FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Seamless Integration of BACnet Devices into oBIX

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software and Information Engineering

eingereicht von

Robert Horvath
Matrikelnummer 1025519

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Dipl.-Ing. Markus Jung
Mitwirkung: Ao. Univ. Prof. Dr. Wolfgang Kastner

Wien, 04.08.2015

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Seamless Integration of BACnet Devices into oBIX

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Robert Horvath
Registration Number 1025519

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Dipl.-Ing. Markus Jung
Assistance: Ao. Univ. Prof. Dr. Wolfgang Kastner

Vienna, 04.08.2015

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Robert Horvath
Hauptstrasse 200, 7212 Forchtenstein

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Abstract

The Internet of Things (IoT) is the idea of a world with billions of interconnected devices. Buildings can be made smarter and more efficient by sharing information intelligently. Therefore it is desirable to integrate already widely deployed Building Automation Systems, such as BACnet, with the IoT.

Building Automation Systems with various communication technologies are already widely deployed. Several approaches exist to establish connectivity between them and the Internet of Things, such as gateway systems or field-to-IP routers.

To fulfill the vision of the Internet of Things, the connected devices need to be able to interoperate with one another, but the various Building Automation Systems usually communicate in their own custom protocols.

Several potential solutions to the interoperability problem have been proposed. One such approach is oBIX, a standard for building exchange. It defines a platform-independent model for device information and a standardized REST Web service interface for communication.

In this thesis, a generic mapping for BACnet devices to oBIX objects is described, as well as a way to automatically discover BACnet devices. This eases the provisioning and commissioning of BACnet devices for oBIX servers and thereby allowing them to take part in the Internet of Things.

As a proof of concept, the described techniques have been implemented in an open source gateway system called IoTSyS, resulting in less effort required to integrate BACnet devices in the IoT through the gateway.

Kurzfassung

Das Internet der Dinge (Internet of Things, IoT) verfolgt die Idee einer Welt mit Milliarden von miteinander verbundenen Geräten. Gebäude können durch Teilen von Informationen effizienter und intelligenter gemacht werden, daher ist es wünschenswert, bereits weit verbreitete Gebäudeautomationssysteme, wie BACnet, in das IoT zu integrieren.

Gebäudeautomationssysteme mit verschiedenen Kommunikationstechnologien sind bereits weit verbreitet. Verschiedene Ansätze zur Herstellung von Konnektivität zwischen ihnen und dem IoT existieren bereits, wie Gateway-Systeme oder Field-to-IP-Routers.

Um die Vision des IoT zu erfüllen, müssen die verbunden Geräte auch miteinander interagieren können, jedoch verwenden verschiedene Gebäuteautomationssysteme üblicherweise eigene Protokolle zur Kommunikation.

Mehrere Lösungsansätze für das Interaktionsproblem wurden vorgestellt. Ein Ansatz ist oBIX, ein Standard zum Gebäudeinformationsaustausch. oBIX definiert ein platformunabhängiges Modell für Gerätinformationen und ein standardisiertes REST Web Service Interface zur Kommunikation.

In dieser Arbeit wird eine Methode zur generischen Abbildung von BACnet Geräten auf oBIX vorgestellt, sowie eine Methode zur automatischen Entdeckung von BACnet Geräten. Dies vereinfacht die Inbetriebnahme von BACnet Geräten in oBIX Servern und erlaubt es ihnen am IoT teilzunehmen.

Die beschriebenen Techniken wurden in einem Open Source System namens IoTSyS implementiert und evaluiert. Der benötigte Aufwand zum Einbinden von BACnet Geräten in das IoT durch das Gateway-System konnte dadurch reduziert werden.

Contents

CHAPTER 1

Introduction

1.1 Building Automation System Trends

The field of Building Automation Systems (BAS) is currently experiencing major changes. According to research [4], the market is expected to grow from $72 billion in 2012 to $146 billion by 2021.

Currently, about 23% of global electrical energy consumption is due to commercial buildings [4]. HVAC (Heating, Ventilation, Air Conditioning) systems are a particularly significant factor in building energy consumption, responsible for about 50% of building energy consumption, and 20% of total consumption in the US [11]. Building stock is growing rapidly, so aggressive energy goals have been set.

BAS are part of current trends transforming the building industry. They provide a way to make buildings more energy efficient. Four general approaches for reducing energy consumption in buildings have been identified [3]: Energy awareness, elimination of stand-by consumption, scheduling of flexible tasks, and adaptive control of electrical appliances. The most common ways BAS help to cut energy costs include scheduling (turning equipment on or off depending on the time of day) and preventing turning on of equipment when it is unnecessary (e.g. turning off cooling when the outside air falls below a certain temperature). Additional inputs (e.g. occupancy) can be used to control operational parameters of lighting and HVAC systems.

The technology used in building automation is also changing. Non-IP communication based on twisted pair, power line or radio frequency technologies is prevalent among BAS [7]. Nowadays, the industry is moving towards IP-based systems. Building automation controls and field devices are being fitted with IP capability [4]. This makes it possible to use the existing IT-infrastructure and protocols for building data networks and will lead to better standardization. The advances made in IT in the last decade also provides improved network security and reliability for BAS [4].

1.2 The Internet of Things

The Internet of Things (IoT) is a vision of billions of devices connected through the Internet. These devices may come from many different domains, including product supply chains, telecommunications, home and building automation and smart grid infrastructures.

Integrating building automation devices with the IoT opens up new possibilities. For example, device maintenance could be made more efficient with devices that report that they need maintenance. Often such errors are only indicated by a blinking light on the device, but instead a device connected to the IoT could automatically place an order for a spare part or arrange a service appointment and take the availability of the home owner or responsible system operator into account using information stored in an online calendar [7]. Smart and sustainable building operation can be realized, for example with automatic HVAC settings based on occupancy, external temperature, the number of people in room, or scheduled sessions and more.

Another possibility enabled by the Internet of Things is the smart grid, a new kind of intelligent power system. By using automation and IT technologies, the operation and management of the electrical power grid can be optimized. It can improve the transmission and distribution on our power infrastructure, as well as improve the safety and reliability of the grid [9]. Sensors and power equipment in the IoT form a real-time network that allows to improve the efficiency of the integrated power grid. The principal characteristics of a smart grid include self-healing, mutual operation and participation of the users, perfect electricity quality, distributed generations and demand response, sophisticated market and effective asset management [15].

For the interconnection of the devices on the IoT, a trend towards Web service based communication, resting on HTTP and XML can be seen [8].

1.3 IoT Gateways

There are multiple possible approaches to integrating building automation in the Internet of Things. The main approaches regarding connectivity [7] are:

1. A centralized server using a single IP address on the IP backbone of a BAS can act as a central gateway. The BAS specific protocols are hidden by the server.

2. Field-to-IP-routers can be used as proxies. Native IP support of field devices is imitated by emulating IP addresses for the connected field devices.

3. Field devices can directly be connected to the IoT by using IP as network layer transporting higher-level protocols. This comes with a high cost of having to implement an IP stack within every single device.

4. Web service interfaces can be emulated with separate IPv6 addresses at either a centralized server or at a field-to-IP-router acting as a transparent gateway.

An arguably bigger challenge than connectivity is the problem of interoperability as existing BAS use their own custom protocols to communicate. Different approaches to solving this problem already exist for the centralized server integration scenario, like BACnet/WS, oBIX or OPC UA. These standards propose a centralized server that offers platform independent interfaces like Web services or RESTful APIs to access the underlying BAS [7].

1.4 Goals, Methodology and Structure of the Thesis

Large BACnet networks may consist of hundreds of BACnet devices. In order to make them accessible through an oBIX server to the IoT, these devices need to be mapped to oBIX objects. Manually performing the mapping requires considerable human effort.

The goal of this thesis is to make the provisioning and commissioning of BACnet devices for oBIX servers easier. Provisioning BACnet devices can be automated using BACnet Remote Management Services to automatically discover devices. In order for BACnet devices to be accessible in oBIX servers, they need to be mapped to oBIX objects, so a generic mapping of BACnet devices to oBIX objects is described.

In the following section, the current state of the art regarding BACnet, the IoT and relevant technologies is discussed.

A mapping of BACnet devices to oBIX objects is described in section 3, using several BACnet properties to create oBIX representations. Read and write requests are also mapped to create fully interactive oBIX objects that transparently forward requests to the BACnet devices.

In section 4, a way to automatically provision BACnet devices in a BACnet network using BACnet Remote Management Services for autodiscovery is described.

The presented techniques have been implemented in an open source IoT gateway project called “IoTSyS” [6]. The implementation is detailed and an evaluation is given in section 5.

CHAPTER 2

State of the Art

2.1 Internet of Things

The idea of the Internet of Things (IoT) is to connect things - such as sensors, actuators, mobile phones or Radio-Frequency Identification (RFID) tags - and through unique addressing schemes let them be able to interact with each other [2]. It is currently exponentially growing towards an ecosystem of tens of billions of smart things [16].

The ongoing miniaturization and cost reduction of electronic devices makes it possible to create smart things - everyday physical objects enhanced by a small electronic device that provides a computational component and connectivity to the Internet. Equipping many objects in the world with such devices would lead to an Internet of Things.

The IoT is a transformative technology that will have a large impact on the everyday-life of its users. For private users, the IoT will be noticeable in home automation, assisted living, e-health, enhanced learning and more. For business users, the IoT will be applicable in automation and industrial manufacturing, logistics, business/process management and intelligent transportation of people and goods [2].

The need for IPv6 and related new protocols

The number of humans currently using the Internet is estimated to have recently surpassed three billion, up from 2.7 billion at the end of 2013 [14]. The number of objects connected to the Internet has already surpassed the number of human Internet users and is expected to expand to 20 to 50 billion smart things [16].

The most common protocol currently used at the network layer, the Internet Protocol version 4 (IPv4), was not designed with the Internet of Things in mind and is limited to only about 4 billion addresses. This limited number of addresses is already being exhausted, so to meet the growing demand for addresses IPv6 has been adopted by IANA.

IPv6 provides a much larger address space, about 3.4×10^{38} addresses. This corresponds to over 6.67×10^{17} unique addresses per square millimeters of Earth surface, thereby having a large enough address space to support a globally connected Internet of Things.

The Internet Protocol is already used by many devices such as printers, sensors, lighting systems, mobile phones, TVs, and more. IPv6-related standards designed for the IoT, such as 6LoWPAN, CoAP and CoRE, have also enabled highly constrained devices to become IP compliant [16].

2.2 CoAP

The Constrained Application Protocol (CoAP) is a specialized Web transfer protocol. It is designed for use in constrained environments, such as constrained nodes (e.g., microcontrollers with small amounts of RAM and ROM) and networks (low-power, high packet error rates, e.g. 6LoWPAN). Constrained nodes may be low power sensors, switches, or similar components that need to be remotely monitored and controlled. Targeted application areas are machine-to-machine (M2M) applications, including building automation [12].

A design goal for CoAP is that it should be easily translatable to HTTP for integration with the Web while meeting specialized requirements for M2M applications such as multicast support and very low overhead [12].

CoAP uses a client/server interaction model very similar to HTTP. However, for its intended use in M2M communication, nodes will typically assume both client and server roles. A request contains a request code, specifying an action (`GET`, `PUT`, `POST`, `DELETE` - equal to HTTP's actions) to be performed on a resource identified by a URI.

A difference to HTTP is the used transport layer protocol. Whereas HTTP requires a reliable, connection-oriented communication, mostly using TCP, CoAP uses a datagram-oriented transport like UDP. Transport over UDP is more efficient than TCP for non-reliable, asynchronous or group communication. Optional reliability is supported by CoAP, in the form of confirmable messages and acknowledgments.

In the OSI model, CoAP is positioned above the transport layer, as seen in Figure 2.1. CoAP can be viewed as following a two-layer approach, with a messaging layer, dealing with the asynchronous communication and UDP, and a request/response layer above, dealing with interactions using method and response codes. But these layers are part of a single CoAP protocol, and requests/responses are just features of the CoAP header.

CoAP is tailored to a polling interaction model, but a protocol extension allows clients to "observe" resources on a CoAP server. Clients can specify the "observe" option on GET requests to receive the updated state of the requested resource whenever it changes, eliminating the need for constant polling. A best-effort approach is used for sending new representations to clients, and provides eventual consistency between the state observed by each client and the actual resource state at the server [5].

The low overhead, multicast support, and observing of resources makes CoAP a good fit for a protocol for the Internet of Things.

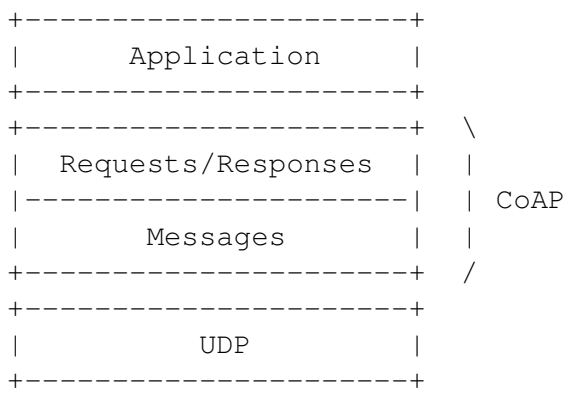


Figure 2.1: Abstract layering of CoAP [12]

2.3 oBIX

As M2M and the IoT become ever more prevalent, new standards are needed to allow the machines to communicate autonomously and effectively with each other. oBIX (Open Building Information eXchange) is a standard developed by the Organization for the Advancement of Structured Information Standards (OASIS). It uses existing standard technologies like XML, HTTP and URIs to define a standard Web service protocol to enable such communication for building control systems [10].

oBIX object model

oBIX has a simple object model, based on a small set of object types corresponding to primitive values types (e.g. integer, string, real) Figure 2.2 shows a summary of the object model. The boxes show the object types, as well as a list of attributes (or "facets") the object supports.

There are 17 generic object types: `obj`, `str`, `int`, `real`, `bool`, and more. The base object type is "`obj`". Every other object type is derived from "`obj`". Therefore, every object in oBIX inherits the base facets `name`, `href`, `is`, `null`, `icon`, `displayName`, `display`, `writable` and `status`. The object model can be extended by the use of "contracts", which define a subclassing relation between object types.

This generic object model is able to represent information from diverse M2M systems through a standardized XML format. An example of an oBIX object in XML syntax can be seen in Figure 2.3.

Everything in oBIX is modeled as an object, and to identify objects, oBIX uses URIs (Uniform Resource Identifiers). URIs are the standard way of identifying resources on the Web and are already well established.

The oBIX specification also provides a normalized representation for common concepts in BAS: points, histories and alarms. Points represent a single scalar value and its status. They usually map directly to sensors, actuators or setpoints. Each value is a separate object in oBIX. Histories and alarms are available as oBIX services.

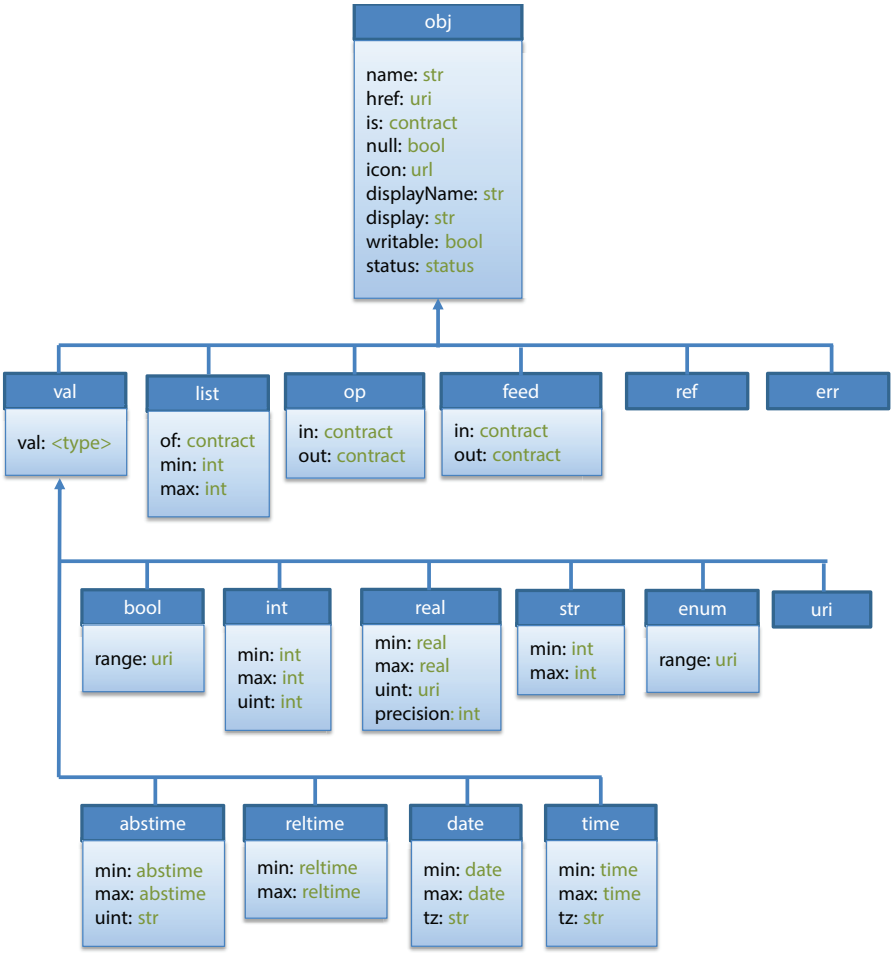


Figure 2.2: oBIX object model [8] [10]

oBIX also allows to represent available operations on objects. Operations are also represented as objects - the "`op`" object defines such operations. An operation takes an oBIX object as input and returns an oBIX object. The type of allowed input and output objects are specified through contracts using the operation's "in" and "out" facets.

Contracts

The "`is`" facet specifies the contracts that the object adheres to by their URIs. Multiple contracts can be specified by separating them with whitespace. Contracts are a mechanism of inheritance and allow "subclassing" in the oBIX object model. A contract itself is specified as an object, and can be expressed in XML syntax.

```
<obj href="http://bradybunch/people/Mike-Brady/">
  <obj name="fullName">
    <str name="first" val="Mike"/>
    <str name="last" val="Brady"/>
  </obj>
  <int name="age" val="45"/>
  <ref name="spouse" href="/people/Carol-Brady"/>
  <list name="children">
    <ref href="/people/Greg-Brady"/>
    <ref href="/people/Peter-Brady"/>
    <ref href="/people/Bobby-Brady"/>
    <ref href="/people/Marsha-Brady"/>
    <ref href="/people/Jan-Brady"/>
    <ref href="/people/Cindy-Brady"/>
  </list>
</obj>
```

Figure 2.3: Example oBIX XML [10]

A contract defines the child objects that an object implementing the contract contains. An object conforming to the contract can override a child object's value by explicitly specifying an object with the same name as the object to be overwritten from the contract. If an object from the contract isn't specified, it implicitly inherits the object, using the default value from the contract.

```
<obj href="/def/television">
    <bool name="power" val="false"/>
    <int name="channel" val="2" min="2" max="200"/>
</obj>
<obj href="/livingRoom/tv" is="/def/television">
    <int name="channel" val="8"/>
    <int name="volume" val="22"/>
</obj>
```

Figure 2.4: Example oBIX contract and an implementation [10]

Figure 2.4 shows an example of how contracts are used. A contract with the URI "`/def/television`" is defined. It specifies that televisions have a boolean object called "`power`" and an integer value between 2 and 200 (with a default value of 2) named "`channel`". Next it shows an object with the URI "`/livingRoom/tv`" being defined. Using the `is`-facet it extends the previously defined contract. It overwrites its channel value with 8 and defines an additional integer value called "`volume`". The "`power`" object isn't explicitly defined in the `tv` object, but it is inherited from the contract. The default value for "`power`" of the object `tv` is set to `false`.

Contracts can be used to define the data points offered by different types of devices, such as

temperature sensors or fan speed actuators [8]. One such contract is shown in Figure 2.5. It is the contract for a push button, which has only one binary data point. Since the state of a button is not writable, the `writable` facet can be omitted since it defaults to `false`.

More complex devices may have multiple data points. A dimming push button, for example, could have an additional integer value representing the current amount of dimming.

```
<obj href="iot:PushButton" is="iot:Sensor">
  <bool name="value" val="false"/>
</obj>
```

Figure 2.5: PushButton contract [8]

oBIX Services

These operations allow the implementation of services in oBIX. The standard defines several services like the Watch-service, History-service and Alarm-service.

The Watch-service is used to create watches that enable a client to observe changes to specified objects. The client can poll for changes in objects registered with the watch. Several objects can be added to a watch at once, so that they can be polled together. Only objects that have been modified since the last poll request are returned when polling.

Many automation systems provide a mechanism to create historical archives of point data. oBIX provides a standardized way to model and query such time sampled data using the History-service. A rollup of summarized data can be generated, containing the minimum, maximum and average values of numeric data points, for use in higher level applications.

The alarming feature provides a normalized way to query, watch, and acknowledge alarms. An alarm indicates a condition which requires notification of a user or another application. For that reason, alarms require acknowledgment to indicate that someone has taken action to resolve the alarm condition.

RESTful

oBIX provides a RESTful transfer mechanism for its XML documents. REST stands for REpresentational State Transfer and is an architectural style for Web services. There are three generic request types for oBIX requests: read, write, invoke. They allow to read object values, write object values, and invoke operations.

A mapping of HTTP requests to oBIX requests is defined in the oBIX standard [10]. This mapping can be seen in Table 2.1. Another protocol binding is defined for SOAP (Simple Object Access Protocol).

HTTP request method	oBIX request
GET	read
PUT	write
POST	invoke

Table 2.1: Mapping of HTTP requests to oBIX requests

Object discovery

In order to facilitate object discovery, oBIX servers implement a "lobby". This is the central entry point and lists the URIs for objects defined by the oBIX standard, such as an "about" object and the Watch-service object. The lobby is where vendor specific data and service discovery objects should be placed. Conventionally the URI for the lobby is `/obix`.

2.4 BACnet

BACnet (Building Automation and Control Network) is a data communication protocol developed by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) [1]. It is an American national standard, a European standard and an ISO global standard.

Development of the standard began in 1987, and is under continuous maintenance by the ASHRAE Standing Standard Project Committee 135 (SSPC 135). It was motivated by the desire for interoperability between building automation devices from different vendors. The standard defines how automation and control systems can interoperate with other BACnet systems. Multiple BACnet systems on the same network are able to communicate and to request functions from another.

BACnet networking architecture

BACnet can communicate over local area networks such as Ethernet, ARCnet, MS/TP, PTP and LonTalk. The data on the BAS can also be routed through some IP (Internet Protocol) routers, so that remote monitoring and control can then become possible (via a BACnet/IP device).

The BACnet standard adheres to a collapsed version of the ISO OSI model of a layered communication architecture, seen in Figure 2.6. Therefore, various network access methods and physical media may be used to exchange messages.

BACnet object model

BACnet follows an object oriented approach in modeling its devices and data points. A BACnet device can be seen as a collection of objects. Each device must have the special "device"-object that provides additional information about itself, including its device identifier, its name, and a list of all objects available on the device.

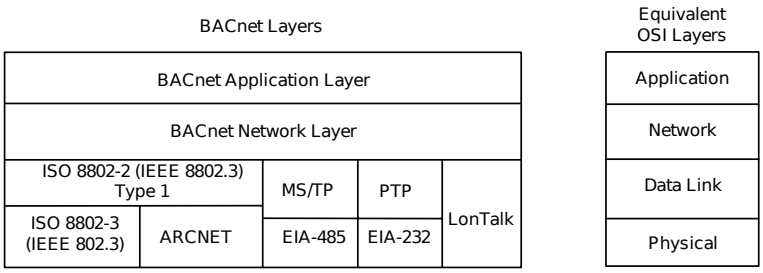


Figure 2.6: BACnet networking architecture [1]

The BACnet standard defines a number of object types [1]. These objects represent hard data points or describe soft data points such as setpoints. The "Program" object type, for example, represents a process running within a BACnet device.

Hard data points can be modeled using generic object types for binary and analog values. There are also separate object types for input, output and value objects, resulting in a total of 6 object types: `AnalogInput`, `AnalogOutput`, `AnalogValue`, `BinaryInput`, `BinaryOutput`, `BinaryValue`.

Input objects receive their value from an external source (like sensors) and therefore are not writable. Output objects are writable and represent control outputs. Value and output objects can be used as setpoints (for example, the target temperature in a heating system).

Each object has a collection of properties. Among these properties are name and type of the object, as well as a description. For analog objects, a property provides a way to specify the units of the value. There are a number of other properties, and each object type has its own set of properties that are specific to that type.

Object Identifier

Objects within a device are identified through their object identifier. This identifier consists of two parts: The object type and an instance number. In order to uniquely identify a property of an object inside a BACnet network four values are required: the device identifier of the device the object is residing on, the object identifier (object type and instance number) and the property identifier.

Present_Value

Arguably the most important property is the "`Present_Value`". As the name suggests, the property stores the current value of an object. It might represent the temperature in degrees Celsius in the case of a temperature sensor, for example.

The `Present_Value` is not necessarily writable on all objects. If this property is writable, then the properties "`Priority_Array`" and "`Relinquish_Default`" are also present on the object. These properties are part of the prioritization mechanism.

The priority array is an array of 16 values which correspond to 16 available levels of priority. The elements are in order of decreasing priority, so the first element (priority 1) has the highest priority. An entry in the priority array can be either a value or `NULL`. The value with highest priority that is not `NULL` is used as the "`Present_Value`" property. If all entries in the priority array are NULL, then the value of the "`Relinquish_Default`" property is used as a fallback.

The priority array property cannot be written to directly. To set an entry in the priority array, a `WriteProperty` request including the priority to override is issued on the `Present_Value` property instead. To clear entries, the value of the `WriteProperty` request shall be `NULL`. If no priority is specified on the request, a default priority of 16 (the lowest priority) is assumed.

BACnet Interoperability Building Blocks

BACnet assures interoperability between devices from different vendors if they agree to implement a set of BIBBs (BACnet Interoperability Building Blocks). BIBBs are collections of BACnet services, that are prescribed in terms of an "A" (client) and "B" (server) device. Together they define what services have to be supported on two devices to enable successful interaction between them [1]. A device's PICS (Protocol Implementation Conformance Statement) identifies the BIBBs that are implemented.

CHAPTER 3

Generic oBIX mapping for BACnet devices

A way to represent a BACnet network and its devices and objects inside an oBIX server is desired in order to be able to access and manipulate them through an oBIX representation. To achieve this, BACnet objects can be mapped to oBIX objects. In this thesis, only the mapping of the BACnet object types `AnalogInput`, `AnalogOutput`, `AnalogValue`, `BinaryInput`, `BinaryOutput` and `BinaryValue` to oBIX objects is discussed, as they are sufficient to model a wide variety of devices and applications, including sensor and actuator values, as well as setpoints. Other object types can be mapped in a similar fashion if needed.

3.1 Mapping of `Present_Value`

The group of analog object types in BACnet have a present value with data type `REAL`, which can be directly mapped to the oBIX object type `real`. BACnet's binary object types have the data type `BACnetBinaryPV`, which can take the values `active` and `inactive`. These values can be mapped to `true` and `false` of the oBIX object type `bool`, respectively. Values of the BACnet priority array can also take the value `NULL`. Such null values can be expressed in oBIX by setting the value object's `null` facet to `true`.

To create a functional mapping, that allows reading and writing values from and to devices, the present value is the only property required to be mapped, other properties like the description are not essential. Therefore, we can already define basic oBIX contracts corresponding to the BACnet objects. These contracts are shown in Figure 3.1. BACnet's input objects are by default not writable; output objects are writable by default. This is reflected in the contract definition.

```
<obj href="iot:AnalogInput">
  <real name="value" val="0" writable="false"/>
</obj>
<obj href="iot:AnalogOutput">
  <real name="value" val="0" writable="true"/>
</obj>
<obj href="iot:AnalogValue">
  <real name="value" val="0"/>
</obj>
<obj href="iot:BinaryInput">
  <bool name="value" val="false" writable="false"/>
</obj>
<obj href="iot:BinaryOutput">
  <bool name="value" val="false" writable="true"/>
</obj>
<obj href="iot:BinaryValue">
  <bool name="value" val="false"/>
</obj>
```

Figure 3.1: Basic oBIX contracts corresponding to BACnet object types

3.2 Mapping the object identifier to a URI

To uniquely identify objects, BACnet uses the object identifier, while oBIX uses URIs. A structured approach for constructing the URI of the mapped object is proposed. This structure organizes the oBIX objects in a hierarchy, making it easy to use and browsable.

An oBIX server may contain devices from multiple BACnet networks, however object identifiers are only unique within the network. To distinguish devices from different networks, an object representing the network is created. This object is the root of the object tree we are creating and might be published through the oBIX lobby directly. Its `name` and `href` facets should correspond to the name of the network. Figure 3.2 shows an example network object.

```
<obj name="BACnet Lab" href="/BACnetLab/">
  <ref name="2098177" href="/BACnetLab/2098177"/>
</obj>
```

Figure 3.2: Example network object

The next layer in the object hierarchy contains the devices within the BACnet network. BACnet devices are identified by their device instance number, therefore each device is represented by an oBIX object whose `href` corresponds to the instance number. Figure 3.3 shows an example device object.

Finally, the layer below the device objects contains the actual BACnet objects that are mapped to functional oBIX objects. Their `href` is a concatenation of their object type and instance number, such as “AnalogInput0”. Figure 3.3 shows some example `href`s.

```
<obj href="/BACnet/2098177/">
  <ref href="/BACnet/2098177/BinaryOutput0"/>
  <ref href="/BACnet/2098177/BinaryInput0"/>
  <ref href="/BACnet/2098177/BinaryValue0"/>
  <ref href="/BACnet/2098177/AnalogInput0"/>
  <ref href="/BACnet/2098177/AnalogOutput0"/>
  <ref href="/BACnet/2098177/AnalogOutput1"/>
</obj>
```

Figure 3.3: Example device object

The oBIX objects from each layer only contain references to the objects of the layer below, using the `ref` object type and specifying the `href` facet to point to the objects. This has multiple advantages. For one, less information has to be transmitted when a higher level object is read. Only the layer requested needs to be transmitted, clients can then follow the references to request objects of the next layer. It is also easier to handle for a human operator, as the structure allows easy browsing and does not showing unnecessarily detailed information. Finally, if the complete object tree would be requested, each BACnet object represented in this tree would have to be read. In large networks, this would lead to a lot of unnecessary BACnet traffic.

3.3 Mapping read and write requests

If the object identifier of a BACnet object is known, then the oBIX contract it maps to can be derived from the object type. In this section, the mapping of read and write requests from oBIX to BACnet is discussed. A summary can be seen in Table 3.1.

oBIX request (HTTP binding)	**BACnet request**
`GET /bacnet/123123/AnalogOutput1 HTTP/1.1`	`ReadProperty` Request - Object Identifier: AnalogOutput, 1 - Property Identifier: Present_Value

`PUT /bacnet/123123/AnalogOutput1 HTTP/1.1`
 	
`<real val="3.1415"/>`
	`WriteProperty` Request
	- Object Identifier: AnalogOutput, 1
	- Property Identifier: Present_Value
	- Property Value: 3.1415
	- Priority: 10

`PUT /bacnet/123123/AnalogOutput1 HTTP/1.1`
 	
`<real null="true"/>`
	`WriteProperty` Request
	- Object Identifier: AnalogOutput, 1
	- Property Identifier: Present_Value
	- Property Value: `NULL`
	- Priority: 10

Table 3.1: Mapping oBIX to BACnet requests

Read requests

When the constructed oBIX object is read, the value has to be retrieved from the BACnet object it represents. To this end, oBIX read requests are mapped to the BACnet `ReadProperty` service. The `ReadProperty` service takes three arguments: the object identifier of the object to read from, the property identifier of the property to be read, and optionally a property array index.

The object identifier is already known. The property identifier is set to the identifier of the `Present_Value` property. The property array index is only needed if the property being read is an array. Since `Present_Value` is only a single value this argument is omitted. The response of the service contains the read property value. Its data type is either `REAL` or `BACnetBinaryPV`, depending on whether the object type is one of the analog or binary object types. The value can then be interpreted and mapped to an oBIX value as previously discussed.

Write requests

For write requests to the oBIX object, a similar mapping has to be defined. The write request maps to the `WriteProperty` service. This service takes five arguments: the object identifier, a property identifier, a property array index, a property value and a priority.

For the object identifier, property identifier and property array index the same considerations are made as for the `ReadProperty` service before. The property value argument contains the value we want to write to the object. For the priority, an integer in the range 1-16 can be chosen. If the priority is omitted, a default priority of 16 (lowest priority) is implied. The highest priorities should be reserved for life safety emergency overrides, so a mid-range priority of around 10 is suggested.

The priority mechanism allows to override a value with lower priority, and then later revoke the new, high priority value and to return to the original value. To revoke a value with certain priority, its entry in the priority array has to be reset to `NULL`. oBIX does not have a null object, instead it uses a facet to indicate a null value. Therefore, writing to an oBIX object with a null value can be used to reset its value. In this case, the property value to use in the `WriteProperty` service request is BACnet's `NULL` data type.

Figure 3.4 shows examples of how the priority array and the `Relinquish_Default` property affect the `Present_Value`. If all entries in the priority array are NULL, then the value of the "`Relinquish_Default`" property is used as a fallback, otherwise the highest-priority value of the priority array - the one with the lowest index - is used.

Priority Array

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	RD	PV
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	**12**	12
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	**42**	NULL	NULL	NULL	NULL	NULL	NULL	12	42
NULL	NULL	NULL	NULL	**22**	NULL	NULL	NULL	NULL	42	NULL	NULL	NULL	NULL	NULL	NULL	12	22

Figure 3.4: Examples for the priority array and how it affects the `Present_Value`

3.4 Check if BACnet object is writable

The contract definitions already set the default of the `writable` facet for each object type. However, the present values of input and value objects can become writable depending on the `Out_Of_Service` and `Relinquish_Default` properties [1]. In order to correctly reflect this in the oBIX object, these properties have to be checked.

Output objects are always writable. Input objects are writable, if the property `Out_Of_Service` is set to `TRUE` or if the `Present_Value` property is commandable, otherwise they are read-only.

The boolean property `Out_Of_Service` indicates whether the physical input to the object is currently in service. If `Out_Of_Service` is `active`, the physical input is decoupled from the `Present_Value` property and manually changing the `Present_Value` is allowed.

If the `Present_Value` is commandable, then the properties `Priority_Array` and `Relinquish_Default` are both present in the object, too. To check if these properties are available, a `ReadProperty` service request can be attempted. If they are not available, the request will result in an `UNKNOWN_PROPERTY` error, indicating that the `Present_Value` property is not commandable.

These conditions have to repeatedly be checked every time the oBIX object is read, as they can change over time.

Using the techniques described so far, a BACnet object can be mapped to a functional oBIX object that can be read and written to. An example of an object using this mapping is shown in Figure 3.5.

```
<obj href="/BACnet/10003/AnalogOutput1" is="iot:AnalogOutput">
  <real name="value" href="value" val="4200" writable="true"/>
</obj>
```

Figure 3.5: Example oBIX object representing a BACnet Analog Output object

3.5 Additional properties

At this point, a functional mapping, that allows reading and writing BACnet objects through oBIX, has already been defined. However, the mapping can be further improved by making use of additional properties that BACnet objects provide. For instance, there is no information about the role of the object, as only its value is mapped. But in order to effectively use a value, it is necessary to know what it represents. In the following, a few useful properties are discussed.

Name

Every BACnet object has an `Object_Name` property. A name is unique within the BACnet device that consists of the object. In oBIX, the children of an object must have unique names, too. Names in BACnet are restricted to printable characters only, and a minimum length of one character. oBIX imposes stricter constraints on names and only allows ASCII letters, digits,

underbars, and dollar signs. Additionally, a digit must not be used as first character [10]. Invalid characters have to be stripped from the `Object_Name` before it can be used as name for the oBIX object. The problem of duplicate names from different BACnet objects is largely avoided by structuring the oBIX representation as discussed in the next section.

Description

BACnet also provides a `Description` property. This property can be useful to understand the purpose of the object and can be included in the constructed oBIX object. To this end, a new child object of type `str` by the name of "description" is added to the object. Its value is the value obtained from the `Description` property of the BACnet object by a `ReadProperty` service request. No sanitation of the obtained string is required, as it is restricted to printable characters only.

Units

For analog values, a unit is required to be able to interpret it correctly. Both, BACnet and oBIX provide means to specify the units for a value. In BACnet, the `Units` property of the data type `BACnetEngineeringUnits` represents the `Present_Value` property's units of measurement.

`BACnetEngineeringUnits` is an enumeration of many units from different domains, such as acceleration, area, currency, force, humidity, length, velocity, volumetric flow and others. Units with an enumerated value in the range 0-255 are reserved for definition by ASHRAE, values in the range 256-65535 may be used freely.

oBIX features a flexible way to define units numerically. Dimensions are specified using the `obix:Dimension` contract using the seven fundamental SI units and their exponent (Figure 3.6).

```
<obj href="obix:Dimension">
  <int name="kg" val="0"/>
  <int name="m" val="0"/>
  <int name="sec" val="0"/>
  <int name="K" val="0"/>
  <int name="A" val="0"/>
  <int name="mol" val="0"/>
  <int name="cd" val="0"/>
</obj>
```

Figure 3.6: oBIX's Dimension contract [10]

An actual unit is represented with the `obix:Unit` contract (Figure 3.7). It contains a dimension that can be scaled and offset ($unit = dimension \cdot scale + offset$) and the unit's symbol [10]. Figure 3.8 shows how a kilowatt can be expressed using this system.

Some units don't fit into this model, like logarithmic units or units dealing with angles. Such units should use a dimension where every exponent is set to zero.

```
<obj href="obix:Unit">
  <str name="symbol"/>
  <obj name="dimension" is="obix:Dimension"/>
  <real name="scale" val="1"/>
  <real name="offset" val="0"/>
</obj>
```

Figure 3.7: oBIX's Unit contract [10]

```
<obj href="obix:units/kilowatt" display="kilowatt">
  <str name="symbol" val="kW"/>
  <obj name="dimension">
    <int name="m" val="2"/>
    <int name="kg" val="1"/>
    <int name="sec" val="-3"/>
  </obj>
  <real name="scale" val="1000"/>
</obj>
```

Figure 3.8: Kilowatt as obix:Unit [10]

oBIX provides a database of predefined units. If possible, `BACnetEngineeringUnits` should be mapped to the corresponding unit in this database, otherwise new units can be defined as shown. For example, BACnet's `revolutions-per-minute` (with an enumerated value of 104) maps to `obix:units/revolutions_per_minute`.

BACnet has a special enumerated value representing the absence of a unit, "no-units". This value can be mapped by simply omitting the `unit` facet of the oBIX object.

3.6 Completed oBIX object

By mapping these additional properties, a much more meaningful object is obtained. Compare Figure 3.5 with Figure 3.9. The second object gives a much better idea of what it does, even though both may actually represent the same BACnet object. A summary of BACnet properties and how they were used in the mapping is given in Table 3.2.

```
<obj name="lfan" href="/BACnet/10003/AnalogOutput1"
    is="iot:AnalogOutput">
<real name="value" href="value" val="4200"
    unit="obix:units/revolutions_per_minute" writable="true"/>
<str name="description" href="description"
    val="left fan speed setpoint"/>
</obj>
```

Figure 3.9: Example oBIX object representing a BACnet Analog Output object

BACnet property	**Use in oBIX object mapping**
`Present_Value`	value object, as `real` or `bool`
`Object_Name`	`name` facet
`Description`	`string` description object
`Units`	`unit` facet on value object

`Out_Of_Service`
`Priority_Array`
`Relinquish_Default`
	Used to check if `Present_Value` is writable

`Object_Identifier`	`href` facet

Table 3.2: BACnet properties and their usage in the mapping to an oBIX object

CHAPTER 4

Auto-discovery of BACnet devices

4.1 BACnet's Remote Device Management Services

Among the services that BACnet provides are a group of services collectively known as "Remote Device Management Services". These services provide a number of functions, including operator control and auto-configuration.

In particular, there are two services that can be used to discover devices, the `Who-Is` and `Who-Has` services. They eliminate the effort of having to program other devices' network addresses into each device. The service messages are broadcast in the BACnet network to every device, and the receiving devices may respond with an acknowledgment message containing their address.

Who-Is and I-Am

The `Who-Is` service is used to determine the device identifier, the network address, or both of other BACnet devices that are on the same network as the device issuing the service request. It is an unconfirmed service, meaning that it does not require a response.

There are multiple ways of using the service. It can be used to determine the device identifier and network addresses of all devices on the network. If a device identifier is already known, then the service can be used to determine the address of the corresponding device.

The `I-Am` service informs other devices about its sender by broadcasting an unconfirmed request containing its device identifier. The service may be used at any time. Usually an `I-Am` request is sent after a device has initialized to inform other devices about its availability.

When a device receives a `Who-Is` request, it may respond by sending an `I-Am` service request.

Who-Has and I-Have

Another way to locate devices on the network is provided by the unconfirmed services `Who-Has` and `I-Have`.

The `Who-Has` service is similar to the `Who-Is` service. It can be used to ask for the device identifier of devices that contain an object that has a given object name or object identifier.

In response, devices that have the requested resource send an `I-Have` service request, containing the device identifier, as well as both the object name and object type of the requested object.

Device Instance Ranges

Optionally, a range of device instances can be specified as an argument for the `Who-Is` and `Who-Has` service requests. If the range is omitted, then every device that receives the request will process it. Otherwise, only devices whose device object's instance number is in between the specified range will answer.

In large networks, an unbounded `Who-Is` request to all devices at once would result in a lot of answers at the same time and sudden network load. As a consequence, packets containing the `I-Am` response are more likely to be dropped on their way to the requesting device. Since the `I-Am` service is unconfirmed, packets will not be resent. This may lead to not all devices being discovered.

Device instance ranges provide a solution to this problem. Instead of one `Who-Is` request to all devices, the request can be split up into several smaller requests, thereby reducing the network load per request.

Device objects and object lists

Every BACnet device is required to have a `device` object. Its instance number is the same as the device's identifier.

The `device` object offers a property called `Object_List`. It is an array type property that contains the object identifiers of all objects available on the device. This property allows to query all the objects contained in a device that has been discovered by the `Who-Is` service.

4.2 BACnet/IP broadcast management devices

BACnet networks consisting of multiple subnets connected through IP routers build an internetwork. Because IP Routers do not forward any broadcasts, the Remote Device Management services, which rely on broadcast messages, are not able to propagate past them and are confined to the subnet the message originated in.

To enable the Remote Device Management services to work on the complete network, a BACnet/IP broadcast management device (BBMD) can be used. Figure 4.1 shows how BBMDs are used in an internetwork.

A BBMD is placed on every subnet. They listen for broadcasts originating from their local subnet and forward these messages directly to the BBMDs on the other subnets. These BBMDs

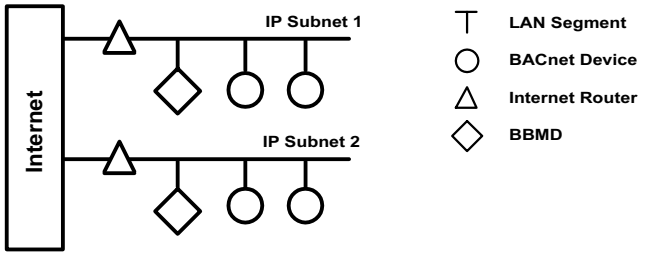


Figure 4.1: BBMDs forward broadcasts between subnet [1]

then broadcast the message on their own subnet. Because the BBMDs forward the messages using point-to-point connections (instead of broadcasts) they pass through IP Routers without a problem.

BBMDs can't use broadcasts to discover the BBMDs on other subnets, therefore each BBMD contains a Broadcast Distribution Table (BDT) that is manually populated. The entries in the BDT specify where broadcasts should be forwarded to. Devices on a subnet without a local BBMD that wish to receive BACnet broadcast messages can register with a remote BBMD to be included in their BDT as a foreign device.

BACnet Routers

BACnet/IP devices may operate together with BACnet devices on non-BACnet/IP networks. To enable communication between devices on differing data links, a BACnet Router is placed between them (see Figure 4.2).

Routed messages travel from router to router as directed messages until the routed message reaches its final destination network. If the originating message was a broadcast, the router will substitute the appropriate broadcast command for the attached data link.

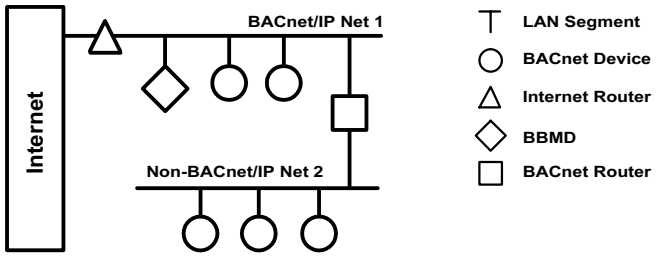


Figure 4.2: The addition of a non-BACnet/IP data link results in two networks [1]

4.3 Automatic configuration for oBIX servers

The Remote Device Management services offered by BACnet can be used to automatically configure devices for use in oBIX servers.

Discovery of objects

Using the `Who-Is` service, BACnet devices can be discovered and then mapped to oBIX objects. The following steps describe the general approach.

1. A virtual local BACnet device is set up, in order to be able to interact with the BACnet network. The virtual device is able to send and receive BACnet messages.
2. A listener for `I-Am` and `I-Have` requests is set up.
3. A `Who-Is` service request is broadcast. If the network is large, the request can be split up into several requests using the device instance range arguments to target devices in chunks, as described above.
4. The `I-Am` listener receives messages from devices on the network in response to the `Who-Is` request. The device's identifier is part of the response. To get a list of objects contained in the device, a `ReadProperty` service request for the `Object_List` property is sent to each device.
5. The BACnet objects contained in the object list are mapped to oBIX objects based on their object type as described in the previous section.
6. Finally, the constructed oBIX objects can be published through an oBIX server.

The last two steps can be skipped if the object has already been mapped before, as it would be if multiple `I-Am` requests are received from the same device.

Following these steps, an object hierarchy like described in the previous section can be created. When a BACnet device comes online, it usually sends an `I-Am` broadcast on its own to signal its availability. Those requests are received by the virtual local device and can also be mapped and published to an oBIX server at runtime.

CHAPTER 5

Case Study

5.1 The IoTSyS Gateway

The techniques for automatic discovery and mapping of BACnet devices to oBIX objects presented in this thesis have been implemented in the open-source project "IoTSyS" [8] [6].

IoTSyS is a transparent IPv6 multi-protocol gateway that allows sensors and actuators from various home and building automation systems to participate in the Internet of Things. The gateway can be equipped with multiple protocol stacks and physical interfaces to different media to communicate with various BAS. It is a Java application based on Web services and oBIX to provide an interoperable interface for smart objects.

oBIX protocol binding to CoAP

IoTSyS uses the OASIS standard oBIX as application layer protocol. In addition to the HTTP and SOAP bindings oBIX provides, a CoAP binding is defined.

oBIX provides the concept of Watches for observing resources, but it is tailored to a polling interaction model. The oBIX watch services can still be used with CoAP, but to take advantage of the asynchronous communication of CoAP, the CoAP observe extension [5] is implemented and included in the oBIX protocol binding to CoAP. This allows a CoAP GET request with the `observe` option set to be sent and to receive multiple responses without the need for constantly repeated polling.

Efficient encoding using EXI

For constrained devices, using XML based messages is heavyweight. With IEEE 802.15.4 as data link layer, for example, the application payload for a single message is limited to less than 72 Bytes in order to be transmitted without fragmentation [8]. The oBIX standard defines an efficient custom binary encoding as a solution to this problem. IoTSyS implements an additional, more standardized solution - the W3C recommendation EXI (Efficient XML Interchange) [8].

EXI is a binary encoding for XML documents. A schema can be provided for an informed encoding, leading to an even more efficient result.

For IoTSyS, promising results can be achieved using EXI even without having a schema [8]. IoTSyS specifies IoT contracts for various device types as oBIX contracts. This allows the definition of XML schema documents leading to a fixed EXI grammar and optimal binary representation of exchanged messages.

5.2 Architecture of IoTSyS

Figure 5.1 shows an overview of the components of the IoTSyS gateway. A key part of the architecture are the various protocol adapters. They provide the interface to the BAS. This connection might require different physical and data link layers, depending on the BAS.

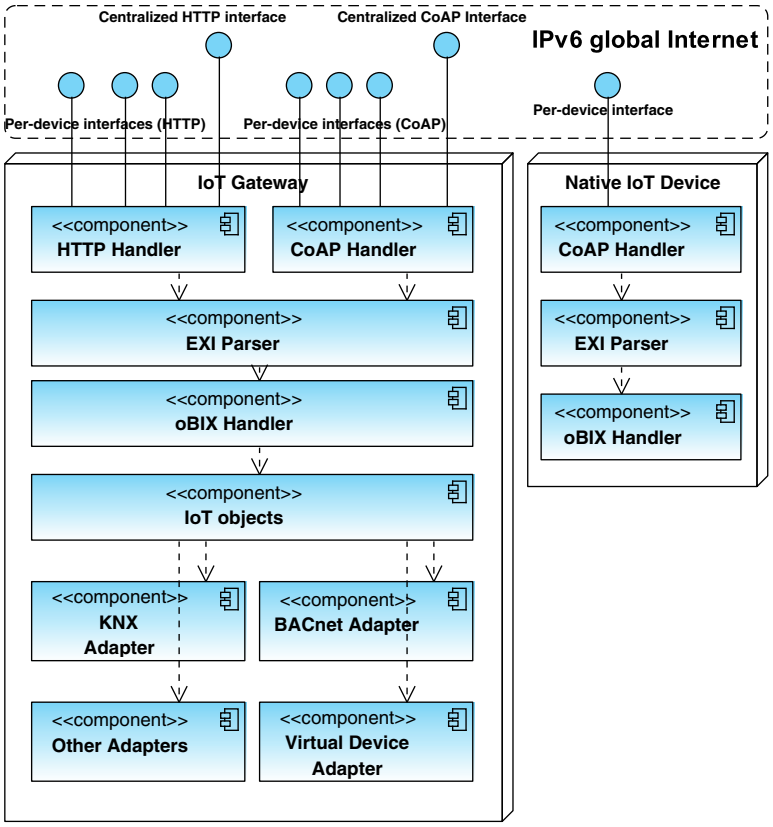


Figure 5.1: IoTSyS multi-protocol gateway architecture [8]

oBIX Handler

The oBIX Handler implements the handling of read, write and invoke requests on oBIX objects. oBIX services Alarming and the watch service are also implemented in this component. In order to allow for direct device access over HTTP and CoAP, the oBIX Handler publishes each device using separate handlers on per-device IPv6 addresses. This makes it possible to use the centralized approach of accessing the device through the gateway and the direct device access approach in parallel.

The oBIX Handler only works on XML data, the EXI compression and decompression happens transparently in the EXI Parser layer between the oBIX Handler and the HTTP/CoAP Handlers.

Technology connectors

To communicate with a variety of BAS, IoTSyS employs "technology connectors". Technology connectors are components that publish devices of a BAS as oBIX objects through the gateway's object broker. They implement the read and write operations of the oBIX objects, transparently translating them to commands appropriate for the BAS. Some technology connectors are already included in the IoTSyS project, including connectors for KNX, W-MBus and BACnet.

Device Loader

When IoTSyS starts, the main device loader reads a configuration file, `devices.xml`, in which specialized device loaders for technology connectors can be defined (see Figure 5.2).

```
<devices>
  <deviceloaders>
    <device-loader>at.ac.tuwien.auto.iotsys.gateway.connectors.
      bacnet.BacnetDeviceLoaderImpl</device-loader>
  </deviceloaders>
  [...]
</devices>
```

Figure 5.2: Device loader specified in the devices configuration file

The main device loader continues instantiating the specified device loaders and calls their initialization methods. Each technology connector's device loader sets up its devices and publishes them as oBIX objects through the gateway's object broker. To do so, it looks for configuration inside the `devices.xml` file.

For the BACnet technology connector, the BACnet network to access has to be configured. The `devices.xml` file contains an entry for each network in form of a `connector` entity.

```
<devices>
  [...]
  <bacnet>
    <connector>
      <name>BACnet Lab</name>
      <enabled>true</enabled>
      <localDeviceID>4235</localDeviceID>
      <localPort>47808</localPort>
      <broadcastAddress>192.168.161.255</broadcastAddress>
      [...]
    </connector>
  </bacnet>
  [...]
</devices>
```

Figure 5.3: BACnet connector definition

IoTSyS's BACnet technology connector uses the BACnet4J library by Serotonin Software [13], which creates a virtual BACnet device through which it interacts with the network, therefore a local device identifier is required. An example definition of a BACnet connector is seen in Figure 5.3.

So far, only network access information has been specified. BACnet devices can be explicitly configured by defining `device` entities in the connector. The devices need to be configured using the address of an object and the oBIX object type that it maps to, specified as an oBIX object's class. This class will be instantiated and set up according to the configuration. The URI under which the device should be published also needs to be specified using an `href` entity. For an example device configuration, see Figure 5.4.

5.3 Implementation in IoTSyS

To evaluate the techniques described in this thesis, the BACnet technology connector for IoTSyS has been extended to support automatic discovery and configuration of BACnet devices. This section contains implementation details.

BACnet device loader

The configuration for BACnet connectors in the `devices.xml` file has been extended by an additional setting, the boolean `discovery-enabled`. The BACnet device loader reads the configuration file, and instantiates a `BACnetConnector` for each connector defined in `devices.xml`'s `bacnet` section. If `discovery-enabled` is set to `true`, the BACnet device loader sets up a listener for BACnet events for the virtual local BACnet device corresponding to the connector. The listener will handle incoming `I-Am` and `I-Have` requests. After the listener is set up, a `Who-Is` request is broadcast in the network.

```
<devices>
  [...]
  <bacnet>
    <connector>
      [...]
      <device>
        <type>at.ac.tuwien.auto.iotsys.gateway.obix.objects.
          iot.sensors.impl.bacnet.TemperatureSensorImplBacnet</type>
        <address>2098178, 0, 1, 85</address>
        <href>temperature2</href>
        <ipv6>2001:629:2500:570::10c</ipv6>
        <historyEnabled>true</historyEnabled>
      </device>
      [...]
    </connector>
  </bacnet>
  [...]
</devices>
```

Figure 5.4: BACnet device definition

Listener for `I-Am` and `I-Have` requests

When the listener receives `I-Am` and `I-Have` requests, the information about the sending device is encapsulated in an instance of `RemoteDevice`. By reading the device object's `Object_List` property, a list of `ObjectIdentifier` instances is retrieved.

Each object is given to the `BACnetDeviceFactory`, which tries to map it to an oBIX object instance. If successful, the `href` of the object is set as discussed before, where the top-level path element corresponds to the connector's name. The object is then published through the oBIX object broker.

BACnet Device Factory

The `BACnetDeviceFactory` class has a static method `createDevice` that instantiates an oBIX object implementation based on a BACnet object's type. It supports the generic object types discussed in section 3.

Generic object implementations

The generic object types have been implemented in a class hierarchy as seen in Figure 5.5.

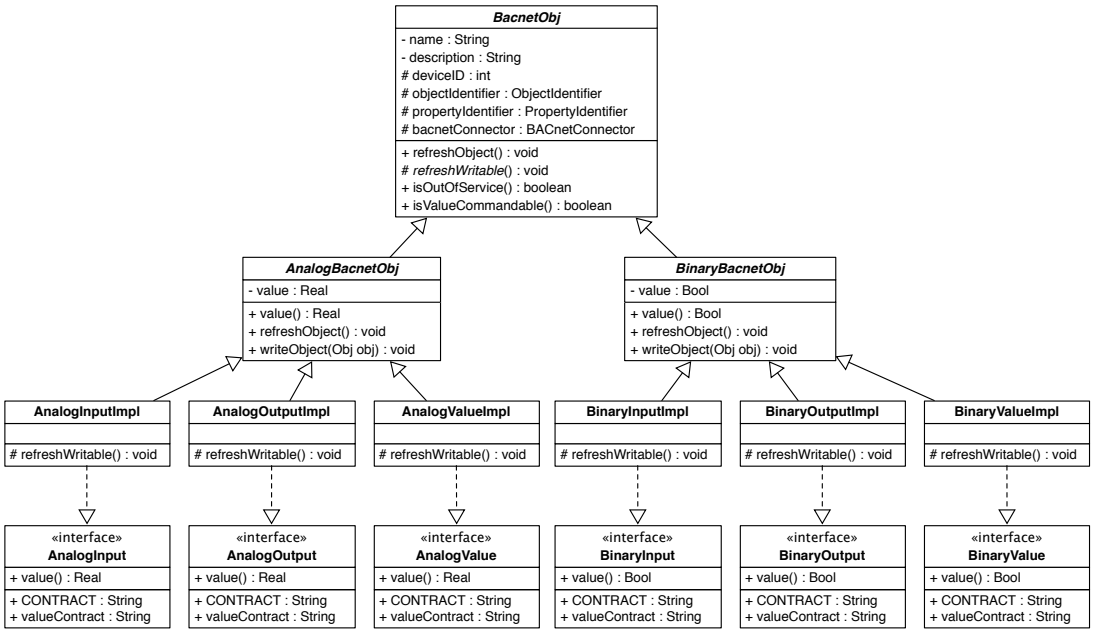


Figure 5.5: Class hierarchy of the generic object type implementations

Abstract class `BACnetObj`

The root of the class hierarchy is given by the `BACnetObj` class. It is an abstract class that extends the `Obj` class from the oBIX toolkit. The `name` and `description` child objects, shared by all classes in the hierarchy, are defined here.

When the object is read through the oBIX server, the method `refreshObject` is called first. The first time this method is called, the BACnet properties for name and description are read and stored. On subsequent reads, the stored values are used, as these properties are assumed not to change over the object's lifetime.

The class also provides implementations for methods called `isOutOfService` and `isValueCommandable`, which can be used in descendant classes to check if the value object is writable. To this end, `refreshObject` also calls the method `refreshWritable`, which descendants can overwrite.

Analog object types

The abstract class `AnalogBacnetObj` is the shared parent class for the concrete analog object type implementations. As such, it defines the child object `value` of type `real`.

The `refreshObject` method is extended by the ability to read the underlying BACnet object's value. Additionally, on the first call, the BACnet object's units are determined and cached. `BACnetEngineeringUnits` is an enumerated value. Its integer representation is mapped to the corresponding oBIX unit contract name by a static HashMap stored in the class `BacnetUnits`.

oBIX write requests are handled by the `writeObject` method. It is implemented by sending `WriteProperty` requests to the BACnet object.

The concrete classes inheriting from `AnalogBacnetObj` are `AnalogInputObjectImpl`, `AnalogOutputObjectImpl` and `AnalogValueObjectImpl`. In these concrete classes, the corresponding oBIX contract is set. The contracts are named after the BACnet object type, with the prefix "iot:" (e.g. `iot:AnalogInput`).

The concrete classes also implement the `refreshWritable` method, checking for writability as discussed in section 3 using the methods `isOutOfService` and `isValueCommandable` defined in `BACnetObj`.

Binary object types

The abstract class `BinaryBacnetObj` is the shared parent class for the concrete binary object type implementations. As such, it defines the child object `value` of type `bool`. Its methods and concrete object type implementations are defined similarly to those of the analog object types. However, binary objects do not define a unit.

Interaction sequence

Figure 5.6 shows the sequence of messages that occur when the gateway broadcasts a `Who_Is` request to discover devices, and a device answering the request. BACnet devices that receive a `Who_Is` request answer with an `I_Am` request. The gateway proceeds by asking for the object list of the device.

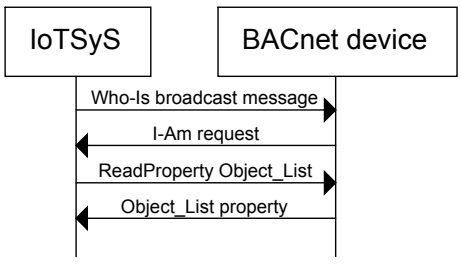
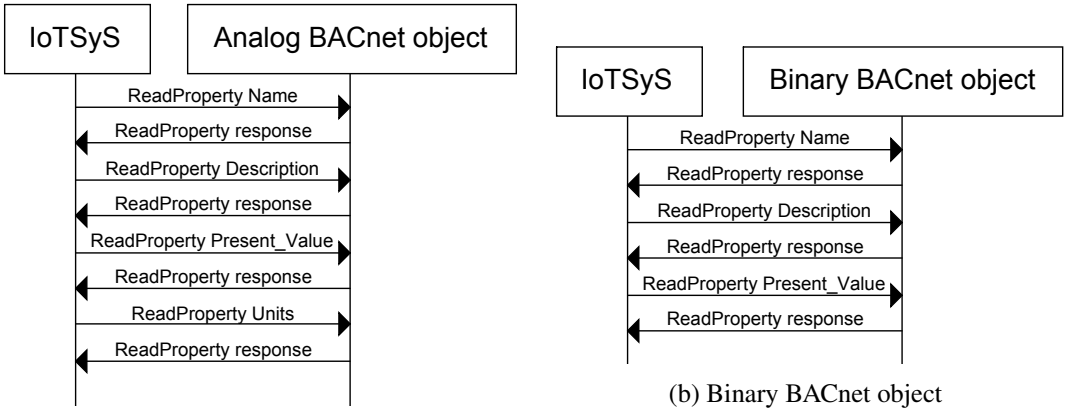


Figure 5.6: `Who_Is` broadcast is answered by BACnet device

After an object that can be mapped is found, the gateway requests some basic information about the object, such as its name and description, as seen in Figure 5.7. For analog objects, the units that the value represents are also read.

For input and value objects, the `Out_Of_Service` and `Relinquish_Default` properties may also be requested in order to determine if the object is writable, as described in section 3.



(a) Analog BACnet object

(b) Binary BACnet object

Figure 5.7: Initial message exchange

When the gateway receives an HTTP GET request for a mapped BACnet object, it refreshes its value by sending a `Read_Property` request for the `Present_Value` property to the BACnet object (Figure 5.8). The `Out_Of_Service` and `Relinquish_Default` properties might again be requested to update the writability of the object for input and value objects. Afterwards, the gateway answers the HTTP request with the updated oBIX representation of the BACnet object.

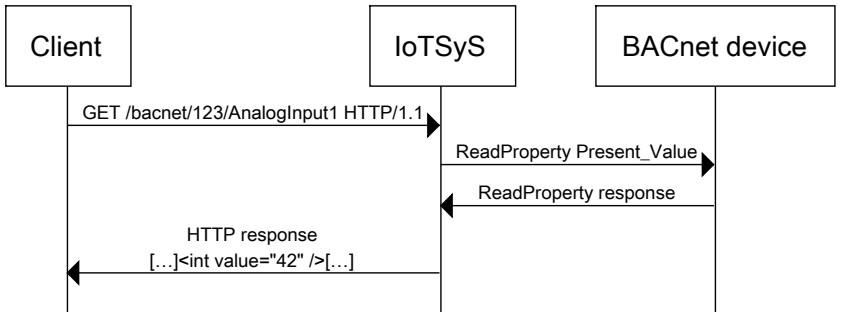


Figure 5.8: Sequence diagram showing message exchange on GET request

HTTP PUT requests are used to write to objects. Input and value objects are not always writable, and therefore the `Out_Of_Service` and `Relinquish_Default` properties are requested to check for writability of the object. If it is writable, a `Write_Property` request is sent by the gateway to the BACnet object to update its `Present_Value`. After the `Write_Property` request is confirmed, the gateway answers the HTTP request with the new state of the oBIX object.

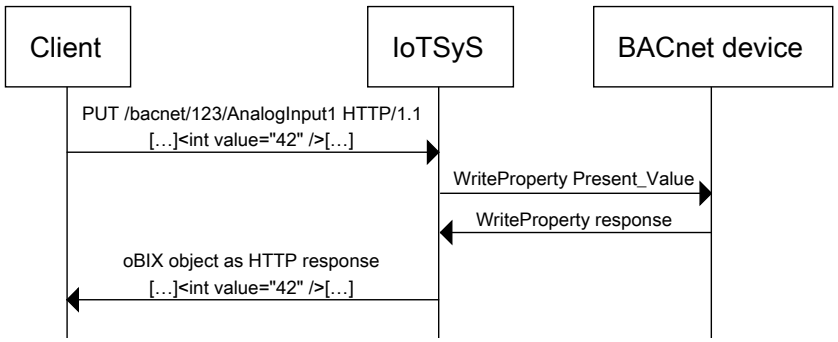


Figure 5.9: Sequence diagram showing message exchange on PUT request

5.4 Evaluation

At the Vienna University of Technology, a test-bed with several BACnet devices is set up. To interact with the devices through IoTSyS, they previously had to be manually configured in the `devices.xml` file. The auto-discovery and auto-configuration capabilities implemented in IoTSyS have been tested on this setup.

The configuration for the BACnet connector using auto-discovery used in this case study can be seen in Figure 5.10.

Using this configuration, the BACnet devices on the test-bed have been successfully discovered and published through the oBIX lobby. In Figure 5.11 the output of querying the oBIX lobby at `/obix` is shown. The lobby contains a reference to the top-level connector object.

This object contains references to devices as oBIX objects, as seen in Figure 5.12.

```
<devices>
  [...]
  <bacnet>
    [...]
    <connector>
      <name>BACnet A-Lab (Auto)</name>
      <enabled>true</enabled>
      <discovery-enabled>true</discovery-enabled>
      <localDeviceID>12345</localDeviceID>
      <broadcastAddress>128.130.56.255</broadcastAddress>
      <localPort>47808</localPort>
    </connector>
  </bacnet>
</devices>
```

Figure 5.10: BACnet connector definition with auto-discovery enabled

```
<obj href="/obix">
  <ref name="about" href="about"/>
  <ref href="/watchService" is="obix:WatchService"/>
  <ref name="enums" href="/enums" is="obix:obj"/>
  <ref name="BACnet A-Lab (Auto)"
    href="/BACnetA-Lab(Auto)" is="obix:obj"/>
  [...]
</obj>
```

Figure 5.11: oBIX lobby exposing connector object

```
<obj name="BACnet A-Lab (Auto)" href="/BACnetA-Lab(Auto)">
  <ref name="89785" href="/BACnetA-Lab(Auto)/89785"/>
  [...]
</obj>
```

Figure 5.12: Connector object listing BACnet devices

The device objects contain references to oBIX objects representing the datapoints or objects of the BACnet device, seen in Figure 5.13

These datapoints and their properties are mapped as discussed in section 3. An example of such a mapped object, representing a BACnet object from the test-bed, is seen in Figure 5.14.

```
<obj href="/BACnetA-Lab(Auto)/89785">
  <ref name="AnalogOutput0"
    href="/BACnetA-Lab(Auto)/89785/AnalogOutput0"/>
</obj>
```

Figure 5.13: Device object listing mapped BACnet objects

```
<obj name="AnalogOutput0"
     href="/BACnetA-Lab(Auto)/89785/AnalogOutput0"
     is="iot:AnalogOutput">
  <str name="name" href="name" val="TempSensor 0"/>
  <str name="description" href="description" val=""/>
  <real name="value" href="value" val="21.2" writable="true"
    unit="obix:units/degreesCelsius"/>
</obj>
```

Figure 5.14: BACnet object mapped to oBIX

The study showed that the described techniques successfully create a browsable and explorable hierarchy that represents the structure of BACnet devices on the test-bed. Reading the objects repeatedly delivers correct and up-to-date values. The oBIX objects have been used to successfully write to and control BACnet devices, such as turning lamps on and off.

The quality of the mapping depends in part on the vendors of BACnet devices and the BACnet network operators. If the device does not report units, or the name and description aren't understandable and comprehensive, or even incorrect, the quality of the mapping suffers, as the objects' function cannot be easily determined.

CHAPTER 6

Conclusion

The vision of the Internet of Things (IoT), with billions of devices connected and communicating through the Internet, is increasingly becoming a reality. Among other things, the IoT will help to realize smart and sustainable building operation. Smart buildings and a smart grid will increase building efficiency and reduce energy consumption. IPv6 will be a key technology to support the IoT. New technologies and protocols are developed to specifically meet the demands of small, low-power devices wanting to participate in the IoT, such as CoAP.

Legacy Building Automation Systems (BAS), like KNX or BACnet, are already widely deployed. It is desirable to use existing BAS infrastructure and let it be part of the IoT. Different integration approaches to achieve this goal have been explored. oBIX, a standard to represent and communicate building information through Web services, based on established standards such as XML and HTTP, promises to be a good fit for building data operating on the IoT.

This thesis proposes a technique to map BACnet devices to oBIX. Using several descriptive BACnet object properties, generic BACnet objects are mapped to oBIX objects that express the role and current state of the underlying BACnet object. These oBIX objects can be read from and written to, transparently accessing the underlying BACnet object. The topology of BACnet networks is also mapped to browsable oBIX object hierarchies. Furthermore, BACnet's Remote Device Management Services have been discussed and a way to automatically discover and map BACnet devices has been presented. This work allows seamless integration of BACnet devices into the IoT.

The presented techniques were implemented in IoTSyS, a multi-protocol gateway that makes devices from different BAS accessible via IP-based addressing and standard protocols, such as oBIX and CoAP. To evaluate the techniques described in this thesis, the implementation has been tested on a BACnet test-bed. The effort to set up BACnet networks for use in IoTSyS has been greatly reduced, requiring only minimal configuration. The resulting mapping of BACnet devices worked well, although in general, the quality of the mapping depends on the correct and complete description of BACnet devices by their vendors and the BACnet operators.

In this thesis, only the mapping of the six generic BACnet object types has been handled, which already covers a large subset of common use cases. Further work may include specialized mapping of other object types or supporting BACnet features like intrinsic reporting and alarms through the mapping.

Bibliography

[1] Standard ASHRAE. Standard 135—2004: BACnet—a data communication protocol for building automation and control networks. *American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta, Georgia, USA*, 2004.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[3] Francesco Corucci, Giuseppe Anastasi, and Francesco Marcelloni. A WSN-based testbed for energy efficiency in buildings. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 990–993. IEEE, 2011.

[4] Eric Bloom David Emmerich. Commercial building automation systems. Technical report, Pike Research, 2012.

[5] Klaus Hartke. Observing resources in CoAP. `https://tools.ietf.org/html/draft-ietf-core-observe-16`, 2014. Accessed: 2015-03-23.

[6] Markus Jung. IoTSyS open source project. `https://code.google.com/p/iotsys/`. Accessed: 2015-03-23.

[7] Markus Jung, Christian Reinisch, and Wolfgang Kastner. Integrating building automation systems and IPv6 in the Internet of Things. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 683–688. IEEE, 2012.

[8] Markus Jung, Jurgen Weidinger, Christian Reinisch, Wolfgang Kastner, Cedric Crettaz, Alex Olivieri, and Yann Bocchi. A transparent IPv6 multi-protocol gateway to integrate Building Automation Systems in the Internet of Things. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pages 225–233. IEEE, 2012.

[9] Li Li, Hu Xiaoguang, Chen Ke, and He Ketai. The applications of WiFi-based wireless sensor network in Internet of Things and smart grid. In *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*, pages 789–793. IEEE, 2011.

[10] OASIS. OBIX version 1.1 working draft 08. 2013.

[11] Luis Pérez-Lombard, José Ortiz, and Christine Pout. A review on buildings energy consumption information. *Energy and buildings*, 40(3):394–398, 2008.

[12] Zach Shelby, Klaus Hartke, and Carsten Bormann. Constrained application protocol (CoAP). `https://tools.ietf.org/html/rfc7252`, 2014. Accessed: 2015-03-23.

[13] Serotonin Software. BACnet4J. `http://bacnet4j.sourceforge.net`, 2008. Accessed: 2015-06-18.

[14] International Telecommunication Union. Measuring the information society report. 2014.

[15] Miao Yun and Bu Yuxin. Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid. In *Advances in Energy Engineering (ICAEE), 2010 International Conference on*, pages 69–72. IEEE, 2010.

[16] Sébastien Ziegler, Cedric Crettaz, Latif Ladid, Srdjan Krco, Boris Pokric, Antonio F Skarmeta, Antonio Jara, Wolfgang Kastner, and Markus Jung. IoT6–moving to an IPv6-based future IoT. In *The Future Internet*, pages 161–172. Springer, 2013.