Towards Integration between OPC UA and OCF

Salvatore Cavalieri, Salvatore Mulè, Marco Giuseppe Salafia and Marco Stefano Scroppo
University of Catania, Department of Electrical Electronic and Computer Engineering (DIEEI), Viale A. Doria 6, 95125 Catania, Italy

Keywords: OPC UA, OCF, Industry 4.0, IoT, Information Model, Integration, Interoperability.

Abstract: The paper deals with Industry 4.0 presenting a solution to improve interoperability between industrial applications and IoT ecosystems. In particular, the proposal aims to reach interoperability between OPC UA and the emerging Open Connectivity Foundation (OCF), through a mapping between the relevant information models. A novel OPC UA information model will be presented as extension of the standard one, in order to allow the mapping of whatever information produced by an OCF device into the information model of an OPC UA server. The paper fills the existing lack of integration between OPC UA and the OCF specifications, as no other solution is present in literature at the moment.

1 INTRODUCTION

Industry 4.0 features the application of modern Information & Communication Technology (ICT) concepts, such as Internet of Things (IoT) (Guarda, et al., 2017), in industrial contexts to create more flexible and innovative products and services leading to new business models and added value (Liao, et al., 2017)(Da Xu, et al., 2018).

One of the main features of the recent industrial revolution is the need to achieve full integration of the industrial applications with the IoT. In this new vision, a traditional SCADA (Supervisory Control and Data Acquisition) system may collect and analyse information coming from IoT devices, for example.

Both industrial and IoT contexts feature the presence of several communication systems and the convergence to a unique communication system seems, at the moment, only a dream.

Among the current communication systems enabling the exchange of information between industrial applications, the international standard OPC UA, IEC 62541, (Mahnke, et al., 2009) plays an important role. This is confirmed by the inclusion of this standard into reference architectures recently defined in the context of the Industry 4.0, e.g., the "Reference Architecture Model for Industry 4.0 - RAMI 4.0" (VDI/VDE, 2015) and the Industrial Internet Reference Architecture (IIRA) defined by the Industrial Internet Consortium (IIC) (Industrial Internet Consortium, 2017).

Existence of several communication solutions is also present in the context of the Internet of Things. Among them, there is that defined by the Open Connectivity Foundation (OCF) (Open Connectivity Foundation Website, 2018). At this moment, OCF specifications are under ISO/IEC standardisation by ISO/IEC JTC1 Information Technology committee (i.e. ISO/IEC DIS 30118).

Integration of industrial applications with the IoT may be achieved through the integration of communication systems existing in the two different environments. For example, during these last years several solutions dealing with the integration of OPC UA and IoT appeared, due to important role played by OPC UA inside the current Industry 4.0 reference models, as pointed out before. Among them, (Izaguirre, et al., 2011) describes a solution for enabling interoperability between OPC UA and DPWS; (Derhamy, et al., 2017) proposes an OPC UA translator between OPC UA and HTTP, CoAP and MQTT.

In this paper, the authors propose another solution towards integration of OPC UA and IoT ecosystems. Among the current communication systems existing in the IoT, OCF has been chosen, as it seems a promising solution to standardise the exchange of information into the IoT.

Integration between OPC UA and OCF here proposed, is realised through a mapping between the relevant information models. Through this mapping, information maintained by an OPC UA Server may be used to populate an OCF device, allowing it to

Cavalieri, S., Mulè, S., Salafia, M. and Scroppo, M.
Towards Integration between OPC UA and OCF.
DOI: 10.5220/0007672205550562
In *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 555-562
ISBN: 978-989-758-372-8

expose this information to whatever OCF devices in the OCF ecosystem. Vice versa, information maintained by an OCF device may be published by an OPC UA Server making this information available to whatever OPC UA–based applications. Mapping from OPC UA to OCF information models has been presented by the authors in previous works (Cavalieri, et al., 2018). In this paper, the mapping from OCF to OPC UA is presented in great details, for the first time. For this reason, the proposal here presented represents an enhancement of the previous works, targeting the full integration between OPC UA and OCF.

No other solutions of integration between OPC UA and OCF are present in the current literature, with the exception of the authors' papers cited before. For this reason, the authors believe that the contribution of the paper is remarkable as it allows the industrial applications based on the international standard OPC UA to interwork with the emerging IoT ecosystem based on the OCF specifications.

2 OPC UA

The aim of this section is to give an overview about two important features of the OPC UA international standard: OPC UA *AddressSpace* and OPC UA *Device Model*.

2.1 OPC UA AddressSpace

In OPC UA, the set of information that a server makes available to OPC UA-based applications is named *AddressSpace*. In particular an AddressSpace contains OPC UA Nodes used to represent any kind of information (Mahnke, et al., 2009)(OPCFoundation, 2015).

Each OPC UA Node belongs to a class named NodeClass, derived from the Base NodeClass, which defines several attributes. Among them, there are: BrowseName (used as a non-localised human-readable name when browsing the AddressSpace), DisplayName (containing the name of the OPC UA Node to be displayed to the user) and Description (explaining the meaning of the Node in a localised text).

Among the available NodeClasses, there is the *Variable* NodeClass, modelling values of the system. Two types of Variables are defined: *Properties* (containing metadata) and *DataVariables* (representing the data of an OPC UA Object). Both of them holds an attribute named *Value*, containing the data. Another NodeClass is the *Method*, modelling callable functions that initiate actions within an OPC UA Server. *Object* NodeClass represents 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, therefore an OPC UA DataVariable Node is used to represent the data of an Object.

OPC UA defines particular NodeClasses defining types. Some of them are called *Concrete*, for which instances can be realised. Other types are *Abstracts*, for which instances may exist only for the relevant subtypes. Among NodeClasses defining types there is the *ObjectType* NodeClass, which holds type definition for OPC UA Objects. Objects are instances of ObjectTypes in the sense that they inherit the Nodes beneath the ObjectTypes defining them. OPC UA defines the *BaseObjectType* which all the ObjectTypes must be extended from. OPC UA already defines several standard ObjectTypes derived from BaseObjectType. Among them there is the *FolderType* whose aim is to model hierarchy among OPC UA Nodes. Instances of FolderType ObjectType are used to organise the AddressSpace into a hierarchy of OPC UA Nodes. *VariableType* is another NodeClass used to provide type definition for Variables. OPC UA defines the *BaseVariableType* which all the VariableTypes must be extended from. Among the standard VariableTypes derived from BaseVariableType, there are the *BaseDataVariableType* and the *PropertyType*. The former is used to define a DataVariable Node, whilst the latter defines a Property Node.

Particular relationships may be defined between OPC UA Nodes; they are called References. The *ReferenceType* NodeClass used to define different types of References. They may be classified in two different main categories: Hierarchical and NonHierarchical.

Among the Hierarchical References there is the *HasComponent* Reference used to specify that an OPC UA Object contains another OPC UA Object, OPC UA DataVariable or OPC UA Method specified as target of the Reference. Another Hierarchical Reference is *Organizes* which allows to organise several OPC UA Nodes under a folder (i.e. a FolderType Object). The last Hierarchical Reference cited in the paper is the Abstract *Aggregates*, which indicates that the target Node belongs to the source Node.

Among the NonHierarchical References there is the *HasTypeDefinition*, which is used to bind an OPC UA Object or Variable to its ObjectType or

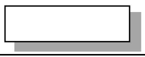
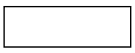
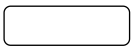
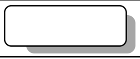
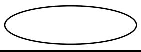
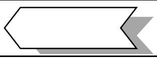
VariableType, respectively. Another NonHiererchical Reference is the *HasSubtype* which expresses a subtype relationship between types.

HasModellingRule Reference is a NonHierarchical Reference used to describe how instances of OPC UA types should be created. The source of this Reference is an *InstanceDeclaration* which may be a Variable, Object or Method. The InstanceDeclaration must be the target Node of a Hierarchical Reference from a Node defying an OPC UA type. HasModellingRule has a ModellingRule Object as target Node.

A ModellingRule Object specifies what happens to the InstanceDeclaration with respect to instances of the OPC UA type to which it is linked by the Hierarchical Reference. A *Mandatory* ModellingRule for a specific InstanceDeclaration specifies that instances of the OPC UA type must have a counterpart of that InstanceDeclaration. An *Optional* ModellingRule, 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*; in this case, the counterparts of InstanceDeclaration may be more than one, regardless of the BrowseName of the InstanceDeclaration.

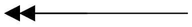
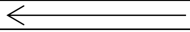
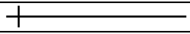
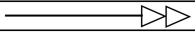
OPC UA specifications define graphical symbols to represent Nodes and References; some of them are shown by Table 1 and 2, respectively.

Table 1: Graphical notation of some OPC UA Nodes.

OPC UA Node	Standard graphical notation
ObjectType	
Object	
DataVariable/Property	
VariableType	
Method	
ReferenceType	

In the standard graphical notation, a ModellingRule Object and relevant HasModellingRule Reference are represented inside the source InstanceDeclaration Node only by the kind of the ModellingRule Object in square brackets (i.e. [Mandatory], [Optional], [OptionalPlaceholder], [MandatoryPlaceholder]). Furthermore, the BrowseName of InstanceDeclarations having the *OptionalPlaceholder* and *MandatoryPlaceholder* ModellingRule are enclosed within angle brackets.

Table 2: Graphical notation of some OPC UA References.

OPC UA Reference	Standard graphical notation
HasTypeDefinition	
Organizes	
HasComponent	
HasSubType	

2.2 OPC UA Device Model

OPC UA specification (OPCFoundation, 2015) is aimed to define the information model associated with devices in automation systems, in order to enhance their integration. The specification defines three models which build upon each other. At the lowest level there is the *Device Model* which aims to provide a unified view of devices irrespective of the underlying device protocols. Several OPC UA Nodes have been defined into the OPC UA Device Model, all derived from the OPC UA *BaseObjectType.*

TopologyElementType is the base *ObjectType* defining the basic information components for all configurable elements in a device topology. All elements in a device topology may have *Parameters* and *Methods*. *Parameters* are modelled with OPC UA DataVariable Nodes. *TopologyElement* Object has two components named *ParameterSet* and *MethodSet*, which are Objects containing a flat list of Parameters and Methods, respectively. *FunctionalGroups* can be used to organise the Parameters and Methods to reflect the structure of the TopologyElement; a TopologyElement may have an arbitrary number of FunctionalGroups. A special FunctionalGroup called *Identification* shall be used to organise Parameters for identification of a TopologyElement. The same Parameter or Method might be referenced from more than one FunctionalGroups.

The *DeviceType ObjectType* provides a general type definition for any Device. *Devices* - in addition to *Parameters* and *Methods* - may support sub-devices.

ConfigurableObjectType is used to expose and configure components, according to the following principles. An Object of ConfigurableObjectType shall contain a folder called *SupportedTypes* that references the list of types available for configuring components using Organizes References. The instances of the available types shall be components of the configurable object (through HasComponent References).

3 OCF RESOURCE MODEL

The OCF Resource Model is based on the concepts of *Device* and *Resource*. According to the OCF Core Specification (Open Connectivity Foundation, 2018), a Device models a logical entity (e.g., corresponding to a real device like a controller) whilst a Resource is the representation of a component of a Device (e.g., a sensor integrated in a controller).

An OCF Resource is an instance of one or more OCF Resource Types. Each Resource Type defines a set of properties exposed by the Resource, representing its *state*. In addition, a Resource declare a set of OCF *Interfaces*. Each Interface specifies how is possible to interact with the Resource itself. A Resource is addressed using URI and contains properties defined as key-value pairs.

OCF specification defines a basic Resource Type for all OCF Resources named “oic.core”. This Resource Type defines several common properties whose must be present in a Resource; some of these properties are “id”, “n” and “rt”, which define a unique identifier for the resource in the context of a Device, the name of the resource and the resource types, respectively.

OCF specifications state that a Resource can be related to another Resource through OCF Link. A Link consists of a set of parameters that define a context URI, a target URI, a relationship from the context URI to the target URI and metadata about the target URI.

In order to enable the functional interaction between OCF Client and OCF Server, 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, one representing the platform, another one representing the device and one providing an entry point to the Resources exposed by the Device (i.e. providing a list of OCF Links to the OCF Resources exposed). These core Resources are addressed using the URIs “/oic/p”, “/oic/d” and “/oic/res” and belong to the Resource Types “oic.wk.p”, “oic.wk.d” and “oic.wk.res”, respectively.

It is worth noting that the properties of the Resource representing the Device are defined by the “oic.wk.d” Resource Type. Furthermore, alongside this Resource Type, a Device Type may be specified. A Device Type is used to mandate the list of minimum OCF Resources that must be exposed by the Device itself.

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 belong to a Device Type and shall at minimum expose the mandatory Resource Properties defined by “oic.wk.d” Resource Type.

4 PROPOSAL OF INTEGRATION

The aim of this paper is the proposal of a solution which enables integration of OCF and OPC UA communication systems. In particular, the solution aims to allow a generic application based on OPC UA communication system to access each information produced by a generic OCF device. This aim is realised through a mapping of each information produced by OCF device into a corresponding information maintained inside the AddressSpace of an OPC UA Server. In this way, an OPC UA-based application may access the OPC UA Server to retrieve the information coming from OCF system.

As said in the Introduction, the mapping in the opposite direction (i.e. from OPC UA to OCF) has been proposed and presented in previous publications of the same authors. It is worth noting that the solution here proposed requires a mapping from OCF Resource Model to the OPC UA AddressSpace, to allow the representation of each element featuring the OCF Resource Model into a correspondent element belonging to the OPC UA AddressSpace.

It is very important to point out that this has been realised through an extension of the OPC UA Information Model, as the authors discovered that the native one was not able to represent the entire set of elements featuring the OCF Resource Model. In the paper, the novel Information Model will be introduced with the name *OCF OPC UA Information Model*. It is built on top of the standard OPC UA Device Model, described in the previous section.

4.1 OCF OPC UA Information Model

The proposed *OCF OPC UA Information Model* offers several novel OPC UA *ObjectTypes*, called *OCF ObjectTypes* and described in the remainder of this section. They are: *OCFResourceType*, *OCFResourceInstanceType* and *OCFDeviceType*.

4.1.1 OCFResourceType ObjectType

The OCFResourceType ObjectType has been defined into the OPC UA AddressSpace with the aim to represent an OCF Resource Type. OCFResourceType is Abstract; this means that an ObjectType extending it shall be created for each OCF Resource Type to be represented.

Figure 1 shows the structure of the OCFResourceType.

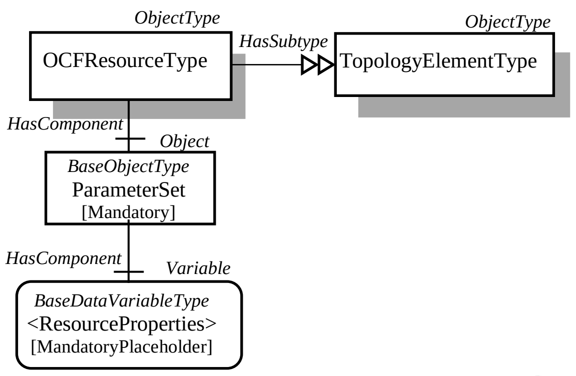


Figure 1: OCFResourceType.

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 them, there is the *ParameterSet* Object, which 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 1 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 called “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 2 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 a Mandatory ModellingRule Object; the ModellingRule Object associated to the InstanceDeclaration is Mandatory as the “brightness” property is mandatory, as said before.

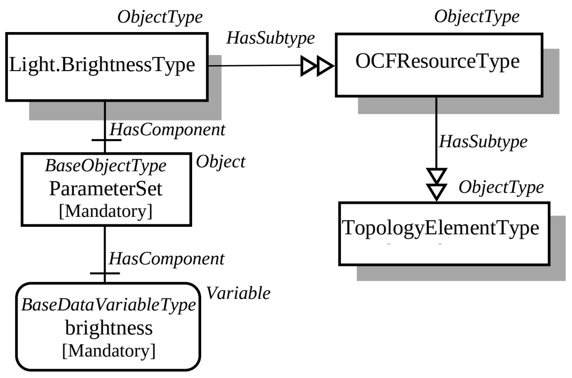


Figure 2: Light.BrightnessType ObjectType.

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 3 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; in this example, it has been assumed that the actual value was 80, as shown by Figure 3.

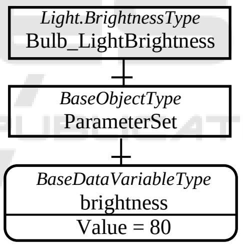


Figure 3: Instance of Light.BrightnessType.

4.1.2 OCFResourceInstanceType ObjectType

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

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 two aggregations, 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 3). 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 two aggregations just pointed out, are realised using the Configurable Component pattern defined in (OPCFoundation, 2013) and the ConfigurableObjectType described in Section 2. The instance of OCFResourceInstanceType created for each OCF Resource, contains (through a HasComponent Reference) an OPC UA Object of ConfigurableObjectType ObjectType, named *Aspects* in this paper. 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 ObjectType, Aspects owns a folder named *SupportedTypes*; it is used to organise the subtypes of OCFResourceTypes allowed as component of the Aspects Object. Figure 4 shows the details of the 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 to all the OCF properties featured by the OCF Resource.

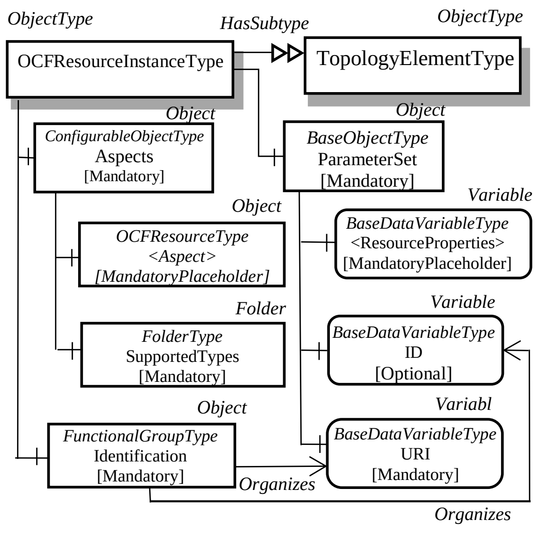


Figure 4: Details of OCFResourceInstanceType ObjectType.

ParameterSet groups also other Parameters, among which Figure 4 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 2.

4.1.3 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 to be represented into the OPC UA AddressSpace; an instance of such subtype maps an OCF Device and the information it gathers. OCFDeviceType is graphically described by Figure 5.

As explained in Section 3, 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.

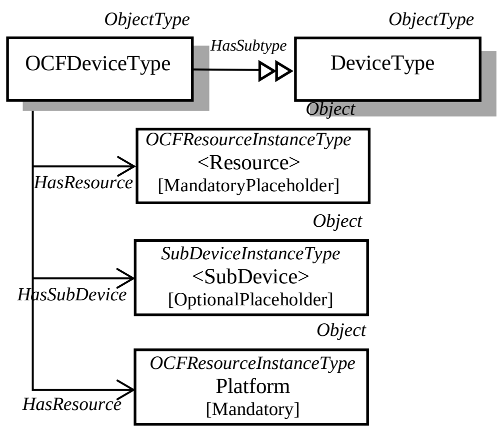


Figure 5: OCFDeviceType ObjectType.

As shown by Figure 5, 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 3.

The OCF Resource addressed by "/oic/p" is mapped as an instance of OCFResourceInstanceType, named *Platform*; this explains the presence in Figure 5 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 5, 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.

5 EXAMPLE OF INTEGRATION

The aim of this Section is to provide an example of the mapping from OCF Resource Model to OPC UA AddressSpace, using the OCF ObjectTypes presented in the previous section. For the sake of simplicity, the example will focus only on the mapping of an OCF Resource into OPC UA Nodes.

Let us consider an OCF Resource addressed by the URI "/a/bulb" and let us assume that this it is an 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 has been already described before; as said, it features only one mandatory property of integer type, named "brightness". Both the integer "value" and the boolean "brightness" properties are part of the state of the OCF Resource. It has been assumed that their actual values are *80* and *true*, respectively.

The OCF Resource addressed by the URI "/a/bulb", is mapped to the OPC UA AddressSpace by the instance of OPC UA Object of *OCFResourceInstanceType* type, named Bulb as shown by the Figure 6. According to the definition of the type shown by Figure 4, the instance has three components: Aspects, ParameterSet and Identification.

Aspects has three components shown by Figure 6. The SupportedTypes is a folder which organises Switch.BinaryType and Light.BrightnessType modelling the two OCF Resource Types "oic.r.switch.binary" and "oic.r.light.brightness".

The other two components of Aspects are Bulb_SwitchBinary and Bulb_LightBrightness, which are instances of Switch.BinaryType and Light.BrightnessType, respectively. For each of them, the actual values of the properties are shown (i.e. value and brightness).

Another component of the Bulb Object is the ParameterSet Object. It is used to allow a direct access to the two actual values of the properties of Bulb_SwitchBinary and Bulb_LightBrightness Objects (i.e. value and brightness), and to the mandatory URI Variable (containing the URI address of the OCF Resource).

The last component of the Bulb Object is the Indentification; it is a folder organising only the URI Variable, as it is assumed that the ID Variable has not been implemented.

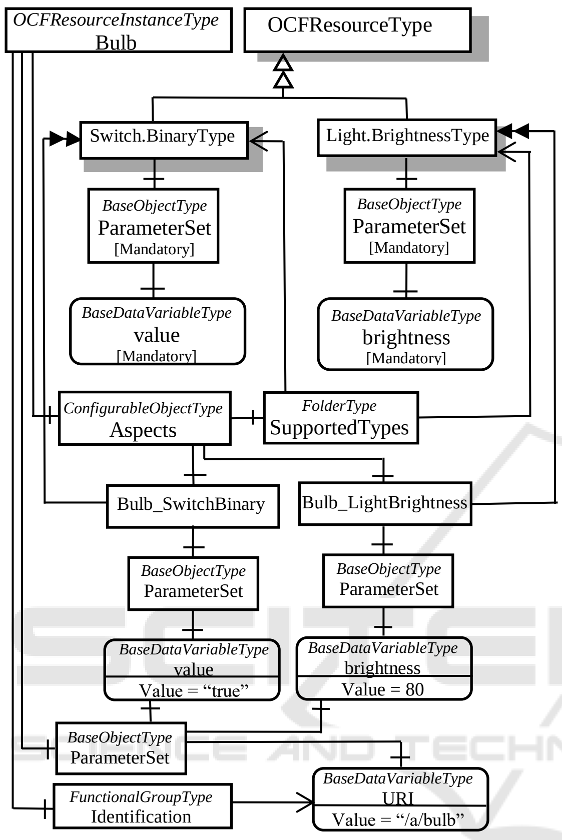


Figure 6: Example of Mapping of an OCF Resource.

6 CONCLUSIONS

In this paper, a novel OPC UA information model able to map elements of the OCF Resource Model into the OPC UA AddressSpace has been proposed. A GitHub repository has been realised by the authors at the address *https://github.com/OPCUAUniCT*, containing the definition of the entire novel OPC UA information model here presented.

ACKNOWLEDGEMENTS

The research has been partially funded with the "Research of Ateneo di Catania-Plan for Research 2016/2018".

REFERENCES

Cavalieri, S., Salafia, M.G., Scroppo, M.S., 2018. Mapping OPC UA AddressSpace to OCF Resource Model. In *Proceedings of the 1st International Conference on Industrial Cyber-Physical Systems*, San Petersburg (Russia), pp. 135–140.

Cavalieri, S., Scroppo, M.S., 2018. A proposal to make OCF and OPC UA interoperable. In Proceedings of *19th IEEE International Conference on Industrial Technology*, pp. 1551 – 1556.

Da Xu, L., Xu, E.L., Li, L., 2018. Industry 4.0: state of the art and future trends. *International Journal of Production Research*, 56(8), pp. 2941-2962, 2018.

Derhamy, H., Rönnholm, J., Delsing, J., 2017. Protocol interoperability of OPC UA in service oriented architectures. In *Proceedings of 15th IEEE International Conference on Industrial Informatics*, pp. 44-50, 2017.

Guarda, T., Leon, M., Augusto, M. F., Haz, L., de la Cruz, M., Orozco, W., Alvarez, J., 2017. Internet of Things Challenges. In Proceedings of *12th Iberian Conference on Information Systems and Technologies*, pp. 628-631.

Industrial Internet Consortium, 2017. The Industrial Internet of Things Volume G1: Reference Architecture (Version 1.80).

Izaguirre, M. J. A. G., Lobov, A., Lastra, J. L. M., 2011. OPC-UA and DPWS interoperability for factory floor monitoring using complex event processing. In 9th *IEEE International Conference on Industrial Informatics*, pp. 205–211, July 2011.

Liao, Y., Deschamps, F., Loures, E.F.R., Ramos, L.F.P., 2017. Past, present and future of Industry 4.0 - a systematic literature review and research agenda proposal. *International Journal of Production Research*, 55(12), pp. 3609-3629, 2017.

Mahnke, W., Leitner, S.H., Damm, M., 2009. OPC Unified Architecture. *Springer Verlag*, ISBN 978-3-540-68899-0, 2009.

OPCFoundation, 2013. *OPC UA for Device Companion Specification*, release 1.01, 2013.

OPCFoundation, 2015. *OPC UA Part 3: Address Space Model Specification*, release 1.03, 2015.

Open Connectivity Foundation Website, 2018. https://openconnectivity.org/

Open Connectivity Foundation, 2018. Core Specification. https://openconnectivity.org/specs/OCF_Core_Specification_v1.3.1.pdf, version 1.3.1.

VDI/VDE, 2015. Status Report-Reference Architecture Model Industrie 4.0 (RAMI4.0). *Technical Report.* Retrieved from https://www.zvei.org/en/press-media/publications/gma-status-report-reference-archtitecture-model-industrie-40-rami-40/