This regular paper was presented as part of the main technical program at IEEE ETFA'2011

Interoperability at the Management Level of Building Automation Systems: A Case Study for BACnet and OPC UA

Andreas Fernbach, Wolfgang Granzer, Wolfgang Kastner
Vienna University of Technology, Automation Systems Group
Treitlstrasse 1-3, A-1040 Vienna, Austria
{afernbach,w,k}@auto.tuwien.ac.at

Abstract

In modern building automation systems a plethora of different networking technologies exists. Therefore, interoperability between devices using various technologies is a key requirement. The use of Web Services as a platform- and technological-independent method of communication is a promising approach to address this challenge. Since IP extensions to available technologies are more and more established in building automation systems the network infrastructure and necessary protocols for Web Services communication are already present. However, providing appropriate concepts to model information that can be accessed in a generic way are still missing. OPC Unified Architecture (OPC UA) is a powerful and promising standard that aims at solving this challenge. This work discusses an approach to map the interworking model of BACnet to OPC UA. Using the resulting information model BACnet applications can be represented in OPC UA and, thus, be accessed by OPC UA clients in a standard and well-defined way.

1. Introduction

A commonly agreed model of Building Automation Systems (BAS) is the automation pyramid [1, 12]. It is divided into three levels which reflect the different functional aspects of BAS. The lowest level is the field level where the interaction with the technical process such as metering, setting, and switching happens. The automation level placed in the middle of the pyramid provides the control functionality (i.e., the execution of control loops on data prepared by the field level). At the top level of this hierarchy, the management level acts as an interface to management and enterprise applications. Configuration of the system, visualization, and archiving of process data are typical activities at this level. Another task of the management level is to provide interoperability between different systems and technologies used at the lower two tiers of the BAS. To achieve interoperability, a *general application model* covering the functionality of these systems has to be defined. Since IP based networks are commonly used at the management level of today's BAS, the most suitable concept of communication at this level is the use of Web Services (WS) [11]. WS have the advantage that they provide platform- and programming language independence. Based on the exchange of messages, WS follow the Service Oriented Architecture (SOA) paradigm. This enables devices to exchange data independently from the underlying network technologies.

Within this context, *OPC Unified Architecture (OPC UA)* is one of the most important standards supporting WS. While OPC UA is already well-established in industrial automation systems [8, 15], it gains importance within the building automation domain. In addition to the less used standards, such as oBIX [2] and BACnet/Web Services (BACnet/WS) [3], OPC UA can be used to provide a generic view to management clients that need global access to the entire BAS. However, to be able to use OPC UA at the management level, interfaces to the underlying technologies are required. Therefore, this paper presents an approach how OPC UA can be integrated into one of the most important open BAS standards used at the automation and management level, namely Building Automation and Control Network (BACnet). The paper starts with an introduction to BACnet and its application model (cf. Section 2). In the following Section 3, the main concepts of OPC UA are described. In Section 4, the main contribution of this paper is presented – i.e., a method to map the interworking model of BACnet to an OPC UA information model. Finally, as a proof-of-concept a prototype application of an OPC UA server interfacing with BACnet/IP networks is presented in Section 5. In Section 6, the paper is concluded with an outlook on ongoing research activities.

2. BACnet

The Building Automation and Control Network (BACnet) protocol was developed by the American Society of Heating, Refrigerating, and Air Conditioning Engineers (ASHRAE) and was standardized in 1995. Continuous maintenance and development are applied since then. The current standard is BACnet 2008 [3] which is also laid down as ISO 16484-5:2010 [6].

To allow remote devices to access process data, an object-oriented, "network-visible" representation of the

978-1-4577-0018-7/11/$26.00

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	REAL	W
Progress_Value	REAL	R
Resolution	REAL	O
Binary_Present_Value	BACnetBinaryPV	O
Output_Type	BACnetLightingOutputType	R
Lighting_Command	BACnetLightingCommand	W

Figure 1: BACnet object type definition [5]

stored data has been specified by BACnet. Up to now, 30 different *BACnet object types* are defined within the current BACnet standard. They differ in the composition of their so called *BACnet properties* which can be seen as datapoints i.e., the logical representation of the process data of the technical process under control. Each property has a unique identifier referred to as `Property_Identifier`, a designated property type, and a conformance code attribute. The conformance code defines the access permissions of a property and specifies whether a property must be present or not. Possible values are Readable (R), Writable (W), and Optionally present (O). Figure 1 gives an example of such an object type definition. There are three mandatory properties that must be defined for each BACnet object: `Object_Identifier`, `Object_Name`, and `Object_Type`. The former two properties must be unique within a BACnet device. Since a BACnet object is always dedicated to exactly one device, these properties can be used to reference a BACnet object within the device.

Available BACnet object types as well as the included properties are mostly generic ones. For example, BACnet defines generic object types such as the `BACnet Binary Output Object` type and the `BACnet Analog Input Object` type. It is within the responsibility of the application program to map the functionality of a dedicated application to certain BACnet objects. However, there are also efforts underway to standardize more application-specific object types in BACnet. In Addendum i to BACnet 2008, a basic BACnet object for lighting has been defined [5]. Figure 1 shows parts of the definition of the BACnet `Lighting Output` object.

Every BACnet device holds exactly one special object called `Device Object`. The `Device Object` provides basic information about the BACnet device like vendor information, firmware and protocol version, and local time and date. Additionally, its `Object_Identifier` and `Object_Name` must be unique within the whole BACnet network and so it can be used to identify the device in the network.

In addition to the representation of process data, the standard defines different communication services. Two important object access services are the `ReadProperty` and the `WriteProperty` services for getting and setting the value of a property. The `ReadProperty` service takes the `Object_Identifier`, the `Property_Identifier` and optionally the `Property_Array_Index` of the property that has to be read as arguments. If succeeded, the service response of a `ReadProperty` contains the input arguments of the request and the value of the property to be read. The `WriteProperty` service, on the other hand, takes the `Object_Identifier`, the `Property_Identifier`, the `Property_Array_Index` as well as the `Property_Value` and the corresponding `Priority` of the property that has to be written as arguments. Success is indicated to the client by sending a positive confirmation response.

3. OPC Unified Architecture

In 1995, an association of vendors developing Human Machine Interface (HMI) and Supervisory Control and Data Acquisition (SCADA) software was founded. It targeted to address the drawbacks of the plenty of vendor-specific fieldbus systems and protocols already available on the market but being not compatible among each other. The association was named *OPC Foundation*.

Its first release was a standard providing services for reading and writing process data. It was named *OLE for Process Control (OPC)*, since the protocol was based on Microsoft OLE. The idea behind OPC was that each vendor provides specific OPC drivers for network devices. These drivers link the individual network protocols to the OPC Application Programming Interface (API). This enables devices implementing different communication standards to exchange data and control information using the uniform OPC representation of data and services. In the beginning, Microsoft's Component Object Model (COM) and Distributed COM (DCOM) were used as APIs. This reuse of intellectual property enabled the foundation to focus on the development of important new features and quick adoption of the standard for the addressed use cases [13] which was an advantage of the OPC Foundation against other organizations. In addition to the original OPC standard that was later renamed to OPC Data Access (OPC DA), additional specifications were defined. Examples are *OPC Alarm & Events* (OPC A&E) that describes the handling of event based information, and *OPC Historical Data Access* (OPC HDA) which specifies an interface to archived process data.

Originally an advantage, the COM/DCOM dependency of these so called *classical OPC specifications* became more and more a limitation to many applications [14]. In addition to the insufficiency of COM/DCOM (e.g., limited remote access support, weak security mechanisms, dependency to Microsoft Windows systems, incompatibility of COM/DCOM between different Windows versions), another drawback of classical OPC was the weaknesses in modeling complex data and systems caused by the lack of object oriented concepts like using a type hierarchy. To eliminate these drawbacks, the *OPC Unified Architecture (OPC UA)* [4, 7] was released as a full replacement of the classical OPC specifications [10]. The main points of evolution of this new standard are:

- Combine all features from the classical OPC specifi-

cation into one specification
- Achieve platform independency by using Web Services and TCP based protocols for communication
- Allow remote access over the Internet
- Provide strong security mechanisms
- Use of a common object-oriented model for representing any kind of data
- Allow scalability in data complexity
- Offer the possibility to model meta information of process data
- Provide an abstract base model from which other user-defined models can be derived

Data modeling and transportation are the two core components of the OPC UA specification. The *meta model* defines base modeling concepts and rules that can be used to model data. The *transport* part describes communication services that can be used in combination with two different transport protocols: a TCP based binary protocol for efficient communication and data exchange as well as a Web Service based protocol using XML and SOAP. The exchange of data in OPC UA follows the client-server model. On top of these two core components, the *OPC UA service API* and the *OPC UA information model* are located. The services in OPC UA are used to exchange data between OPC clients and servers. If a client wants to access data on the server, it calls a distinct method of the service set. Typical examples are the *discovery service set* to discover the available servers and the *attribute service set* to read and write OPC UA attributes. The *OPC UA information model*, on the other hand, that is used to represent process data is described in the following.

3.1. Information modeling in OPC UA

Contrary to classical OPC which only provides possibilities to represent basic data, OPC UA supports mechanisms to expose specific semantics to data. For example, in addition to the measurement value of a sensor, information about the senor type or the device that implements the sensor functionality can be modeled.

Interoperability between devices of different vendors requires a uniform representation of data. In OPC UA, the idea is to define *information models* (i.e., data representations) for different application domains. Vendors can use these models or can even extend them by their own domain-specific knowledge. Clients do not have to distinguish between different vendors since they all have the same base model exposing data in common.

Information models in OPC UA are based on a meta model called *address space* which prescribes the following rules:

- Information is modeled in form of nodes carrying attributes and references linking the nodes.
- Type hierarchies and inheritance are used as object-oriented principles.
- There is no distinction between the exposure of data and type information. The latter is needed by clients to interpret the data which is accessed.
- Information is modeled in form of a network with full-meshed nodes. There is no unique way to model information. Each use case requires a specific manner of modeling.
- The base information model as part of the specification is extensible with regard to defining subtypes of nodes and references between them.
- Information models only exist on OPC UA servers. Clients gain their knowledge about how data is modeled by fetching that information from the server.

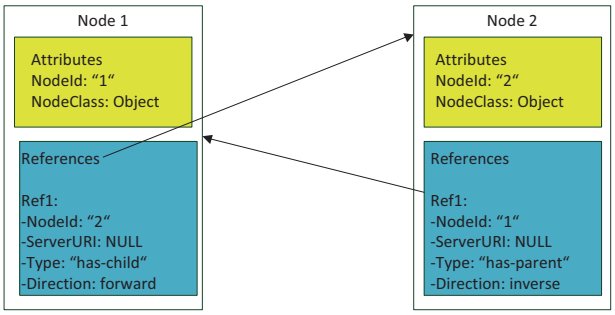


Figure 2: The concept of nodes and references [13]

Nodes in OPC UA consist of attributes which give a description of the node and references creating links to other nodes (cf. Figure 2). Some attributes are inherent in all node classes, some are specific. Examples of common attributes are the `NodeId` for uniquely identifying the Node in the address space, the `BrowseName` that identifies a node when browsing through the address space, and the `DisplayName` attribute which contains the name of the node to be displayed in a user interface. For the entire list of attributes see [4].

Each node is assigned to a distinct *node class*. Basically, it can be distinguished between node classes defining types and node classes defining instances of types. The built-in type definition node classes are the following:

- `DataType` node class: defines the data type of the value attribute of a variable or variable type
- `VariableType` node class: used to define the type of a variable
- `ObjectType` node class: specifies the type of an object
- `ReferenceType` node class: used to specify reference types

The following built-in instance definition node classes are defined:

- `Variable` node class: variables must always belong to another node (e.g., an object). The `Value` attribute holds a physical value of a technical process (if it is linked by a `HasComponent` reference) or provides meta information for the superior node (when referenced by `HasProperty`). When refer-

enced by a `HasProperty` reference, a variable is called *Property*.

- `Object` node class: objects consist of variables, methods, and properties. They are used to model devices or components of the technical process under control like a temperature controller or a motor controller.
- `Method` node class: methods are always referenced to an object. They represent functions that can be called by the OPC UA client (e.g., start and stop routines of a motor controller).
- `View` node class: In order to reduce the scope of a client accessing an information model on a server, views can be used to make only parts of it visible. Depending on the use case, only the relevant part of the whole model is accessible by the client.

Users can extend the built-in information model by defining their own use case specific type definitions. These types are inherited from built-in ones and enhanced with additional semantics by using user-defined names (*simple types*) or by defining further subnodes (*complex types*).

References in OPC UA are applied to create a link between two nodes. Although a reference type is handled internally as a node, references do not have attributes and are not directly accessible – only indirectly by browsing a node. However, reference types follow the same extensible concept as nodes. Users can likewise inherit special reference types from built-in ones in order to define the required semantics. References are divided into hierarchical and non-hierarchical ones. Hierarchical reference types are typically used in type hierarchies (e.g., the `HasSubtype` reference) or when assigning properties to objects or variables by a `HasProperty` reference. The `HasTypeDefinition` is a typical non-hierarchical reference.

4. OPC UA information model for BACnet

The same challenges the OPC foundation originally addressed when releasing the classical OPC standards exist in the building automation domain. There are many different control and fieldbus networks and technologies available but they are not compatible among each other. Since OPC UA is mainly designed for usage at the management level of the automation pyramid, it is mostly utilized in this realm of building automation. Used at the management level, OPC UA can provide interoperability by abstracting the underlying networking technologies. It creates a uniform view of the process data and allows communication between network devices of different technologies. Figure 3 shows a possible setup of an OPC UA server in a building automation network that uses BACnet over IP as network. One or more OPC UA servers are used to gather data from different BACnet controllers to create a live process image. OPC UA clients that are located within the same subnet or even in a foreign WAN can access this process image for monitoring purposes (e.g., for visualization and trending applications). Another use case

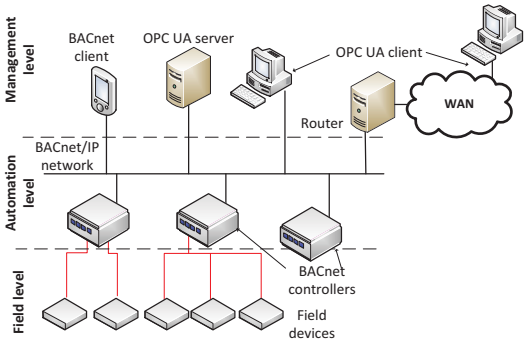


Figure 3: OPC UA in a BACnet network

is taking over control of the process from the management level by an operator or facility manager.

In the work presented, the focus is on building an OPC UA information model for BACnet. Using this information model, OPC UA servers and clients can be used to implement management applications that need to gather data from BACnet networks. There is a significant resemblance in BACnet and OPC UA with respect to data mapping. Both standards follow an object oriented approach. However, the modeling concept in OPC UA is more advanced than in BACnet since the latter does not support inheritance. Thus, defining a type hierarchy is not possible in BACnet. Section 4.1 presents how the interworking model of BACnet can be mapped to OPC UA. Another similarity exists in addressing the objects holding the process data. In BACnet, objects have an `Object_Identifier`, properties have a `Property_Identifier`. In OPC UA nodes are referenced by their `NodeId`. A mapping of these two addressing schemes is given in Section 4.2. Furthermore, the concepts of services used to access data are similar in both standards. Access services to read and write exist in both worlds. Alarm and event handling are also defined which allow, for example, the monitoring of process variables and triggering of an event or an alarm if a change of values happens or a threshold is exceeded. While this aspect is not treated in this paper, an outlook is given in Section 6.

4.1. BACnet interworking model

Due to the powerful capabilities of OPC UA the BACnet view of data can be modeled in OPC UA. The chosen approach is to transform BACnet objects to OPC UA complex objects. BACnet properties as members of BACnet objects are in turn mapped to OPC UA variables referenced by the corresponding OPC UA objects. In order to instantiate an entity in OPC UA, a type describing it has to be defined before. This needs to be done for the objects, variables, and references.

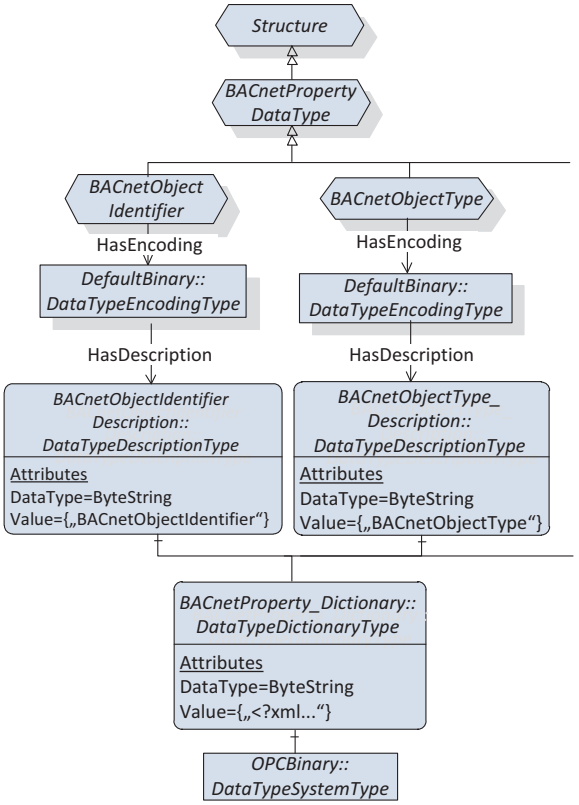
Since the value attribute of a variable is of a particular data type, the first thing to do is to define a data type hierarchy that represents the available BACnet data types. Some of these BACnet data types can directly be mapped to the built-in OPC UA data types.

For instance, the BACnet property type `REAL` (e.g., used by the property `Present_Value` of a BACnet `Lighting Output Object` type) can be modeled as the OPC UA `Float` data type. However, there are more complex BACnet property types that can not be represented by built-in OPC UA data types. Two examples are the `BACnetObjectIdentifier` and the `BACnetObjectType`. These BACnet data types have to be modeled as subtypes of OPC UA built-in data type `Structure` which can be used to model complex data types. An exemplary part of it is shown in Figure 4a. All user-defined BACnet data types are subtypes of the user-defined abstract data type `BACnetPropertyDatatype` that is inherited from the OPC UA built-in data type `Structure`. For each user-defined structured data type, at least one encoding has to be defined that is used by clients to correctly interpret the user-defined data. In the proposed model, `DefaultBinary` encoding is chosen for all user-defined data types. For every encoding, a description of the type (represented by a `DataTypeDescriptionType` node) exists which in turn is a component of the `BACnetPropertyDictionary`. Within this user-defined dictionary, the entire encoding is described in XML format. For the BACnet property type `BACnetObjectIdentifier`, this XML representation looks as follows[1]:

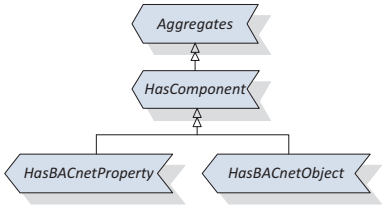
```
<StructuredType Name="BACnetObjectIdentifier">
        <Field Name="ObjectType"
               TypeName="Bit" Length="10">
        </Field>
        <Field Name="InstanceNumber"
               TypeName="Bit" Length="22">
        </Field>
</StructuredType>
```

After having defined the BACnet data types, the BACnet properties have to be represented in OPC UA. To achieve this, user-defined OPC UA variable types are defined that are used for the instance declarations of the BACnet properties. Each of these BACnet specific user-defined variable types is a subtype of the abstract user-defined `BACnetPropertyType` variable type. This abstract variable type contains the user-defined OPC UA property `BACnetPropertyId` which represents the BACnet `Property_Identifier`. This attribute is unique for each BACnet property. Some definitions of such variable types are shown in Figure 4c. To create user-defined OPC UA variable types, the corresponding attributes of the new variable type have to be set. The `DataType` attribute is set to the corresponding user-defined OPC UA data type defined before. The `AccessLevel` informs the OPC UA client about access permissions to the particular variable. In this information model the access permission facet of the conformance code of BACnet properties is mapped to the OPC UA `AccessLevel` attribute. Possible values are `Readable` and `Writeable`. Examples for further attributes to be set are the `BrowseName` and the

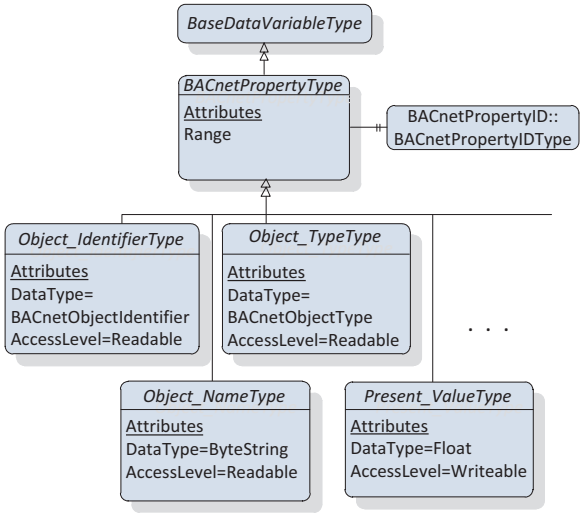
[1]For details about the XML representation refer to Part 3 of [4].



(a) Datatype Definition



(b) Reference Type Definition



(c) Variable Type Definition

Figure 4: Definition of BACnet Types

DisplayName which are both set to the human-readable name of the BACnet property defined in the standard. To assign the variables representing BACnet properties to OPC UA objects, references are used. To express the special semantics of these references, the new reference type HasBACnetProperty has been defined. This reference type is inherited from the hierarchical type HasComponent (cf. Figure 4b).

Now having all the necessary components available, the BACnet object types can be modeled in OPC UA. All BACnet object types are represented by user-defined OPC UA complex object types that are all subtypes of the abstract user-defined BACnetObjectType. This object type contains the BACnet properties Object_Identifier, Object_Name, and Object_Type that are common to all BACnet objects. The assignment of the variables that represent the BACnet properties to the corresponding object type is done by using the HasBACnetProperty reference mentioned before. To model the part of the conformance code of BACnet properties that specifies whether a property must be present or not, an OPC UA ModellingRule is defined for each variable. In this information model only the Mandatory and Optional modeling rules are used. The former forces the particular variable to be instantiated, the latter leaves only an option for that. Figure 5 shows example how modeling rules can be used.

Inherited from the abstract BACnetObjectType all BACnet object types that are specified in the standard can be defined in OPC UA. Figure 5 shows an example how the BACnet Device Object type and the BACnet Lighting Output Object type are represented using this concept. As shown in this figure, only the object specific variables are defined – the common ones are inherited from the supertype. As it is common in OPC UA, the HasSubtype reference is used to model the relation between sub- and supertype. An example of how meta information can be modeled is also shown in Figure 5 in form of the EngineeringUnit node referenced from the Power variable. To model the assignment of a unit to the value of a variable, the UPC UA built-in reference HasProperty is taken.

4.2. Instantiating and addressing of BACnet objects in OPC UA

After having presented how BACnet object and property types are modeled in OPC UA, it must be specified how instances of BACnet objects and properties are represented by the OPC UA server and how they are addressed within OPC UA. In BACnet, each BACnet object is dedicated to exactly one BACnet device – BACnet objects are therefore never distributed across more than one BACnet device. Therefore, it is reasonable to use a device-centric view – each BACnet device is represented as an OPC UA object instance of the user-defined object type BACnetDeviceType which in turn is a subtype of the standard OPC UA BaseObjectType (cf.

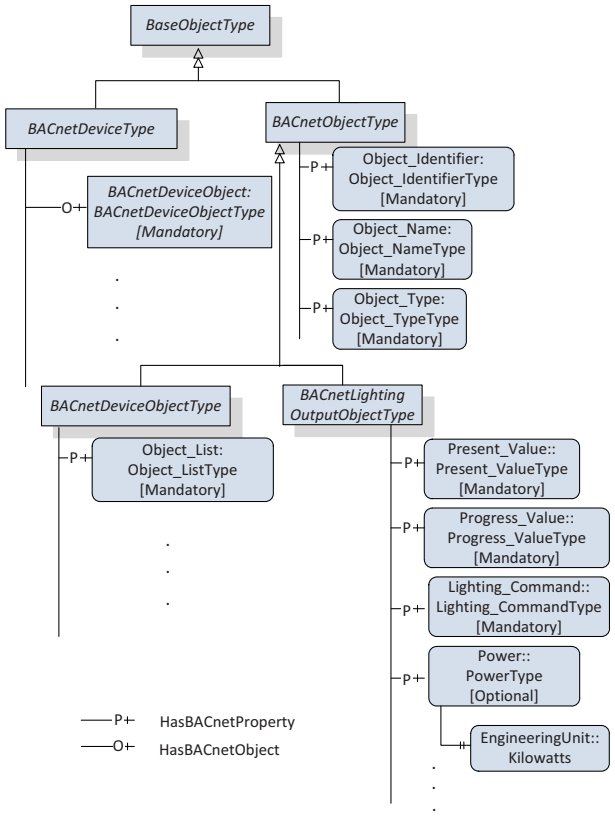


Figure 5: Object type definition

Figure 5). The corresponding BACnet objects are assigned to the OPC UA object by using the user-defined HasBACnetObject reference which is a subtype of the standard OPC UA HasComponent reference type (cf. Figure 4b). Figure 6 shows an example how a BACnet device that contains a BACnet Device Object and a BACnet Lighting Output Object is modeled.

What is still remaining is how the BACnet properties can be addressed. BACnet properties are addressed by the Property_Identifier which is unique within the object. In the proposed information model, this can be done by reading the BACnetPropertyID property that is dedicated to each BACnet variable definition. To address the BACnet object, the Object_Identifier which is unique within the device is used. The Object_Identifier can be determined by the reading the value of the Object_Identifier variable that is mandatory for each BACnet object. Finally, to address the device itself, the BACnet Device_Id or the Device_Name which are both unique within the whole BACnet network can be used. To determine the BACnet Device_Id within the OPC UA model, the value of the Object_Identifier variable of the Device Object has to be read – to determine the Device_Name, the value of the Object_Name variable of the Device Object has to be retrieved. As a result, the combination of the value of the BACnetPropertyID property, the value of the OPC UA Object_Identifier variable, and the value of

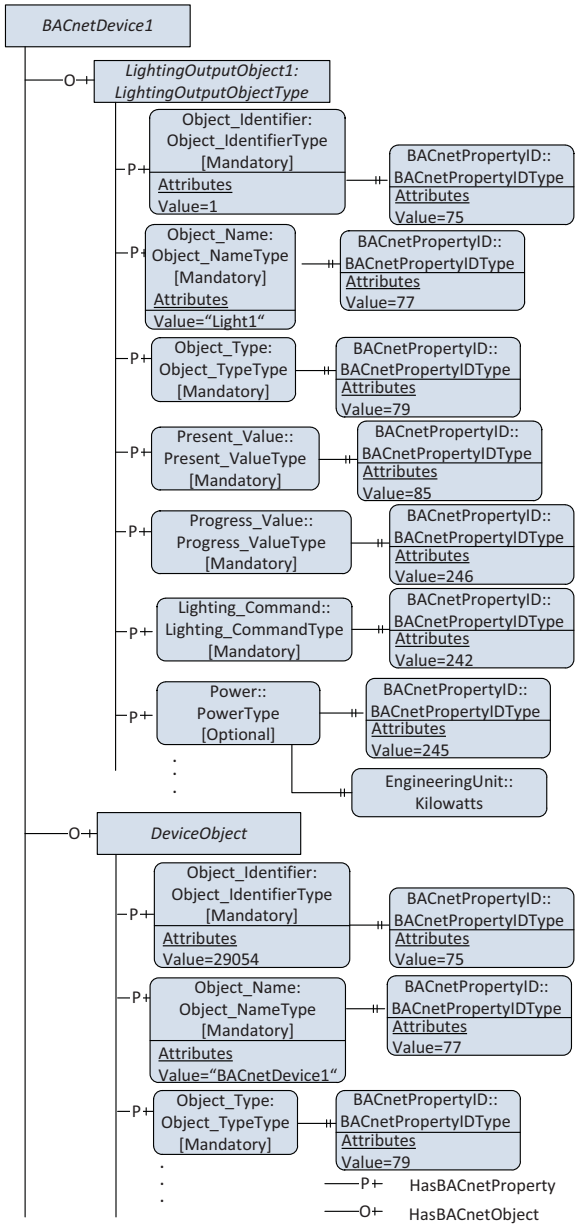


Figure 6: Instantiation of a BACnet device

the `Object_Identifier` variable of the `Device Object` (or the value of the `Object_Name` variable of the `Device Object`) is used to address a BACnet property in the presented OPC UA model.

Figure 6 illustrates an instantiation of a BACnet `Lighting Output Object`. Consider, for example, an OPC UA client browses to the `Present_Value` variable of the BACnet `Lighting Output Object` and wants to read the value of it. To read its current value, the OPC UA server needs to invoke the BACnet `ReadProperty` service. To send this request, the address information has to be determined. First, the `Property_Identifier` is determined by reading the `BACnetPropertyID` property of the `Present_Value` variable (in the proposed example `85`). Afterwards, the value of the `Object_Identifier` variable is read (in the given example `1`). Then, the `Device_Id` is determined by reading the `Object_Identifier` variable of the `Device Object` (in the proposed example `29054`). Using the combination of these values, the OPC server is able to send the `ReadProperty` request to the BACnet device. After having received the response, the OPC server is able to forward the present value to the OPC UA client.

5. Implementation

In the context of the EraSME project "Web-based Communication in Automation (WebCom)"[2] an OPC UA framework called *Comet* has been developed by the project partner HB-Softsolution[3]. Among other software modules it contains a Software Development Kit (SDK) for implementing Java based OPC UA servers. This server SDK is functionally separated into two parts: one is the core OPC UA server which is based on the OPC UA Java stack released by the OPC foundation. This core server loads the standard OPC UA information model plus any user-defined information model out of one or more XML files. This way the configuration part is completely isolated from the server's code. As a result, the server's information model can be changed and extended even during runtime. The second part of the server module consists of a driver framework which allows to implement drivers for particular network technologies that can be loaded into the core server. These drivers are responsible for interfacing with the required protocol stacks of the used technologies. Depending on these technologies the stack implementations can freely be chosen. The driver framework only provides an API for read and write methods which have to be implemented individually.

To evaluate the developed information model for BACnet, a proof-of-concept implementation was performed. The implementation uses the Comet framework to implement an OPC UA for BACnet/IP networks. The driver implementation for the required interface to the BACnet/IP network is based on the open source *BACnet/IP for Java* stack[4]. It is a high-performance implementation of the BACnet/IP protocol supporting the most important kinds of BACnet services and objects. Emulating a BACnet device by instantiating local BACnet objects is also suitable for this implementation.

Another important software module of the Comet framework is the *OPC UA Model Designer* which was used to implement the developed BACnet information model. As an editing tool, it can be used to generate information models and extend existing ones. It provides a graphical user interface that supports the user in applying definitions of data types, variable types, reference types, and object types. Furthermore, instances can be derived from these type definitions in a very comfortable way. The hierarchical structure of the resulting information model

[2] http://www.webcom-eu.org/
[3] http://www.hb-softsolution.com/
[4] http://bacnet4j.sourceforge.net/

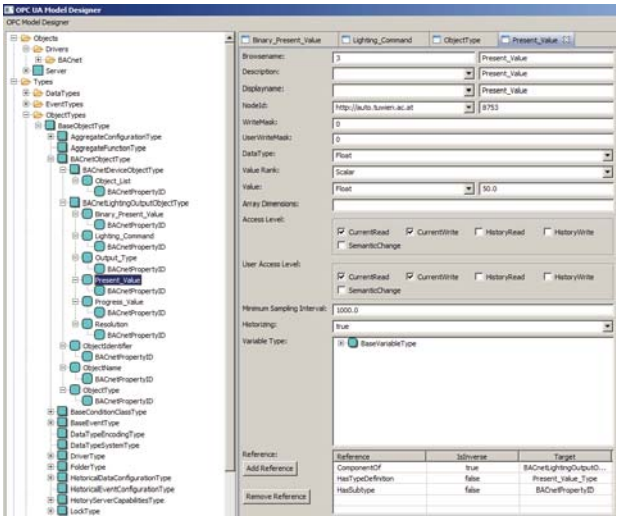


Figure 7: OPC UA Model Designer

is expressed by a tree view. A screenshot in Figure 7 shows the definition of the BACnet `Lighting Output Object` (without completeness of properties) embedded in its type hierarchy. The information model created by the Comet Model Designer is finally exported in XML format. This file can be opened again by the model designer for further editing or it can be used as input for the Comet OPC UA Server.

To show the feasibility of the developed information model, an instance of a real-world BACnet controller was modeled within the Comet Model Designer. This BACnet controller is used to control a Heating, Ventilation, and Air Conditioning (HVAC) test installation within a laboratory equipment. The resulting OPC UA model of this BACnet controller is loaded into the Comet OPC UA server which can be accessed by any OPC UA client to control the HVAC test installation.

6. Conclusion and outlook

In this paper an approach of establishing an OPC UA information model for BACnet was presented. As an example, the BACnet `Lighting Output Object` was taken to show the way a BACnet object with its properties can be mapped to OPC UA. In addition to the BACnet `Lighting Output Object`, it is also planned to map the remaining standardized BACnet object types into the developed information model. On the way to a complete information model, other OPC UA concepts shall be transferred to BACnet. Especially the alarm and event service sets which allow to monitor BACnet properties and generate alarms on particular conditions have great practical relevance and so a mapping to monitoring and subscription mechanism of OPC UA is one of the next steps. Furthermore, it shall be investigated how other parts of the OPC UA specification (e.g., OPC Historical Access) can be used to represent additional aspects of BACnet.

To achieve the desired interoperability in BAS, information models have to be introduced for other technologies, too. A similar information model that maps the interworking model of KNX into OPC UA was already presented in [9]. A final step in this process is to design a model representing the common aspects of BAS in general.

Also within the focus of the WebCom project is the implementation of a KNX driver for the OPC framework presented in the previous section. It should act as a proof of concept like the BACnet driver implemented in the context of this work. Furthermore, an interoperability lab will be set up to put OPC UA software implementations into operation and to run distributed tests over the Internet.

Acknowledgement

This work was funded by FFG (Austrian Research Promotion Agency) under the EraSME/COIN project “Web-based Communication in Automation” (P824675).

References

[1] Building Automation and Control Systems (BACS) – Part 2: Hardware. ISO 16484-2, 2004.

[2] oBIX 1.0 Committe Specification. OASIS, 2006.

[3] BACnet – A Data Communication Protocol for Building Automation and Control Networks. ANSI/ASHRAE 135, 2008.

[4] OPC Unified Architecture Specification. OPC Foundation, 2009.

[5] BACnet – A Data Communication Protocol for Building Automation and Control Networks. ANSI/ASHRAE 135-2008: Addendum i, 2010. Status: 4th public review.

[6] Building Automation and Control Systems (BACS) – Part 5: Data Communication Protocol. ISO 16484-5, 2010.

[7] OPC Unified Architecture. IEC 62541, 2010. Current status: Approved for FDIS circulation.

[8] R. Cupek and A. Maka. OPC UA for vertical communication in logistic informatics systems. In *IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–4, 2010.

[9] W. Granzer, W. Kastner, and P. Furtak. KNX and OPC UA. In *Konnex Scientific Conference*, Nov. 2010.

[10] T. Hannelius, M. Salmenpera, and S. Kuikka. Roadmap to adopting OPC UA. In *IEEE International Conference on Industrial Informatics*, pages 756–761, 2008.

[11] A. Kalogeras, J. Gialelis, C. Alexakos, M. Georgoudakis, and S. Koubias. Vertical integration of enterprise industrial systems utilizing web services. *IEEE Transactions on Industrial Informatics*, 2(2):120–128, May 2006.

[12] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman. Communication Systems for Building Automation and Control. *Proceedings of the IEEE*, 93(6):1178–1203, June 2005.

[13] W. Mahnke, S.-H. Leitner, and M. Damm. *OPC Unified Architecture*. Springer, 2009.

[14] M. Son and M.-J. Yi. A study on OPC specifications: Perspective and challenges. In *International Forum on Strategic Technology*, pages 193–197, 2010.

[15] J. Virta, I. Seilonen, A. Tuomi, and K. Koskinen. SOA-based integration for batch process management with OPC UA and ISA-88/95. In *IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–8, 2010.