Web Services in Building Automation with focus on BACnet/WS

Stefan Szucsich

e0728333@student.tuwien.ac.at

Institute of Computer Aided Automation

Vienna University of Technology, Vienna, Austria

June 2010

Abstract

Currently, there exist two major challenges in the scope of Building Automation System (BAS) integration. First of all the integration of building automation subsystems from different vendors is difficult due to still existing proprietary solutions. The second challenge is the integration between BASs and enterprise applications. Web services have been proven to be capable of solving both of these problems. This paper outlines the three most common web services based technologies in building automation, i.e., OPC Unified Architecture (OPC UA), Open Building Information Exchange (oBIX), and Building Automation and Control Network - Web Services (BACnet/WS). In the main part of this paper BACnet/WS is discussed in detail including the specified data model, attributes and services. Finally the results of a BACnet/WS performance study are presented.

Keywords: BACnet/WS, building automation, web services.

1 Introduction

Building automation industry has advanced over the last decades. Several communication protocols and a variety of Building Automation System (BAS) products from various vendors are available on the market. The major problem that arises as a direct consequence of this diversity is the interoperability of products from different vendors.

In former BASs those products have typically been interconnected by proprietary communication protocols [1]. This has led to a missing interoperability of products from different vendors. First solutions have addressed this problem by applying gateways between non-cooperating subsystems. Nevertheless, gateways have had some major disadvantages including the huge engineering effort for construction and maintenance, and the lack of scalability and extensibility. The next approach has been the employment of standard communication protocols such as LonWorks [2], Building Automation and Control Network (BACnet)

[3], and KNX [4]. They have been conceived to cover all domains of building automation, including Heating, Ventilating and Air Conditioning (HVAC), lighting, and alarming. Hence, proprietary communication protocols have become unnecessary and interoperability of components from different vendors could have been achieved. However, standard communication protocols have not been able to completely solve the integration problem due to still existing proprietary products. So new approaches have had to be developed for the integration of building automation subsystems.

BASs have typically been separated from IT networks so far [5]. However, in the last years the desire to use data from field devices within existing enterprise applications has arisen. Therefore, communication protocols supporting the TCP/IP network protocol stack have been developed. These protocols have had to cope with many problems in TCP/IP networks, such as firewalls and security issues. As for the integration problem mentioned before, new solutions have had to be found for the integration of BASs into enterprise networks.

The evolution of the web and web technologies has created new opportunities for solving both integration problems mentioned above. Web services are self-contained, modular software applications that can be published, located, and called across the web [6]. They are based on other web technologies, such as Extensible Markup Language (XML), Web Services Description Language (WSDL), and Simple Object Access Protocol (SOAP). More detailed information about the core technologies of web services can be found in [6].

Web services technology has several advantages including platform independency and a loose coupling of distributed system components. However, there exists a major drawback, too. Service requests and responses are encoded in XML format which leads to additional communication overhead and hence longer response times[1]. Also the need for processor power increases as a result of parsing the XML-formatted messages [6].

Web services can be used to implement system architectures based on the Service-Oriented Architecture (SOA) paradigm. There are various definitions for the notion of a SOA. This paper follows the definition given in [8] and [9]:

> *"A service-oriented architecture (SOA) is a set of architectural tenets for building autonomous yet interoperable systems."*

This basic definition helps to understand the most important concepts of a SOA. Its essential keywords are *automomous* and *interoperable. Autonomous* means that the systems are created independently of each other and that they operate independently of their environment. The term *interoperable* implies that the interfaces of services, that are exposed to their environment, are clearly abstracted from the implementation of these services. A summary of service-oriented architectural principles for achieving autonomy and interoperability can be found in [8].

The next section gives an overview about the most common web services based technologies in building automation. In Section 3 Building Automation and Control Network - Web Services (BACnet/WS) is described in detail including the specified data model, attributes, and services. Section 4 presents the results of a performance study of BACnet/WS and concludes this article.

[1]to counter this drawback of the standard XML format, a compressed XML format, e.g., Efficient XML Interchange (EXI) [7], can be used

2 State of the Art

Currently, there exist a couple of web services based technologies for the use in building automation. In the next sections the three most common web services based technologies are outlined and subsequently compared to each other.

2.1 OPC Unified Architecture

OPC Unified Architecture (OPC UA) [10] has been specified by the OPC Foundation[2] to overcome the disadvantages of classical OLE for Process Control (OPC)[3] standards. The main problems with classical OPC variants like OPC Data Access (OPC DA), OPC Alarm & Events (OPC A&E), and OPC Historical Data Access (OPC HDA), are their platform dependency and limited data model [11]. OPC UA has been standardized as series of standards IEC 62541. The structure of the series can be found in [10].

OPC UA is based on a service-oriented client/server architecture [12]. This means that the complete functionality of an OPC UA server is available through services. Services are divided into logical groups, called service sets. Each service set allows access to a specific aspect of an OPC UA server. However, not all services have to be provided by a single server. Thus, OPC UA defines profiles that specify which services have to be supported by a server.

To achieve platform independency, a new communication stack has been defined instead of using Microsoft's Component Object Model (COM) and Distributed Component Object Model (DCOM) technology as in classical OPC. This communication stack consists of two different transport mechanisms [12]. For efficient data transfers the binary TCP protocol *UA Native* can be used. Since there is no overhead for data encoding, this protocol is especially suitable when network and/or computer resources are limited. The second transport mechanism, the *UA Web Services* protocol, uses SOAP over Hypertext Transfer Protocol (HTTP) for transmission of XML-formatted data. This transport protocol allows an easy integration into the existing IT network since standard web service tools can be used.

Also the limited data model of classical OPC has been replaced by an object-oriented extensible data model [11]. Moreover, the new data model allows modeling of complex data as well as modeling of meta data. The collection of information that is visible to clients is called the address space. It basically consists of *nodes* and *references* between nodes. Nodes are described by *attributes* and *properties*. Most of the modeling concepts, such as *reference types* and *object types*, are represented as nodes in the address space. A reference describes the relation between nodes. The semantics of a reference is defined by its reference type.

2.2 oBIX

The Open Building Information Exchange (oBIX) specification [13] has been published by the Organization for the Advancement of Structured Information Standards (OASIS)[4] in December 2006. This platform independent technology

[2]http://www.opcfoundation.org
[3]OLE stands for "Object Linking and Embedding"
[4]http://www.oasis-open.org

is designed to provide machine-to-machine communication between embedded software systems over existing networks using standard technologies such as XML and HTTP.

As OPC UA, oBIX is also based on a service-oriented client/server architecture. Only three request/response services are defined by the oBIX specification. These services can be used to read and manipulate data or to invoke operations. Each service response is an oBIX XML document containing the requested information or the result of the service. The implementation of the three request/response services, i.e., how they are transmitted over the network, is called a protocol binding.

There are two different protocol bindings specified by the oBIX standard. The HTTP binding simply maps oBIX requests to HTTP methods. The complete mapping can be found in [13]. For more information about HTTP 1.1 see [14]. The second protocol binding is the SOAP binding. It maps a SOAP operation to each of the three oBIX requests.

One of the two fundamental elements of the oBIX specification is the concise but extensible object model. It consists of typed *objects* that are described by attributes, called *facets*. Objects are identified by either a name, a Unified Resource Locator (URL) [15], or both. The URL is used to specify the target object of a service request or to refer to other objects. Each object can contain other objects — named or unnamed. The object model can be extended by a mechanism, called *contracts*. Contracts are used to define new types but also provide a possibility of specifying default values. They are kind of templates that can be implemented by objects. By implementing a list of contracts, multiple inheritance can be accomplished.

The second essential part of the oBIX specification is the simple XML syntax to represent the object model. Basically each oBIX object maps to exactly one XML element. Sub-objects result in nesting of XML elements. The built-in object types map to the name of the XML element. All other information, e.g., the facets of an object, is represented as XML attribute.

2.3 BACnet/WS

BACnet/WS [16] has been specified by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE)[5] as addendum to ANSI / ASHRAE standard 135-2004 in October 2006. It extends the BACnet standard [3] by defining a web service interface and data model for the integration of data from field devices into existing enterprise applications. BACnet/WS has been incorporated into ANSI / ASHRAE standard 135-2008.

BACnet/WS is based on a service-oriented client/server architecture. It basically defines services for accessing and manipulating data in the server. Implementations of the service requests and responses shall conform to the Web Services Interoperability Organization (WS-I) Basic Profile 1.0 which specifies the transport of XML-formatted data over network using SOAP over HTTP [17]. Services are described in more detail in Section 3.3.

The data model defined by the BACnet/WS specification is rather simple and not extensible. Its basic element is the node. Nodes are hierarchically arranged and described by attributes. Some attributes are localizable, i.e., they

[5]http://www.ashrae.org

can have multiple values or formats for multiple languages. Section 3.1 depicts the data model in more detail. Attributes are described in Section 3.2.

2.4 Comparison

OPC UA has the most complex data model of all standards mentioned before. The data model of OPC UA is almost arbitrarily extensible by subtyping and composition. Also multiple inheritance is not forbidden in OPC UA [11]. However, only inheritance rules for single inheritance are specified. Furthermore, references with different semantics are provided. On the other hand, the BACnet/WS data model is quite simple and not extensible. There is no possibility of subtyping. Somewhere in between lies the data model of oBIX. It is concise, but very flexible. Multiple inheritance is explicitly allowed and rules for single and multiple inheritance are specified. The flexibility of the data model is achieved by contracts.

All of the above mentioned standards define a transport mechanism that uses SOAP over HTTP for transfer of XML-formatted data. Only OPC UA provides a transport protocol for raw data. Since XML adds some overhead to the data, transport of raw data can be very suitable when network resources are limited.

The integration of the BAS into existing enterprise networks leads to a higher need for security. OPC UA defines its own security model including communication channel security, authentication and authorization, and certificates [11]. It also specifies services to satisfy the security requirements. In oBIX and BACnet/WS no security mechanisms are specified. However, they both recommend the use of the Transport Layer Security (TLS) protocol for secure communication. This includes the usage of HTTP over TLS (HTTPS) instead of HTTP. More information about TLS and HTTPS can be found in [18] and [19]. Additional security measures can be placed in the client and server implementation but are not specified by these standards.

Localization can be used to provide multilingual displays or to adapt the format of dates, times, and numbers according to a given locale. Each of the previously mentioned standards allows some kind of localization. In OPC UA, the built-in data type `LocalizedText` allows to add a language tag to a localized string. The format of the language tags is defined in RFC 3066 [20]. In oBIX, localization of some facets is recommended. Auto-conversion of units may also be possible. However, the mechanisms for localization are not specified by oBIX. Localization in BACnet/WS is similar to oBIX. Several attributes are localizable but the mechanisms for localization are left as an implementation issue. Table 1 depicts the comparison of OPC UA, oBIX, and BACnet/WS.

The next section discusses BACnet/WS in more detail including the specified data model, attributes, and services.

3 BACnet/WS

The BACnet/WS standard [16] defines a web service interface that allows clients to exchange data with BACnet/WS servers. However, it does not define how the data is stored internally in a server. Furthermore this standard specifies a data model and access services. The data model and access services can be used

Table 1: Comparison of OPC UA, oBIX, and BACnet/WS

		OPC UA	**oBIX**	**BACnet/WS**
Data model	complexity	complex	concise	simple
	extensible	yes	yes	no
Transport mechanisms		SOAP/HTTP, binary TCP (for raw data)	SOAP/HTTP	SOAP/HTTP
Security mechanisms		secure channel, authentication, authorization, certificates	not specified (TLS recommended)	not specified (TLS recommended)

to access data from any source. A BACnet/WS server may act as a gateway to other protocols or own the data locally. Hence, BACnet/WS is independent from the underlying BAS.

The next sections will describe the data model, attributes, and services that are specified in BACnet/WS.

3.1 Data Model

The fundamental element in the BACnet/WS data model is the *node*. The data model consists of hierarchically arranged nodes. Each node has exactly one parent node[6] and may have an arbitrary number of children. Figure 1 shows an example node hierarchy including the *standard nodes* (see Section 3.1.2 for more details). Note that the root node is represented by a forward slash ('/') character instead of an empty string as defined in Section 3.1.1. This should help for a better understanding of the node hierarchy.

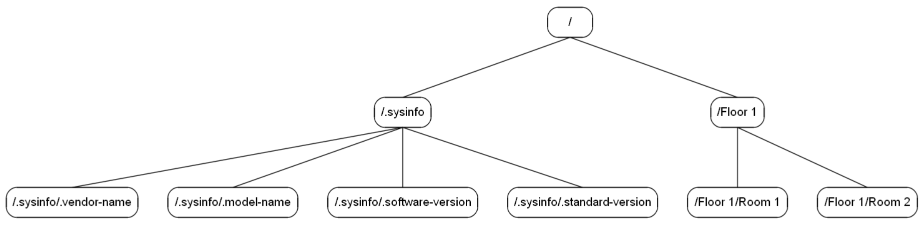

Figure 1: Example of a BACnet/WS node hierarchy

Attributes are used to describe nodes and may themselves have attributes. Some attributes are required for all nodes, and some are optional. BACnet/WS servers may define proprietary attributes at any level of the hierarchy. Attributes are described in more detail in Section 3.2.

[6]except the root node, i.e., the topmost node in the hierarchy

3.1.1 Paths

Nodes and attributes are identified by their *path*. Paths are character strings that are composed of two parts: the node part followed by the attribute part. The node part reflects the node hierarchy and consists of node identifiers separated by forward slash ('/') characters. An empty node part refers to the root node of the hierarchy. The attribute part is made up of attribute identifiers separated by colon (':') characters. If the attribute part is empty the `Value` attribute is assumed by default. The complete path form is:

```
[/node_id[/node_id]...][:attribute_id[:attribute_id]...]
```

where '`[]`' indicates an optional element and '`...`' indicates iteration of the previous element.

Identifiers are case sensitive and have to consist of printable ANSI X3.4 characters except of the following characters: `/` \ `:` `;` | < > `*` `?` `"` `[` `]` { }. Additionally node identifiers beginning with a period ('.') character and attribute identifiers not beginning with a period ('.') character are reserved. Hence, proprietary node identifiers defined by a BACnet/WS server must not begin with a period ('.') character, whereas proprietary attribute identifiers have to do. Some valid path examples are:

- `":Children"`,
- `"/Floor 1/Room 2:.Employee"`,
- `"/.sysinfo/.vendor-name"`, and
- `"/.sysinfo/.vendor-name:Value"`.

The first example accesses the `Children` attribute of the root node. In the second one a proprietary attribute called `.Employee` is accessed. The latter two expamples are logically equal and refer to the vendor name of the BACnet/WS server.

3.1.2 Standard Nodes

Structuring the node hierarchy and the naming of nodes is generally of local concern. However, there exist some standardized nodes that allow clients to retrieve some basic information about the server. As mentioned before these nodes all have identifiers beginning with a period ('.') character. Table 2 lists all standard nodes including their value types and descriptions. Additionally the standard node hierarchy is depicted in the left part of Figure 1.

3.1.3 Reference Nodes

BACnet/WS allows nodes to logically appear in different places of the hierarchy. This can be done through the use of *reference nodes*. The node a reference node refers to is called *referent node*. Except of the attributes `Children`, `Aliases`, `Attributes`, and `Reference`, all attributes of a reference node reflect the corresponding attribute of the referent node. The path to a referent node is stored in the `Reference` attribute of the reference node.

More than one reference node may refer to the same referent node. It is also possible to point to other reference nodes. However, it is not allowed to create

Table 2: Standard nodes [16]

Node Path	Value Type	Description
/.sysinfo	None	This node is the container for the following nodes (it has no value)
/.sysinfo/.vendor-name	String	Contains the name of the vendor of this server
/.sysinfo/.model-name	String	Contains the model name and/or number of this server
/.sysinfo/.software-version	String	Contains the version of the software running in this server
/.sysinfo/.standard-version	Integer	Contains the version of the BACnet/WS standard that the server is implementing

loops, nor is a node allowed to refer to itself. Figure 2 depicts a node hierarchy including references.

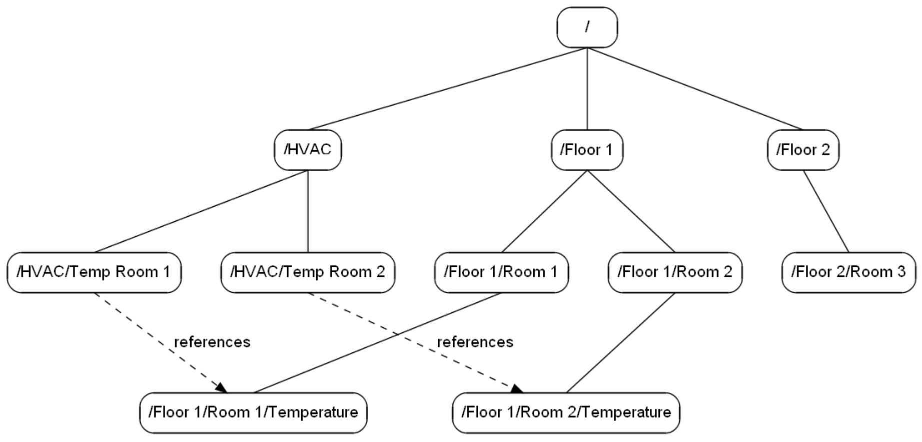

Figure 2: A BACnet/WS node hierarchy including references

3.2 Attributes

Attributes are used to store information about nodes. Some attributes are required for all nodes, and some are optional. However, it can be assumed that the set of available attributes will remain unchanged in normal operation and changes only occur after reconfiguration of the server.

There are three types of attributes. Primitive attributes are attributes of datatypes defined by the XML Schema [21], such as `boolean`, `string`, or `double`. Enumerated attributes are enumerations of XML Schema datatype `string` [21]. The set of allowed values is defined by the BACnet/WS standard. The last type, array attributes, are attributes that consist of an array of primitive attributes.

All elements of the array have the same primitive datatype. Arrays can be accessed either as an array of separate elements or as concatenation of all the elements.

Table 3 lists all attributes including their XML Schema datatype [21]. Columns 3 and 4 indicate if the attribute is an array attribute or an enumerated attribute. Column 5 shows whether or not the attribute is localizable. In column 6, the presence of the attribute is stated.

In the following subsections the attributes are described in more detail.

3.2.1 NodeType

The `NodeType` attribute is required for all nodes and allows the general classification of nodes. This attribute is enumerated. Table 4 shows the allowed values for this attribute.

3.2.2 NodeSubtype

This optional attribute — in combination with the `NodeType` attribute — allows a more specific classification of a node. Its value is a string of printable characters. The `NodeSubtype` attribute may be localized.

3.2.3 DisplayName

This required attribute is a string of printable characters that provides a short descriptive name to display in user interfaces. It can be localized to create multilingual displays. The value of the `DisplayName` attribute does not need to be unique.

3.2.4 Description

The optional `Description` attribute is a string of printable characters that provides a longer description of the node. This attribute may be localized.

3.2.5 ValueType

This required attribute determines the datatype of the `Value` attribute and its restricting attributes. A list of all allowed values for this attribute is shown in Table 5.

Table 6 illustrates the effect of the `ValueType` attribute on the datatype of the `Value` attribute and its restricting attributes. The datatypes are XML Schema datatypes [21]. Attributes listed with the n/a datatype should not be present in the node.

3.2.6 Value

The `Value` attribute represents the value of the node. It is required if the `ValueType` attribute has a value other than “None”. The values of this attribute may be localized if the `ValueType` attribute is equal to “Multistate”, “Boolean”, or “String”.

Table 3: Attribute summary [16]

Attribute Identifier	**Datatype**	**Array**	**Enum-erated**	**Local-izable**	**Presence**
NodeType	string	No	Yes	No	Required
NodeSubtype	string	No	No	Yes	Optional
DisplayName	string	No	No	Yes	Optional
Description	string	No	No	Yes	Optional
ValueType	string	No	Yes	No	Required
Value	varies (see 3.2.5)	No	No	Yes	Required[1]
Units	string	No	Yes	Yes	Required[2]
Writable	boolean	No	No	No	Required[1]
InAlarm	boolean	No	No	No	Optional
Minimum	varies (see 3.2.5)	No	No	Yes	Optional
Maximum	varies (see 3.2.5)	No	No	Yes	Optional
Resolution	varies (see 3.2.5)	No	No	Yes	Optional
MinimumLength	nonNegativeInteger	No	No	No	Optional[3]
MaximumLength	nonNegativeInteger	No	No	No	Optional[3]
IsMultiLine	boolean	No	No	No	Optional
Attributes	string	Yes	No	No	Required
WritableValues	string	Yes	No	Yes	Required[4]
PossibleValues	string	Yes	No	Yes	Required[5]
Overridden	boolean	No	No	No	Optional
ValueAge	double	No	No	Yes	Optional
Aliases	string	Yes	No	No	Required[6]
Children	string	Yes	No	No	Optional
Reference	string	No	No	No	Present[7]
HasHistory	boolean	No	No	No	Required[1]
SinglyWritableLocales	string	Yes	No	No	Present[8]
HasDynamicChildren	boolean	No	No	No	Optional

[1] if `ValueType` is not equal to "None"
[2] if `ValueType` is `Real` or `Integer`
[3] only present if `ValueType` is "String"
[4] if `ValueType` is equal to "Multistate" or "Boolean" and `Writable` is true
[5] if `ValueType` is equal to "Multistate" or "Boolean"
[6] if there are reference nodes referring to this node
[7] if and only if the node is a reference node
[8] if and only if `ValueType` is equal to "String" and `Writable` is true

Table 4: Allowed values for the `NodeType` attribute

Value	**Description**
“Unknown”	For data that originates in another source and no type information is known
“System”	For an entire mechanical system
“Network”	For a communication network
“Device”	For a physical device
“Functional”	For system components or logical components
“Organizational”	For organizational classification like department, working group, etc.
“Area”	For geographical classification like building, floor, etc.
“Equipment”	For a single piece of equipment
“Point”	For a single data point
“Collection”	As container for grouping other nodes
“Property”	For data that logically belongs to the parent node
“Other”	For everything else

Table 5: Allowed values for the `ValueType` attribute

Value	**Description**
“None”	Has to be used when the node does not have a value
“String”	For human readable character strings
“OctetString”	For arbitrary binary data
“Real”	For floating point values
“Integer”	For integer values
“Multistate”	For choices between an arbitrary number of named states
“Boolean”	For choices between exactly two named states
“Date”	For calendar dates
“Time”	For the time of a day
“DateTime”	For an exact moment in time (specifying both date and time)
“Duration”	For time spans

Table 6: Effect of the `ValueType` attribute [16]

`ValueType` **Value**	`Value` **Datatype**	`Minimum` **Datatype**	`Maximum` **Datatype**	`Resolution` **Datatype**
“None”	n/a	n/a	n/a	n/a
“String”	string	n/a	n/a	n/a
“OctetString”	base64Binary	n/a	n/a	n/a
“Real”	double	double	double	double
“Integer”	integer	integer	integer	integer
“Multistate”	string	n/a	n/a	n/a
“Boolean”	string	n/a	n/a	n/a
“Date”	date	date	date	integer[1]
“Time”	time	time	time	double[2]
“DateTime”	dateTime	dateTime	dateTime	double[2]
“Duration”	double[2]	double[2]	double[2]	double[2]

[1] in days
[2] in seconds

3.2.7 Units

This enumerated attribute determines the unit for the `Value` attribute of the node. It has to be present if the `ValueType` of the node is equal to “Integer” or “Real” but it may have the value of “no-units”. For other values of the `ValueType` attribute it may optionally be present. The allowed values for this attribute are defined by the ASN.1 production for `BACnetEngineeringUnits` in Clause 21 of [3]. However, it is possible to extend this attribute to support customized units, too.

The value of this attribute can be obtained in two ways. If the `canonical` service option (see Section 3.3.1) is true, then the value of this attribute has to be one of the enumeration identifiers defined by the standard mentioned before. Otherwise the value is an arbitrary string that may be localized.

3.2.8 Writable

The `Writable` attribute has to be present if and only if the `ValueType` attribute has a value other than “None”. It indicates whether or not the `Value` attribute of the node can be written through web services.

3.2.9 InAlarm

This optional attribute indicates if the node is “in alarm” or not. The meaning of “in alarm” is not defined by the BACnet/WS standard.

3.2.10 Minimum

The `Minimum` attribute determines the minimum value of the `Value` attribute. The datatype of this attribute depends on the `ValueType` attribute of this node (see Table 6).

3.2.11 Maximum

The `Maximum` attribute determines the maximum value of the `Value` attribute. The datatype of this attribute depends on the `ValueType` attribute of this node (see Table 6).

3.2.12 Resolution

The `Resolution` attribute determines the smallest possible change of the `Value` attribute. The datatype of this attribute depends on the `ValueType` attribute of this node (see Table 6).

3.2.13 MinimumLength

This optional attribute defines the minimum length, in characters, for the `Value` attribute. It is only present if the `ValueType` attribute is equal to "String".

3.2.14 MaximumLength

This optional attribute defines the maximum length, in characters, for the `Value` attribute. It is only present if the `ValueType` attribute is equal to "String".

3.2.15 IsMultiLine

The `IsMultiLine` attribute indicates whether or not the `Value` attribute is capable of containing multiple lines of text. This attribute is optional and should only be present if the `ValueType` attribute is equal to "String". Lines are separated by the ANSI X3.4 control character newline (0x0A).

3.2.16 Attributes

This required attribute is an array of character strings. It contains the names of all attributes that are present in this node.

3.2.17 WritableValues

The optional `WritableValues` attribute is an array of character strings containing all string values that may be written to the `Value` attribute when the `ValueType` is equal to "Multistate" or "Boolean".

3.2.18 PossibleValues

The optional `PossibleValues` attribute is an array of character strings. It contains all possible string values for the `Value` attribute of a node whose `ValueType` is equal to "Multistate" or "Boolean". If the `ValueType` is equal to "Boolean" the first entry in the array relates to true and the second to false.

3.2.19 Overridden

This optional attribute indicates whether or not the value of the `Value` attribute has been overridden. For data points representing physical inputs, the `Overridden` attribute may be used to denote that the `Value` attribute does not react to changes of the physical input anymore.

3.2.20 ValueAge

The optional `ValueAge` attribute stores the time, in seconds, since the `Value` attribute was last successfully updated in the server.

3.2.21 Aliases

This attribute is an array of character strings containing the paths of all reference nodes referring to this node. It has to be present if and only if there exists at least one reference node referring to this node.

3.2.22 Children

The `Children` attribute is an array of character strings. It contains the identifiers for the children of this node on a given path. These identifiers can be used to build new paths to child nodes. Child identifiers do not start with a forward slash ('/') character. Hence, a forward slash ('/') character has to be inserted between the original path and the child identifier when constructing the new path.

3.2.23 Reference

This attribute is present if and only if the node is a reference node. Its value is the path to the referent node.

3.2.24 HasHistory

The `HasHistory` attribute is required if the `ValueType` attribute has a value other than "None". It indicates whether or not historical records are available for this node.

3.2.25 SinglyWritableLocales

This attribute is present if and only if the `ValueType` attribute is equal to "String" and the `Writable` attribute is true. It is an array containing locales that can be used with the `writeSingleLocale` service option (see Section 3.3.1) to set individual localized string values for a node. All of its elements have to be contained in the result set of the `getSupportedLocales` service (see Section 3.3.12).

3.2.26 HasDynamicChildren

The `HasDynamicChildren` attribute indicates that the node has a dynamic set of children. Clients should assume that the children nodes may change at any time if this attribute is true. Otherwise clients can assume that changes will only occur after reconfiguration of the server.

Table 7: Service options [16]

Option Name	Datatype	Default Value
“readback”	boolean	false
“errorString”	string	`"? error_number error_message"`
“errorPrefix”	string	`""` (empty string)
“locale”	string	Depends on server configuration
“writeSingleLocale”	boolean	false
“canonical”	boolean	false
“precision”	nonNegativeInteger	6
“noEmptyArrays”	boolean	false

3.3 Services

The BACnet/WS standard defines web services that are used to access and manipulate data in a server. Some services allow to specify service options that manipulate their behavior or their return values.

The specified service options are described in the next section. Sections 3.3.2 to 3.3.12 depict the services that are defined by the BACnet/WS specification [16].

3.3.1 Service Options

Service options are used to modify the behavior of services. They are specified as character strings of form:

```
[option_name[=option_value][;option_name[=option_value]]...]
```

where '`[]`' indicates an optional element and '`...`' indicates iteration of the previous element. Multiple service options are combined into a single string separated by semicolons (';'). Hence, option values must not contain semicolons (';').

For `boolean` service options, the option value can be omitted. In this case the service option is assumed to be true. Option names may be specified more than once in a string. However, only the last occurrence of an option name will be considered.

Table 7 shows the defined service options and their respective XML Schema datatype [21]. If an option name is omitted in the option string the default value is assumed.

The `readback` service option forces services that set values to read back the values just written and return the results.

Specifying the `errorString` service option overrides the default error string. By default error strings are encoded as `"? error_number error_message"`. The error number is a standardized number defined in the BACnet/WS specification [16] whereas the error message is a human readable message whose content is of local concern.

Table 8: Effects of the `locale` and `canonical` service options [16]

Attribute Identifier	**ValueType**	**Effect of `local` service option**	**Effect of `canonical` service option**
Value	"String" "Multistate"	Different values may be returned or accepted for different locales	Ignored
	"Real" "Integer" "Date" "Time" "DateTime" "Duration" "Boolean"	The value is formatted according to the server configuration for the requested locale	Overrides `locale` service option
	"OctetString"	Ignored	Ignored
DisplayName Description WritableValues PossibleValues		Different values may be returned for different locales	Ignored
ValueAge Minimum Maximum Resolution Units		The value is formatted according to the server configuration for the requested locale	Overrides `locale` service option

The `errorPrefix` service option can be used to prefix the error string. Regardless of the use of the `errorString` option the resultant error string is the error prefix followed by the error string.

If the `locale` service option is defined the format of date/time values, string values, numbers, and units for the given locale should be used by the server. Locales are specified by language tags as described in RFC 3066 [20]. However, the language tag must match exactly one of the strings returned by the `getSupportedLocales` service (see Section 3.3.12). For unsupported locales an invalid locale error is returned.

For nodes with a `ValueType` of "String" the `Value` attribute is set in all locales by default. The `writeSingleLocale` service option allows clients to set the value only for the given locale. If the `locale` option is not specified the default locale is assumed. Whether the `locale` option is specified or the default locale is assumed, the locale to be set has to be an element of the `SinglyWritableLocales` attribute of the node. Otherwise an invalid locale error is returned.

The `canonical` service option is used to override certain localized string formats. The canonical form is a locale-independent standardized form especially suitable for machine processing. Table 8 summarizes the effects of the `locale` and `canonical` service options on accessed attributes.

The `precision` service option defines the number of digits after the decimal point for floating point values of any requested attribute. Floating point values should be rounded and not truncated.

When the noEmptyArrays service option is specified the server should return an error string rather than empty arrays.

3.3.2 getValue Service

The getValue service allows clients to retrieve the value for any single attribute of a single node. Its return value is always a single string. When the value of an array attribute, such as the Children attribute, is requested, the elements of the array are concatenated into a single string separated by semicolons (';'). If an array of strings should be returned rather than a single string, the getArray or getArrayRange service (see Sections 3.3.5 and 3.3.6) can be used instead.

Table 9 shows the parameters for the getValue service with their corresponding XML Schema datatype [21]. The Path parameter specifies the node and attribute to be read. If the path's attribute part is empty, the Value attribute is assumed. The Options parameter may be used to modify the behavior of this service as discussed in Section 3.3.1. If no error occurs the result is returned as a single string containing the requested attribute. Otherwise an appropriate error string is returned by the server.

Table 9: Parameters and result of the getValue service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
Path	string	The path as defined in 3.1.1
Result	string	The requested value as single string

3.3.3 getValues Service

The getValues service can be used to read more than one attribute at a time. Instead of specifying a single path, an array of paths is passed to the server. The server processes the entries in the Paths parameter the same way as the getValue service starting with the first element of the array. All single return strings are entered into the return array. If the server is not able to process the Paths parameter, or if the Paths parameter is of zero length, the return value is an array of size one containing the error string. Table 10 summarizes the parameters of the getValues service.

3.3.4 getRelativeValues Service

The getRelativeValues service is an optional service that is similar to the getValues service. Instead of specifying absolute paths, the getRelative-Values service takes a single base path to a node or attribute, and a list of relative sub paths. The base path has to end with a node or attribute identifier. However, the BasePath parameter might be an empty string. Then each sub path becomes an absolute path. In order to form complete paths, the sub paths have to begin with a path delimiter ('/' or ':').

Table 10: Parameters and result of the `getValues` service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
Paths	array of string	An array containing the paths as defined in 3.1.1
Results	array of string	An array containing the requested values as single strings

This service processes the entries of the `Paths` parameter beginning with the first element in the array. A complete path is constructed through concatenation of the base path and a sub path. The complete paths are evaluated separately as if the `getValue` service were called for each of them. Return strings are entered into the return array at the corresponding positions. In Table 11 the parameters of the `getRelativeValues` are shown.

Table 11: Parameters and result of the `getRelativeValues` service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
BasePath	string	The base path as defined in 3.1.1
Paths	array of string	An array containing the relative sub paths
Results	array of string	An array containing the requested values as single strings

3.3.5 getArray Service

Clients can request array attributes using the optional `getArray` service. In opposite to the `getValue` service the array attribute is returned as an array of strings and not as single string. The format of the array elements depends on the attribute's datatype and the service options. If the array attribute is empty, an empty result array is returned unless the `noEmptyArrays` service option is true. In this case and in case of any other error occuring an array of size one containing an appropriate error string is returned. The parameters of the `getArray` service are listed in Table 12.

If the requested array is too large to be returned, the `getArrayRange` service (see Section 3.3.6) can be used to retrieve only a portion of the array. The `getArray` service should be provided together with the `getArraySize` and `getArrayRange` services.

Table 12: Parameters and result of the `getArray` service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
Path	string	The path to the array attribute as defined in 3.1.1
Result	array of string	The requested array attribute

3.3.6 getArrayRange Service

If only a portion of an array attribute should be retrieved, clients can use the optional `getArrayRange` service. The format of the array elements depends on the attribute's datatype and the service options.

Table 13 summarizes the parameters of the `getArrayRange` service. The `Index` parameter is a non negative integer defining the beginning of the requested array portion. Array index zero corresponds to the first entry of the array, index one to the second entry, and so on. The `Count` parameter specifies the number of entries to be read, starting at the `Index` parameter. A count of zero is invalid. If the `Count` parameter exceeds the number of available array entries, the result has to be truncated. This service returns the specified portion of the array if no error occurs, otherwise an array containing a single error string is returned.

Table 13: Parameters and result of the `getArrayRange` service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
Path	string	The path to the array attribute as defined in 3.1.1
Index	nonNegativeInteger	The array index to start from
Count	nonNegativeInteger	The number of entries to retrieve, starting at the `Index` parameter
Result	array of string	The requested portion of the array attribute

3.3.7 getArraySize Service

The optional `getArraySize` service allows clients to retrieve the number of elements of an array attribute. Table 14 shows the parameters of this service. The `Path` parameter specifies the array attribute. If successful, this service returns the non negative size of the array attribute encoded as string. Otherwise an appropriate error string is returned.

Table 14: Parameters and result of the `getArraySize` service

Parameter Name	**Datatype**	**Description**
Options	string	The option string as defined in 3.3.1
Path	string	The path to the array attribute as defined in 3.1.1
Result	string	The size of the array attribute

3.3.8 setValue Service

The `setValue` service can be used to write a new value to the `Value` attribute of a single node. Note that only the `Value` attribute is writable and that the node's `Writable` attribute has to be true.

This service always returns a single string. If successful, an empty string is returned unless the `readback` service option is specified. When the `readback` service option is true, the server tries to read back the written value and returns it to the client if successful. In case of any error occuring during execution of the service an error string is returned.

The `Value` parameter contains the new value that has to be written. If the node's value is localizable, i.e., its value type is equal to "Multistate", "Boolean", or "String", the `locale` and `writeSingleLocale` service options may be used to set the value only for the specified locale. Otherwise the value is set for all locales. The parameters of this service are summarized in Table 15.

Table 15: Parameters and result of the `setValue` service

Parameter Name	**Datatype**	**Description**
Options	string	The option string as defined in 3.3.1
Path	string	The path as defined in 3.1.1
Value	string	The new value to be set
Result	string	An empty string (`readback` = false) or the written value as single string (`readback` = true)

3.3.9 setValues Service

If more than one value has to be set, the optional `setValues` service can be applied to set the `Value` attribute of multiple nodes. This service is similar to the `setValue` service except that it takes multiple paths and values as parameters rather than a single path and value. It also returns multiple results in form of a non-empty array of strings. The parameters of the `setValues` service are shown in Table 16.

Each pair of path and value is processed separately in the same way as the `setValue` service starting with the first entries of the arrays. The return string is entered into the corresponding entry of the result array. If the `Paths` parameter can not be processed or is of zero length, or the same error occurs while processing each of the path value pairs, an array containing a single error string is returned.

Table 16: Parameters and result of the `setValues` service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
Paths	array of string	An array containing the paths as defined in 3.1.1
Values	array of string	An array containing the new values to be set
Results	array of string	An array containing single return strings

3.3.10 getHistoryPeriodic Service

Historical data of a node's `Value` attribute can be retrieved by the optional `getHistoryPeriodic` service. This service takes several parameters that identify the samples of interest. The `Start` parameter represents the point in time of the first sample. The time interval, in seconds, between any two samples is specified by the `Interval` parameter. An interval of length zero is invalid. The total number of samples to return is defined by the `Count` parameter. At least one sample has to be returned. A Client may request historical values with a specific sampling rate regardless of the actual sampling rate of the server. Hence, the server must be able to resample the data for the client's needs. The `ResampleMethod` parameter specifies the resample method to apply to the historical data if necessary. Table 17 lists the possible values for the `ResampleMethod` parameter.

If successful, this service returns an array containing the requested historical values, else an array containing a single error string is returned. The parameters of the `getHistoryPeriodic` service are summarized in Table 18.

3.3.11 getDefaultLocale Service

The `getDefaultLocale` service can be used to retrieve the default locale configured for the server. Its return value is the locale string for the configured default locale, or an emtpy string if there is no default locale. If any error occurs while processing the service request, an error string is returned. Table 19 shows the parameters of the `getDefaultLocale` service.

3.3.12 getSupportedLocales Service

Clients can use the `getSupportedLocales` service to get a list of all locales supported by the server. This service returns an array containing the locale strings of the supported locales, if successful. If localization is not supported by the server, an empty array is returned unless the `noEmptyArrays` service option is true. When any error occurs, an array of size one containing the error string

Table 17: Resample methods

Resample Method	Description
"interpolation"	Each data sample is determined by straight line interpolation between the actual sample before and the actual sample after the specified point in time
"average"	For each data sample the average of all collected samples within the specified time interval, centered on the returned sample time, is returned
"after"	For each data sample the closest actual sample at or after the specified point in time is returned
"before"	For each data sample the closest actual sample at or before the specified point in time is returned
"closest"	For each data sample the closest actual sample at, before, or after the specified point in time is returned
"default"	The most appropriate resample method is used (may be any proprietary method)

Table 18: Parameters and result of the `getHistoryPeriodic` service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
Path	string	The path as defined in 3.1.1
Start	dateTime	The point in time to start from
Interval	double	The time interval between the returned values in seconds
Count	nonNegativeInteger	The number of values to return
ResampleMethod	string	The resample method to apply to the historic data if necessary
Results	array of string	An array containing the historical values

Table 19: Parameters and result of the `getDefaultLocale` service

Parameter Name	Datatype	Description
Options	string	The option string as defined in 3.3.1
Result	string	The locale string for the configured default locale or an empty string (if there is no default locale)

is returned. The parameters of the `getSupportedLocales` service are listed in

Table 20.

Table 20: Parameters and result of the `getSupportedLocales` service

Parameter Name	**Datatype**	**Description**
Options	string	The option string as defined in 3.3.1
Results	array of string	An array containing the locale strings of all supported locales

4 Conclusion

Since BACnet/WS has emerged to a serious alternative solution for integration of the BAS into enterprise networks it is reasonable to take a look at the performance of BACnet/WS. Especially communication performance is a crucial factor that has to be considered because of long message formats and encoding of data. Unfortunately there have not been many studies on the communication performance of BACnet/WS so far. One of these studies tries to analyze the communication performance using the BACnet/WS `getValues` service [22].

The communication performance simulation in [22] is based on a remote monitoring system, i.e., client and server are located on different networks separated by an IP network. As automation system a Variable Refrigerant Flow (VRF) air-conditioning system [23] has been chosen. VRF air-conditioning systems typically consist of hundrets of air-conditioner units that have input/output points, such as sensors and fans. These air-conditioner units are interconnected by a proprietary VRF fieldbus. The BACnet/WS server is directly connected to this fieldbus and represents each input/output unit as BACnet object of a given type, such as `AnalogInput`, or `DigitalOutput`, as defined in Annex H of [16]. The BACnet/WS client accesses these objects using the `getValues` service which is transported over network using SOAP over HTTP. Service data is encoded using the XML format as defined in [24] and [21]. For a more realistic communication scenario constant bit rate background traffic has been introduced.

The main purpose of this research has been to find an optimal combination of the number of accesses and the number of objects per access under various background traffic conditions. At first, the single access time of the BACnet/WS `getValues` service has been analyzed for different numbers of objects. This simulation has shown that the influence of the background traffic varies depending on the number of objects that are read at once. The more objects have been read, the larger the observed influence of the background traffic on the access time has been. This phenomenon is caused by longer BACnet/WS messages as a result of reading more objects at once. In further consequence the total access time, i.e., the product of single access time and the number of accesses has been examined. The number of accesses is defined by the total number of objects divided by the number of objects read per access. Results have shown that in general, the more objects have been read per access, the shorter the total access time has become. However, the total access time has increased dramatically in case of higher background loads when reading many objects. More details about the simulation model and the results of this study can be found in [22].

References

[1] A. Malatras, A. Asgari, T. Bauge, and M. Irons. A service-oriented architecture for building services integration. *Journal of Facilities Management*, 6(2):132 – 151, 2008.

[2] *EN 14908: Open Data Communication in Building Automation, Controls and Building Management - Building Network Protocol*, 2005. LonWorks.

[3] International Organization for Standardization. *ISO 16484-5: Building automation and control systems - Part 5: Data communication protocol*, 2008. BACnet.

[4] International Organization for Standardization. *ISO 14543-x: Information technology - Home Electronic System (HES) Architecture.* KNX.

[5] J. Bai, H. Xiao, X. Yang, and G. Zhang. Study on integration technologies of building automation systems based on web services. In *ISECS International Colloquium on Computing, Communication, Control, and Management, 2009 (CCCM 2009)*, volume 4, pages 262 – 266, 8-9 2009.

[6] S. Wang, Z. Xu, H. Li, J. Hong, and W. Shi. Investigation on intelligent building standard communication protocols and application of IT technologies. *Automation in Construction*, 13(5):607 – 619, 2004. Current IT Research and Development in the Construction Industry of China.

[7] World Wide Web Consortium. *Efficient XML Interchange (EXI) Format 1.0*, December 2009. http://www.w3.org/TR/exi/ (acc. July 1, 2010).

[8] F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, 1(1):62 – 70, February 2005.

[9] F. Jammes, A. Mensch, and H. Smit. Service-Oriented Device Communications Using the Devices Profile for Web services. In *21st International Conference on Advanced Information Networking and Applications Workshops, 2007 (AINAW 2007)*, volume 1, pages 947 – 955, 2007.

[10] OPC Foundation. *OPC Unified Architecture, Part 1: Overview and Concepts*, February 2009. http://www.opcfoundation.org (acc. July 1, 2010).

[11] W. Mahnke, S. Leitner, and M. Damm. *OPC Unified Architecture.* Springer Berlin Heidelberg, 2009. ISBN: 978-3-540-68899-0.

[12] T. Hannelius, M. Salmenpera, and S. Kuikka. Roadmap to adopting OPC UA. In *6th IEEE International Conference on Industrial Informatics, 2008 (INDIN 2008)*, pages 756 – 761, July 2008.

[13] Organization for the Advancement of Structured Information Standards. *oBIX 1.0 Committee Specification 01*, December 2006. http://www.obix.org (acc. July 1, 2010).

[14] Internet Engineering Task Force. *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1*, June 1999. http://www.ietf.org/rfc/rfc2616.txt (acc. May 28, 2010).

[15] Internet Engineering Task Force. *RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax*, January 2005. http://www.ietf.org/rfc/rfc3986.txt (acc. May 27, 2010).

[16] American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc. *ANSI/ASHRAE Addendum c to ANSI/ASHRAE Standard 135-2004*, October 2006. http://www.bacnet.org (acc. July 1, 2010).

[17] Web Services Interoperability Organization. *Basic Profile Version 1.0*, April 2004. http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html (acc. May 30, 2010).

[18] Internet Engineering Task Force. *RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2*, August 2008. http://www.ietf.org/rfc/rfc5246.txt (acc. June 30, 2010).

[19] Internet Engineering Task Force. *RFC 2818 - HTTP Over TLS*, May 2000. http://www.ietf.org/rfc/rfc2818.txt (acc. May 29, 2010).

[20] Internet Engineering Task Force. *RFC 3066 - Tags for the Identification of Languages*, January 2001. http://www.ietf.org/rfc/rfc3066.txt (acc. May 17, 2010).

[21] World Wide Web Consortium. *XML Schema Part 2: Datatypes Second Edition*, October 2004. http://www.w3.org/TR/xmlschema-2/ (acc. May 25, 2010).

[22] C. Ninagawa, T. Sato, and Y. Kawakita. Communication performance simulation for object access of BACnet Web Service building facility monitoring systems. In *IEEE International Conference on Emerging Technologies and Factory Automation, 2008 (ETFA 2008)*, pages 701 – 704, September 2008.

[23] W. Goetzler. Variable Refrigerant Flow Systems. In *ASHRAE Journal*, volume 49, pages 24 – 31, April 2007.

[24] World Wide Web Consortium. *XML Schema Part 1: Structures Second Edition*, October 2004. http://www.w3.org/TR/xmlschema-1/ (acc. May 25, 2010).