Communication interface specification in OPC UA

Diana Strutzenberger†, Thomas Frühwirth*, Thomas Trautner†, Ronald Hinterbichler§, Florian Pauker†

* Austrian Center for Digital Production
Seestadtstraße 27/10, 1220 Vienna, Austria
{diana.strutzenberger, thomas.fruehwirth}@acdp.at

† Workflow Systems and Technology
University of Vienna
Währingerstraße 29, 1090 Vienna, Austria
florian.pauker@univie.ac.at

‡ Institute of Production Engineering and Photonic Technologies
University of Technology Vienna
Getreidemarkt 9/311, 1060 Vienna, Austria
trautner@ift.at, diana.strutzenberger

§ EMCO GmbH
Salzburger Str. 80, 5400 Hallein-Taxach, Austria
ronald.hinterbichler@emco.at

Abstract—Due to its modeling capabilities, platform independence, and extendability, Open Platform Communications Unified Architecture (OPC UA) is considered to be the main candidate for a so-called “enabling technology for Industry 4.0”. However, the engineering effort for OPC UA applications, in particular the server, is very high as the process of information modeling and linking the resulting model to real-world data is complex and time-consuming. This significantly limits the spread of OPC UA in manufacturing and other domains.

This paper presents an approach towards extending OPC UA in a way such that information necessary for the OPC UA server application to access the underlying system can be specified in the information model. This is achieved by defining communication interfaces for all nodes of the information model that represent readable or writable data. The handling of the interfaces needs to be implemented in code only once and from then on they can be modeled, rather than implemented for each node individually. Thus, the implementation effort of OPC UA servers is reduced. Different modeling approaches for communication interfaces are presented and compared.

Index Terms—OPC UA, communication interface, web service

I. INTRODUCTION

Along with the success story of Cyber-Physical Systems (CPSs) and Cyber-Physical Production Systems (CPPSs), the development of new technologies and communication protocols such as OPC UA, which offers numerous features related to data communication such as application and user authentication but also its information modeling capability. While OPC UA clients are currently integrated in several software products, the number of (sophisticated) OPC UA servers for complex systems such as production machinery and industrial robots is still limited.

Various Software Development Kits exist which provide an extensive base for implementation, but nevertheless, for widespread use of OPC UA, the development effort for servers has to be reduced. In this paper, an extension of the OPC UA information model is presented, which allows to model different communication interfaces in the address space of the OPC UA server. Whenever a variable node is accessed by an OPC UA client, the OPC UA server browses its own address space for a referenced communication interface and uses its information to read/write the value from/to the underlying system.

The paper is structured as follows: Section II discusses related scientific work. Section III presents various approaches how communication interfaces can be modeled in OPC UA. The different approaches are compared in Section IV and a proof of concept implementation is discussed in Section V. Finally, Section VI gives a short summary, discusses the main benefits and shortcomings of the presented approach, and provides an outlook on future work.

II. STATE OF THE ART

Various ideas and tools have been developed to support a developer in OPC UA information modeling, software development, and the creation of OPC UA servers in general.

In “A Systematic Approach to OPC UA Information Model Design” [1], a Model-Driven Engineering (MDE)-based approach to OPC UA information modeling is presented in which Unified Modeling Language (UML) is suggested to be used as modeling language. The OPC UA information model is then created from the UML models via model transformation. Additional publications related to this paper describe how such a model transformation can be implemented [1], [2].

On the smart grid sector there exist also some approaches for mapping and transforming models into code and other models [3], [4], [5].

Exposing additional information in the OPC UA server’s address space is a main motivation of the paper “Guarded state machines in OPC UA” [6]. Thereby, additional OPC UA nodes and references have been introduced that enable the modeling of guards (as known from UML state machines) in OPC UA state machines. Before performing a state transition, the OPC UA server can browse its own address space to check if the corresponding guard condition is fulfilled.

In their work “Generic OPC UA Server Framework” [7] Nikiel et al. present a similar approach to the idea of shifting the required effort from software development towards configuration/information modeling in OPC UA. Their generic server
reads a special “server design” Extensible Markup Language (XML) file containing all application-specific information. Thereby, the OPC UA server’s information model is only one part of the server design file together with the “access interface”, i.e. the information for access to the data on the underlying system.

Regarding communication interface description, the OPC UA for Devices companion specification [8] introduces the ConnectionPointType and the ProtocolType ObjectTypes, which allow to model information about interfaces of a device and the communication networks it is connected to as well as the address of the device in the corresponding network.

Further, the Analyser Devices Companion Specification [9] introduces the notion of streams. A stream is defined as mapping between the analyser device channel and the respective data source. The connection and data transfer are described in more detail than in the Devices companion specification, but still the information required for establishing the connection is not modelled.

III. Communication Interface Specification

The CommunicationInterfaceSpecification extends the information model of OPC UA with the information required for an OPC UA server to communicate with the underlying system. Because of the comprehensive information model of OPC UA, communication interfaces can be modeled in different ways: as objects (cf. Section III-A), variables (cf. Section III-B), and data types (cf. Section III-C). The various approaches are discussed in more detail in the corresponding sections, but first, some general ideas shall be presented.

The CommunicationInterfaces folder is introduced in order to provide an entry point for browsing CommunicationInterface nodes. Fig. 1 shows the structure of this folder and the suggested location under the Server object. Additional subfolders may be added such as Read, ReadWrite and Write, which group the different CommunicationInterface nodes via Organizes references according to the access levels of the nodes they are linked to. Furthermore, the CommunicationInterfaces folder object has methods, AddCommunicationInterface() and RemoveCommunicationInterface(), which allow to add additional and delete communication interfaces during runtime.

For linking the interfaces to the corresponding nodes, an additional reference type is defined. HasCommunicationInterface with the inverse name IsCommunicationInterfaceOf is a subtype of HasComponent and therefore a hierarchical reference. The source node of a HasCommunicationInterface reference can be any variable node in the address space. The target node shall be a communication interface. One node may, in principle, have multiple references to different CommunicationInterface nodes, e.g. if the underlying system provides access to the same data value via multiple interfaces. Moreover, multiple nodes may be linked to a single communication interface as in some information models there may be multiple nodes with the same data mapping. This proves to be an advantage as the interface used for multiple nodes needs to be modelled only once.

A. Modeling as OPC UA ObjectType

The self-defined CommunicationInterfaceObjectType, shown in Fig. 2, is a subtype of BaseObjectType. In this example two properties have been identified to be required for almost any communication interface: the properties UserName and Password. As there may exist communication interfaces that do not require authentication, UserName and Password are optional. The CommunicationInterfaceObjectType is an abstract type and, thus, no objects of this type can be created. Instead, additional, more specific subtypes, such as the WsCommunicationInterfaceObjectType have to be defined. All web services require an Uri property to be defined. The WsCommunicationInterfaceObjectType has two subtypes: the WsJsonObjectType and WsXmlObjectType. JavaScript Object Notation (JSON) web services return JSON objects. Likewise, XML web services return XML objects. For extracting a single data point from the JSON/XML object, both ObjectTypes have their own path properties called JsonPath and XmlPath. All properties described above have the data type String.

```
Fig. 1. CommunicationInterfaces folder
```

```
Objects
  □ Server
    □ CommunicationInterfaces
      □ AddCommunicationInterface()
      □ RemoveCommunicationInterface()
      □ Read
      □ ReadWrite
      □ Write

Fig. 2. CommunicationInterfaceObjectType
```

A useful feature of objects is the possibility for OPC UA clients to subscribe to object nodes specified as EventNotifiers. An object used as an EventNotifier with specified triggering conditions may be triggered when a communication interface is added, removed or changed. Additionally, by modeling communication interfaces as objects, methods specific to the communication interface may be added, for instance to activate or deactivate the communication interface on the underlying system and save resources if it is not used.
B. Modeling as OPC UA VariableType

The communication interface can also be modeled as a new variable type, which, in this approach, is a subtype of BaseDataVariableType. The structure is identical to the CommunicationInterfaceObjectType, besides the fact that all objects are variables.

As all information for the communication interface is defined by its properties, the value of the communication interface variable itself can be used for additional information, e.g. the state of the underlying system. Alternatively, it can be used in combination with the third modeling approach, which is presented in the following section.

An OPC UA client handles variables in a different way than objects. As an alternate solution to the EventNotifier attribute, events may be triggered by the CommunicationInterfaces folder of Fig 1. Therefore, if they subscribe to the CommunicationInterfaces folder, OPC UA clients will be informed about changes regarding the communication interfaces.

C. Modeling as OPC UA DataType

This modeling approach is using a structured data type. The structured DataType has similar components to the other two approaches. A DataType can only be applied in combination with a data variable or property and therefore cannot be applied as a standalone solution.

IV. DISCUSSION

Detailed information to several relevant criteria such as flexibility and access control are discussed below.

A. Flexibility in integration and extensibility

Complex objects may consist of other objects as well as variables and methods. However, objects cannot be instantiated under variables. Such constraints confine the possibilities for effective integration into the address space. Therefore, in the ObjectType approach, CommunicationInterface nodes can only be added to other object nodes, but not to variables, methods, etc. with the HasCommunicationInterface reference. This is a severe limitation and favors the VariableType and DataType approaches over the ObjectType approach.

B. Access control

Using the ObjectType approach proves to have benefits regarding access to the individual variables. The variable approach shares this advantage. The restriction of the access level, not only for the value attribute of the variable but for the complete property set, demands addressing every property individually. With the DataType approach, only the restriction of the access level for the complete set of properties is possible.

C. Adding/removing communication interfaces

The complexity of the AddCommunicationInterface() method, introduced in Section III, varies with respect to the modeling approach. In OPC UA a DataType has to be defined for each input argument of methods. One solution could be to use multiple input arguments to create an interface or to use the CommunicationInterfaceDataType which avoids inconsistencies. As variables and objects cannot be passed to the OPC UA server as input arguments of the AddCommunicationInterface() method, the DataType approach is clearly favored.

D. Updating communication interface details

By defining the EventNotifier attribute for each CommunicationInterfaceObject, OPC UA clients can subscribe to changes of each individual CommunicationInterface object they are interested in. As an alternative, the CommunicationInterfaces folder can be used to issue an event whenever any CommunicationInterface variable is updated.

E. Support by OPC UA stacks and clients

The application of self-defined structured DataTypes is restricted by the fact that, although they are defined in the OPC UA specifications, not every OPC UA client may support structured DataTypes. This is also discussed in [10].

F. Support by SDKs

The read and write services, which are provided by the server and called by the client, need to be able to automatically browse all HasCommunicationInterface references starting from the required variable node. The values of the end nodes hold the data necessary to establish a connection to the desired data source. In order to be able to apply this solution the read and write function in the SDK need to be adaptable. Also functionality for an internal OPC UA client needs to be available to implement the internal browsing step.

G. Recommended modeling approach

The issue of how complex data structures, such as the presented CommunicationInterface, shall be modelled has already been identified by Mahnke et al. in “Chapter 3.3.3. Providing Complex Data Structures” [10]. Three approaches are described: providing “several variables with predefined DataTypes”, providing “one variable with a structured DataType” and providing “one variable with a structured DataType and subvariables”. With respect to recommendations provided by Mahnke et al., existing solutions to similar problems provided by the OPC UA standards, and companion specifications, and the above comparison of the various modelling approaches, a combination of the VariableType and the DataType approach is recommended as the preferable way of modelling CommunicationInterface nodes.

Thereby, CommunicationInterface nodes are instances of the corresponding VariableType. The value of the CommunicationInterface node is of type CommunicationInterfaceDataType. The CommunicationInterfaces folder is being defined as EventNotifier and, therefore, the source of all events regarding the change of any CommunicationInterface node. As a minor drawback, the same information is modelled twice and has to be kept consistent.
V. Proof of Concept

The recommended modeling approach has been implemented on an OPC UA server for an industrial robot. The robot controller can be accessed via a web service interface which allows the projection of control unit to another computer. The data returned by the web services is encoded as JSON or XML objects. The relevant subset of the corresponding OPC UA information model is illustrated in Fig. 3. It follows the Devices Companion Specification [8]. The IRB2600 is an instance of the DeviceType, and, therefore, placed under Objects.DeviceSet. It holds the folder object Axes under which every axis of the robot is instantiated. Each axis has the mandatory variable ActPos of DataType Double.

In the presented example, only read access shall be granted on the ActPos of Axis1. The corresponding communication interface CIAxis1ActPos is therefore organized by the Read subfolder. CIAxis1ActPos is an instance of WsJsonVariableType (cf. Section III-B).

ActPos of Axis1 and CIAxis1ActPos are connected via a HasCommunicationInterface reference. Whenever an OPC UA client reads the value of Axis1.ActPos, the OPC UA server issues a web service request to the underlying robot system controller. All necessary information for this request (Password, UserName, and Uri) is specified in the CIAxis1ActPos communication interface object. The result provided by the robot system controller is a JSON object containing the position of Axis1 alongside some meta information.

A proof of concept has been implemented using the modeling tool UaModeler\(^1\) by Unified Automation and the corresponding C++ Software Development Kit (SDK)\(^2\). The overall approach, however, is not limited to a specific software stack.

VI. Conclusion and Future Work

This paper outlined the idea of including information about communication interfaces in OPC UA information models. Whenever a node is accessed by an OPC UA client, the OPC UA server browses its own address space to determine the corresponding communication interface and uses this information to read/write the value from/to the underlying system.

With the presented approach the implementation effort of OPC UA servers is reduced, as most of the workload is shifted from coding to modeling.

While the presented approach is well suited to model communication interfaces for simple data values, its applicability to other OPC UA nodes, e.g. structured DataTypes and methods, is limited. Furthermore, exposing information about the underlying communication interface to any OPC UA client connecting to the OPC UA server, in particular regarding sensitive data such as user names and passwords, may be undesired in many situations. OPC UA offers a number of mechanisms to limit access to the address space, which have to be analyzed with respect to their suitability in the context of this paper.

Acknowledgements

This work has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the “Austrian Competence Center for Digital Production” (CDP) under the contract number 854187.

References


---

\(^1\)https://www.unified-automation.com/products/development-tools/uamodeler.html
\(^2\)https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html