engineering from the University of Catania, Italy, in 2015, where he is currently pursuing the Ph.D. degree in systems, energy, computer and telecommunications engineering.

His research interest focuses on the enhancement of interoperability in Industry 4.0, IoT, and IIoT. The research aims to study and define interoperability solutions that make smooth integration of IIoT applications and increasing security at the same time.