Informatics

BACnet Multiprotocol-Gateway for Fire Alarm Systems

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Jannic Hofmann
Registration Number 11807859

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Seifried Stefan
Leitner Michael

Vienna, 1st September, 2022

Jannic Hofmann Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Jannic Hofmann

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.

Wien, 1. September 2022

Jannic Hofmann

Danksagung

Bedanken möchte ich mich an erster Stelle bei Stefan Seifried für die Möglichkeit diese Arbeit schreiben zu können. Bei etwaigen Fragen stand er mir stehts mit Ratschlägen zur Seite und übernahm auch das Korrekturlesen der Arbeit. Dank gilt ebenfalls Michael Leitner, der bei Problemen mit der Brandmeldezentralle zur Hilfe stand und Herrn Prof. Wolfgang Kastner, der sogar an Wochenenden auf Fragen umgehend antwortete. Weiterer Dank gilt auch Larissa für das Korrekturlesen der Arbeit auf Grammatik und Rechtschreibfehler. Bedanken möchte ich mich außerdem bei Familie und Freunden für die Unterstützung während des Studiums.

Kurzfassung

Zahlreiche Gebäude verfügen oft über zentrale Brandmeldeanlagen mit einer Vielzahl an Feuermeldern, die Parameter wie Temperatur, Kohlenmonoxid oder Rauch messen können. Nutzen dieser Werte ist nicht oder nur begrenzt möglich, da viele Brandmeldeanlagen proprietäre Protokolle verwenden und somit andere Systeme nicht auf die Sensorwerte zugreifen können. Zunächst werden in dieser Arbeit die bestehenden Technologien zur Konvertierung von Werten zwischen verschiedenen Protokollen mithilfe von Gateways vorgestellt. Im Anschluss wird mithilfe eines Ultra-Kompakt-Industrie-PCs, der als speicherprogrammierbare Steuerung fungiert, ein BACnet Gateway entwickelt, der Sensorwerte BACnet-fähigen Geräten zugänglich macht. So sollen auf Basis der Sensorwerte der Brandmeldeanlage weitere Anwendungsmöglichkeiten für die Gebäudeautomation und Industrie 4.0 geschaffen werden.

Abstract

Buildings are often equipped with fire alarm systems, containing hundreds or even thousands of fire detectors. Detectors carry multiple sensors, measuring temperature, carbon monoxide or smoke levels.These data points are useful for purposes other than fire detection, but proprietary protocols impede their widespread use. At first, this thesis will show existing technologies to translate information between protocols with help of gateways. Thereafter, an ultra-compact Industrial PC running a software programmable logic controller is used to create a BACnet gateway, to transform the data entities of the fire alarm system to a BACnet representation that can be read by BACnet capable devices. This way, BACnet accessible sensor values should enable enhanced building automation and Industry 4.0 applications.

Contents

Kurzfassung **vii**

Abstract **ix**

Contents **xi**

1 Introduction **1**
1.1 Motivation and Problem Statement 1
1.2 Aim of the Work 2
1.3 Methodology 2
1.4 Structure of the work 2

2 State of the Art **3**
2.1 BACnet 3
2.2 Fire Alarm System with BACnet 6
2.3 BACnet Protocol Gateways 8

3 Design and Implementation **13**
3.1 PLC - Fire Alarm System Communication 14
3.2 Beckhoff BACnet API 16
3.3 Beckhoff PLC BACnet Information Model 22
3.4 Beckhoff BACnet Stack Results 27
3.5 Life Safety Point and Life Safety Zone Information Model 28
3.6 Data Transfer TwinCAT to C# 34
3.7 Life Safety Objects with C# BACnet Library 37
3.8 C# BACnet Server Results 40

4 Conclusion and Further Work **43**
4.1 Beckhoff vs C# BACnet Stack 43
4.2 Conclusion 44
4.3 Further work 44

List of Figures **45**

List of Tables 47

List of Algorithms 49

Bibliography 51

CHAPTER 1
Introduction

1.1 Motivation and Problem Statement

Regulatory requirements call for fire alarm systems in buildings. Therefore, various forms of fire detectors are a common sight. Up until now, fire alarm systems, as an integral safety system of modern buildings, served the sole purpose of alerting users and preventing damage. However, the different sensors in fire detectors can be put to a second use in building automation and Industry 4.0 scenarios. Sensors like Carbon monoxide (CO) or temperature can be used for building automation and smart climate control. Given a fire alarm, a building automation system may also perform secondary tasks like switching on all lights and open blinds. In case of a real fire, information from fire alarm systems (location of the sensor that triggered the alarm) and building automation systems (e.g., information from motion detectors) may be combined to check if individuals are still in the affected area. Furthermore, a large fire alarm system consists of thousands of sensors, which could allow Industry 4.0 use cases for sophisticated tasks like remote monitoring and predictive maintenance.

To enable these use cases, access to the information of fire alarm systems sensors is needed. Direct access is hampered due to the use of proprietary protocols for fire alarm systems. Hence, a protocol-gateway is needed to improve interconnectivity between the trades in building automation. The big advantage of a gateway solution is that the automation pyramid is flattened, and the management level doesn't need to transfer and translate data between different systems. Therefore, the role of protocol gateways in the implementation of importance in Industry 4.0 is necessary. [1]

Another advantage of using already installed fire alarm systems is the conservation of resources. No additional wires need to be installed, and e-waste can be reduced by avoiding installation of new sensors. Besides that, smart climate control can save a vast amount of energy in large buildings when being equipped with a variety of sensors. [2]

Moreover, the technical requirements of fire alarm system sensors are high, therefore fire alarm system sensors have a high demand on reliability and availability. [3], [4]

1.2 Aim of the Work

The main goal of this thesis is to test whether a PLC is suited to translate proprietary protocol of fire alarm systems to a popular building automation protocol like BACnet/IP. At first, existing literature is evaluated to study already existing solutions. After that, a BACnet information model is created to manage the mapping to BACnet. Based on the domain model and the instance model of a fire alarm system, the BACnet information model should be automatically created. These goals ought to be prototypically implemented with help of a Beckhoff-PLC to prove the viability of the fire alarm system to BACnet protocol gateway.

1.3 Methodology

First, fundamental requirements of the BACnet protocol gateway and information on state of the art are gathered based on the review of existing literature. After that, a proof-of-concept (PoC) implementation is used to gather quantitative as well as qualitative data. The evaluation will be done in cooperation with a manufacturer of the fire alarm systems, Schrack-Seconet AG [5]. A major part of the PoC evaluation is the comparison of the Beckhoff-BACnet stack with third party solutions (e.g., C#-stack). Requirements of the BACnet protocol gateway and gathered data from the prototypical implementation will be assessed to find weaknesses. For instance, the maximum number of supported fire alarm system elements and usability of a proprietary BACnet mapping and BACnet Life Safety objects will be analyzed.

1.4 Structure of the work

First, the chapter **State-of-the-Art** will show the relevant points and objects of BACnet. This is followed by the description of two fire alarm systems which support BACnet natively and a short summary of three different BACnet gateway solutions. Chapter **Design and Implementation** will start with the Beckhoff-PLC and how to access the fire alarm system and create a BACnet server with the Beckhoff stack. Afterwards, the BACnet information model for the Beckhoff BACnet stack will be shown, tested and evaluated. Last but not least, this chapter covers the implementation and testing of the BACnet Life Safety Point and Life Safety Zone information model with an open source C# BACnet stack. The final chapter **Conclusion and Further Work** compares the two information models, summarizes the findings of this thesis, and shows further options to optimize found weaknesses.

CHAPTER 2

State of the Art

2.1 BACnet

BACnet stands for "Building Automation and Control Networks" and is a networking protocol for building automation and control that is standardized in DIN EN ISO 16484-5 [6]. The goal of BACnet is a cross-vendor protocol that connects different parts of building automation like lighting, heating, access control, fire alarm systems and more to allow communication between systems along with central control. BACnet can be used with different underlying networks like LonTalk or KNX. Incompatible devices can be made compatible with the help of protocol gateways. BACnet devices are represented by objects consisting of multiple properties that must be accessible via standardized BACnet services, allowing the actual implementation of the BACnet device to be proprietary. BACnet services are used to access objects, that include direct access, subscriptions as well as alarm and event management. BACnet provides standard object types for fire alarm systems, the Life Safety Zone (LSZ) and Life Safety Point (LSP). The big advantage of LSZ and LSP, interoperability, is opposed by the limited number of standardized properties and therefore of limited use for alternative and advanced use cases. Proprietary object types can be created, or standard object types can be extended at the expense of compatibility with other systems. [6], [7]

2.1.1 Standardized Object Properties

Standardized object types consist of required and optional properties which can be readable, writable or both. The number and type of properties differs from object type to object type. Alongside property data types like *boolean* and *int*, BACnet uses Application Protocol Data Units (APDUs) for many properties. APDUs encode the meaning of properties by mapping numbers to meaningful states e.g., *BACnetAccessEvent* maps none=0,granted=1,muster=2,... . Some APDUs can be expanded with proprietary values at the cost of compatibility. The DIN EN ISO 16484-5 [6] defines 60 standard object types, covering a wide range of applications. [6]

2.1.2 Life Safety Point

The Life Safety Point is used to represent devices like manual call points, fire detectors, sirens and more. It contains 14 required properties and 27 optional properties. However, the list of defined properties provides only one property ("Direct_Reading") suitable for encoding an analog sensor value. The optional properties can also support the use of intrinsic reporting, meaning that an LSP or LSZ can generate life-safety-alarm-, alarm- and fault-events based on changes of life safety state. The life-safety-events are transmitted with higher priority to ensure prioritization compared to normal notifications. LSP and LSZ also contain a notification class property, which sets the class that will be notified when an event occurs. Polling LSPs for changes is computational and network resource intensive, therefore the LSP property “Present_Value” and “Status_Flags” supports change of value (CoV) notifications to notify subscribers in case of state changes. The following table shows relevant properties of the LSP defined in the DIN EN ISO 16484-5 and a brief description [6].

Property	**Datatype**	**Function**
Object_Identifier	BACnetObjectIdentifier	Identifies object within BACnet
Object_Name	CharacterString	Unique Object Name within the BACnet device
Object_Type	BACnetObjectType	LIFE_SAFETY_POINT
Present_Value	BACnetLifeSafetyState	The state of the LSP e.g., Alarm, Active, Deactivation, . . .
Tracking_Value	BACnetLifeSafetyState	Non latched present-value
Description	CharacterString	Textual description of the object
Device_Type	CharacterString	Textual description of the device type, e.g., siren, printer, zone, . . .
Status_Flags	BACnetStatusFlags	four status flags (IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE)
Event_State	BACnetEventState	State of object, always normal without intrinsic reporting
Reliability	BACnetReliability	Describes the reliability of the present value
Out_of_Service	BOOLEAN	Indicates if the LSP is out of service
Mode	BACnetLifeSafetyMode	Operating mode of the LSP
Accepted_Modes	BACnetLIST of BACnetLifeSafetyMode	List of accepted modes that can be written to property mode

Table 2.1 continued from previous page

Silenced	BACnetSilencedState	Indicates the latest change that caused an audible signal
Operation_Expected	BACnetLifeSafety -Operation	Operation expected by the LSP to handle a life safety condition
Direct_Reading	REAL	Analog device value can be represented e.g., temperature, smoke, ...
Following Properties are only needed for Intrinsic reporting		
Notification_Class	Unsigned	Class that gets event and alarm notifications
Life_Safety_Alarm -_Values	BACnetLIST of BACnetLifeSafetyState	List of Present_Values that should trigger Life Safety Alarms
Alarm_Values	BACnetLIST of BACnetLifeSafetyState	List of Present_Values that should trigger Alarms
Fault_Values	BACnetLIST of BACnetLifeSafetyState	List of Present_Values that should trigger Faults
Time_Delay	Unsigned	Time the present value has to hold to trigger an alarm or event
Event_Enable	BACnetEvent -TransitionBits	Three flags (to_offnormal, to_normal, to_fault) to enable and disable the transmission of events
Acked_Transitions	BACnetEvent -TransitionBits	Three flags (to_offnormal, to_normal, to_fault) that represent the acknowledgement state
Notify_Type	BACnetNotifyType	Set the notification type to Alarm or Events
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	Latest timestamp of events (to_offnormal, to_normal, to_fault)
Event_Detection -_Enable	BOOLEAN	True when intrinsic reporting is enabled otherwise false

Table 2.1: Life Safety Point properties explanation [6]

2.1.3 Life Safety Zone

The Life Safety Zone contains nearly the same list of properties as the LSP, with the main difference that it contains the read property **Zone_Members**. This property can be used to group multiple LSPs together into one zone to depict the physical fire alarm system structure [6].

2.1.4 Combination From Multiple Standardized Objects

Not all BACnet devices fully support LSP, LSZ and proprietary objects, e.g., the Beckhoff BACnet stack. [8] To ensure widespread compatibility, the fire alarm system information model must be built based on a minimal set of fundamental BACnet object types. This can be accomplished by grouping multiple BACnet objects into a structured view, where the structured view is a fire alarm Element of the fire alarm system. A disadvantage of this information model is that standardized BACnet objects contain unnecessary properties that increase overhead and reduce usability. Moreover, interoperability is reduced compared to LSP and LSZ.

2.1.5 Proprietary Object Types

Some BACnet stacks also support the creation of proprietary objects that combine an arbitrary number of properties that can be tailor-made to a specify implementation. The downside of proprietary objects is the limited interoperability with other BACnet devices.

2.1.6 Extension of Standardized Object Types

The best solution of both worlds is the use of standardized BACnet objects extended with proprietary properties. All BACnet devices should be able to read the properties defined by the DIN EN ISO 16484-5:2017-12 [6]. The added proprietary properties can be read by BACnet devices that are aware of the added properties. Using extended standardized BACnet objects enables IoT and Industry 4.0 applications.

2.2 Fire Alarm System with BACnet

There are fire alarm systems with BACnet support available on the market. This chapter will show how they connect to the fire alarm control panel (FACP) and, if publicly available, which BACnet object types are used to model the BACnet information model and how the translation from an FACP to BACnet is done.

2.2.1 Zettlerfire UC-8112-LX BACnet Converter

Zettlerfire offers two solutions to make their fire alarm systems BACnet capable. On one hand, a stand-alone solution using a MZX technology panel can be used. Details on the stand-alone solution could not be found. On the other hand, the industrial computer *UC-8112-LX* can take the role of the BACnet converter. The industrial computer is powered by a RISC-Processor and connects to the fire alarm system via the fire alarm systems network called TLI800EN. The converter runs a specialized firmware that needs to be uploaded via a PC to work as a BACnet gateway. Information accessible via BACnet information model include alarms of LSP and LSZ, system faults, warnings, pre-alarms, LSP and LSZ isolation and analogue values of automatic detectors. Supported commands are reset, sounders on/off, silence, evacuation and isolation of points and zones. The

Zettlerfire datasheet provides neither information on how the translation takes place, nor shows information on the utilized BACnet objects. The figure 2.1 shows an example workflow of the BACnet gateway. The MZX Panel Network (1) is connected to the MZX BACnet interface (2) which is configured with the configuration download tool (3). Via Ethernet, the MZX BACnet interface (2) is connected to the BACnet gateway (4). The BACnet management workstation (6) is connected to the BACnet gateway (4) via BACnet capable LAN (5). An upper limit of supported fire alarm elements by one BACnet gateway is not stated in the datasheet [9], [10].

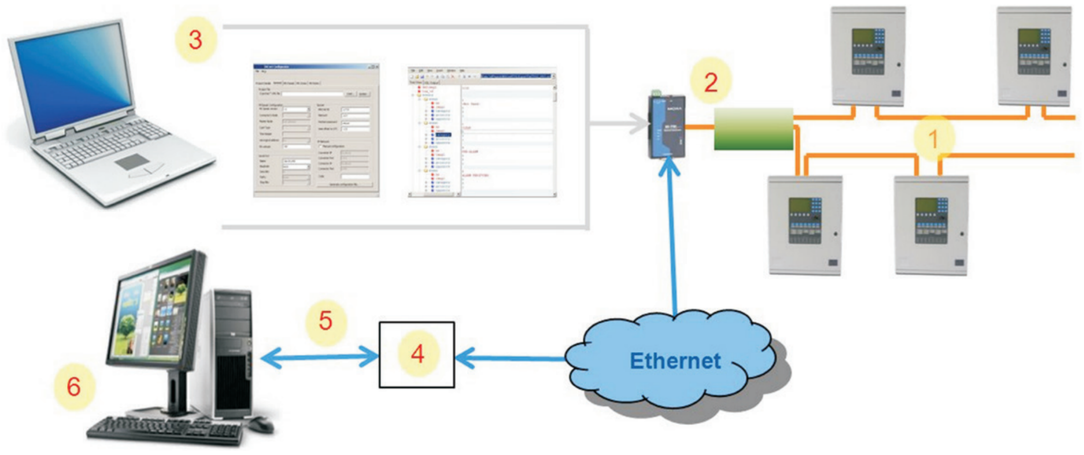


Figure 2.1: Zettlerfire UC-8112-LX BACnet converter workflow [10]

2.2.2 Notifier (Honeywell) BACnet Gateway

Another system is the BACnet gateway from Notifier, which is a division of Honeywell. Honeywell was part of the project-committee SPC 135P that created BACnet. The BACNET_GW-3 from Notifier can be connected to the Notifirenet via a network port to monitor up to 14 Notifirenet systems with up to 15000 objects. For additional nodes, multiple gateways can be used in parallel. When no Notifirenet is used, the BACnet gateway can be connected to the FACP directly. Figure 2.2 shows the system architecture used to connect the FACP to the BACnet gateway. The BACnet gateway is configurable via a built-in browser accessible configuration tool. The gateway supports device-, binary output-, LSP and LSZ, multi-state input-objects, and event/alarm notifications. The objects are used to represent a wide range of devices like zones, detectors, batteries, loops, AC power and more. Possible values of the BACnet properties for LSP and LSZ, Multi-state Input and Binary Output are listed in the *BACnet Gateway-3 Installation and Operation Manual* [11]. Interestingly, the DIRECT_READING property of LSP and LSZ is used to describe the percentage of an alarm and not a temperature or CO value of an analog sensor. Information on the exact hardware, and how the translations takes place, is not publicly available [11], [12].

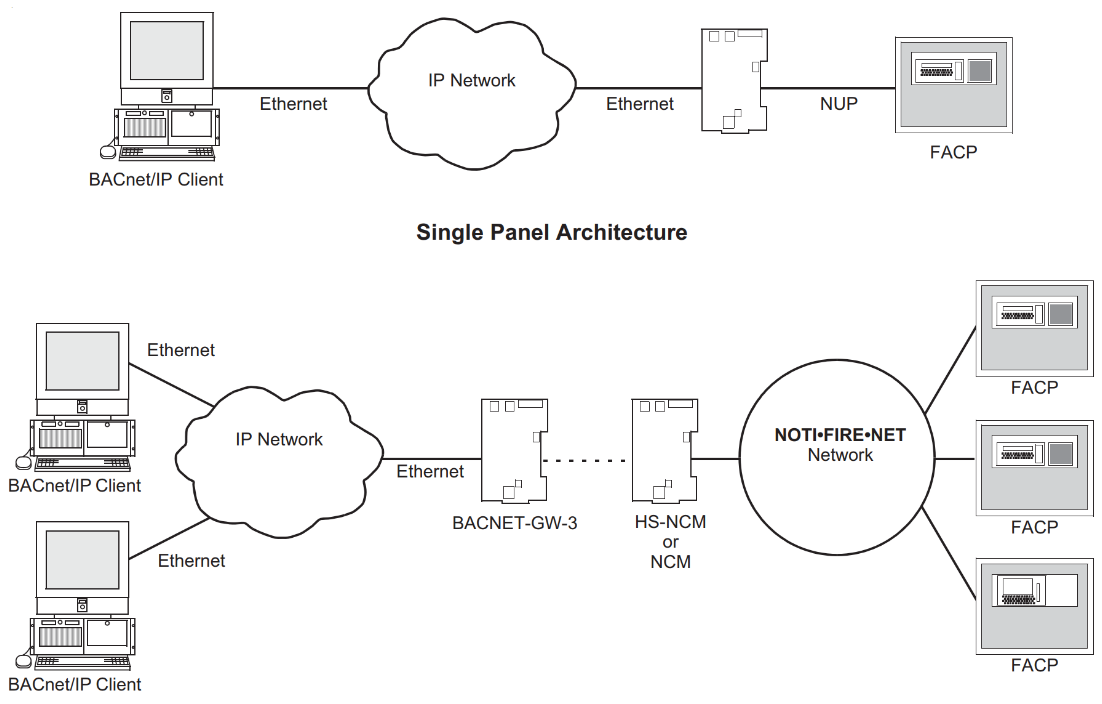


Figure 2.2: NOTIFIER BACnet gateway system architecture [12]

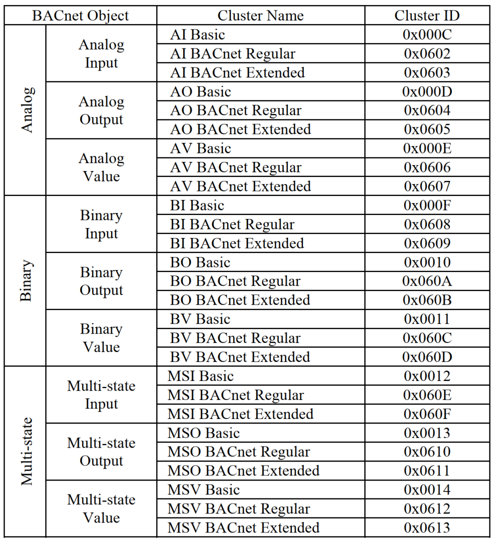
2.3 BACnet Protocol Gateways

This section will show three implementations for BACnet gateways, one of them with fire alarm systems in mind. The other two will focus on the translation between BACnet-Zigbee and BACnet-EnOcean.

2.3.1 Implementation of a BACnet-ZigBee Gateway [13]

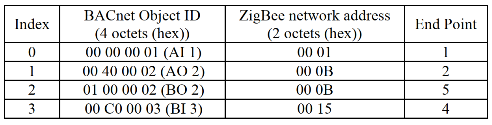
At the time of writing, BACnet had no support for ZigBee, therefore the researcher of the *Implementation of a BACnet-ZigBee Gateway* paper [13] decided to create a BACnet-ZigBee gateway to allow BACnet to be used via wireless ZigBee channels. Besides a BACnet to ZigBee gateway, BACnet over ZigBee (BoZ) is possible as well. BoZ implies that BACnet network protocol data units (NPDUs) are tunneled over ZigBee. Commercial building automation (CBA) profiles are specified by the ZigBee Alliance to define tunnel clusters for BoZ and ZigBee clusters to be mapped to BACnet objects. Besides the cluster to BACnet object mapping, a second mapping table for data mapping is needed. The second mapping table consists of the ZigBee network address, end point and BACnet object IDs. The two mapping tables are shown in Figure 2.3. [13]

The implementation of the BACnet gateway uses ZigBee and BACnet hardware modules, the two hardware modules communicate via RS-232 with each other. As BACnet data link protocol, the BACnet/MSTP protocol is used. The figure below shows the hardware



BACnet Object		Cluster Name	Cluster ID
Analog	Analog Input	AI Basic	0x000C
		AI BACnet Regular	0x0602
		AI BACnet Extended	0x0603
	Analog Output	AO Basic	0x000D
		AO BACnet Regular	0x0604
		AO BACnet Extended	0x0605
	Analog Value	AV Basic	0x000E
		AV BACnet Regular	0x0606
		AV BACnet Extended	0x0607
Binary	Binary Input	BI Basic	0x000F
		BI BACnet Regular	0x0608
		BI BACnet Extended	0x0609
	Binary Output	BO Basic	0x0010
		BO BACnet Regular	0x060A
		BO BACnet Extended	0x060B
	Binary Value	BV Basic	0x0011
		BV BACnet Regular	0x060C
		BV BACnet Extended	0x060D
Multi-state	Multi-state Input	MSI Basic	0x0012
		MSI BACnet Regular	0x060E
		MSI BACnet Extended	0x060F
	Multi-state Output	MSO Basic	0x0013
		MSO BACnet Regular	0x0610
		MSO BACnet Extended	0x0611
	Multi-state Value	MSV Basic	0x0014
		MSV BACnet Regular	0x0612
		MSV BACnet Extended	0x0613

(a) cluster to BACnet object mapping

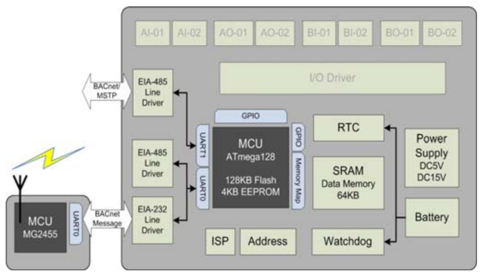


Index	BACnet Object ID (4 octets (hex))	ZigBee network address (2 octets (hex))	End Point
0	00 00 00 01 (AI 1)	00 01	1
1	00 40 00 02 (AO 2)	00 0B	2
2	01 00 00 02 (BO 2)	00 0B	5
3	00 C0 00 03 (BI 3)	00 15	4

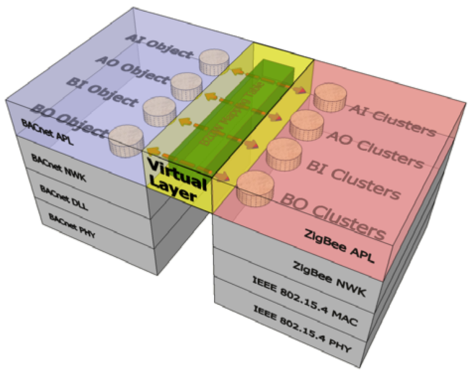
(b) data mapping table

Figure 2.3: ZigBee BACnet gateway mapping table [13]

structure as well as the architecture of the gateway. The MCU used is a ATmega128 with 128 KB of Flash memory. The gateway translates the ZigBee entities attribute, command and cluster to the BACnet entities property, service and object and the other way around. Address translations are done with a mapping table (see Figure 2.3b). The experimental testing of the BACnet gateway showed the successful translation between BACnet and ZigBee. Figure 2.4 shows the hardware structure and architecture used for the BACnet gateway. [13]



(a) BACnet gateway hardware structure



(b) BACnet gateway architecture

Figure 2.4: BACnet gateway hardware structure and architecture [13]

2.3.2 A BMS Client and Gateway Using BACnet Protocol [14]

The goal of the *A BMS Client and Gateway Using BACnet Protocol* paper [14] is to use BACnet for a Building Management System (BMS) and to develop a BMS client application and a gateway to interface fire alarm panels, in this case an FACP from Bosch. The paper is mainly focused on the BMS and BACnet protocol in general, the BACnet gateway is only mentioned briefly. Moreover, the BACnet gateway is running on a computer in the form of software and uses two network ports of the computer to communicate with the FACP and BACnet. Translation works in both directions, sending commands from BACnet to the FACP and translating the state of the fire alarm system to BACnet objects. Alarms and events are generated by the gateway based on the fire alarm system state. The elements from the fire alarm system are converted to BACnet objects, but the paper does neither mention how exactly the translation takes place, nor which BACnet object types are used. Figure 2.5 illustrates system architecture of the BACnet gateway interfacing with the FACP. The BMS is connected to the gateway PC via LAN, which is connected to the fire alarm system network. No test results, like the number of supported elements and functioning of translation commands and alarms/events of the BACnet gateway, are shown. [14]

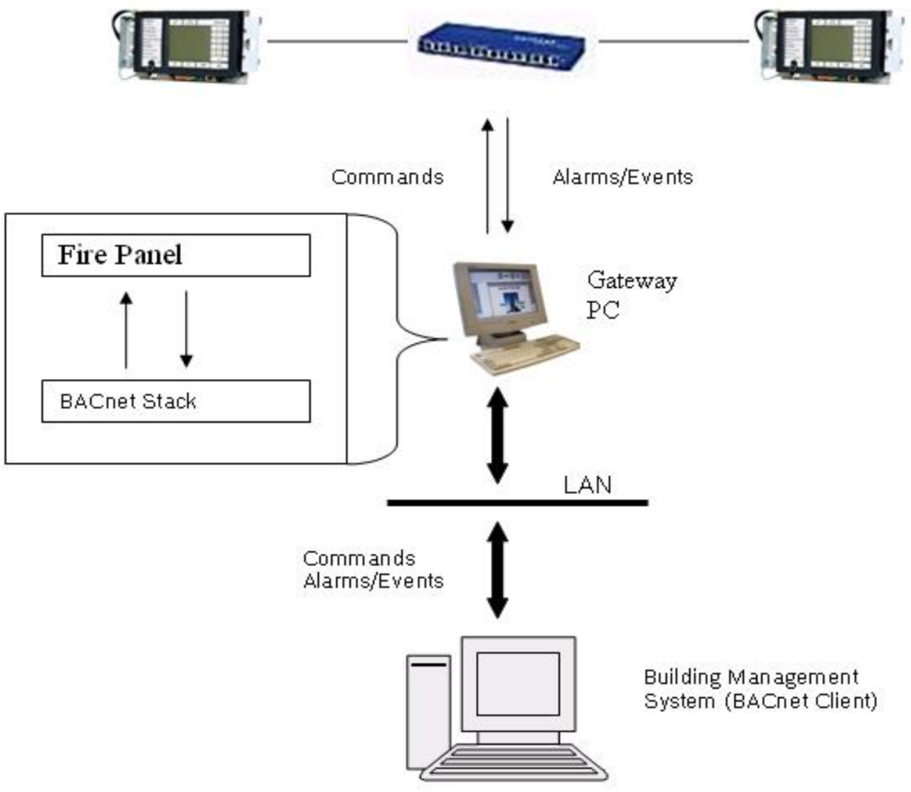


Figure 2.5: BMS client and gateway system architecture [14]

2.3.3 Implementation of a BACnet-EnOcean Gateway in Buildings [15]

The aim of the *Implementation of a BACnet-EnOcean Gateway in Buildings* paper [15] is to test if it is feasible to add EnOcean energy harvesting devices to BACnet with help of a protocol gateway. The gateway needs to do address and property-attribute translation. Therefore, the researchers categorized the EnOcean devices into three groups. First, the *energy-harvesting input device*, *energy-harvesting-output device* and a *Line powered device* need to be mapped to a BACnet object. BACnet analog inputs, analog outputs, binary inputs, and binary outputs are selected as target objects. The read and write requests are translated by the gateway from BACnet to EnOcean and vice versa. The main hardware parts of the gateway are the BACnet module, EnOcean transmitter, connect selection part (used to switch between the PC, transmitter and BACnet module) and PC interface as shown in Figure 2.6. The BACnet module contains a 32-bit ARM cortex MCU with 40 KB of RAM. The PC interface is used for monitoring and to download the gateway protocol source code to the BACnet module. [15]

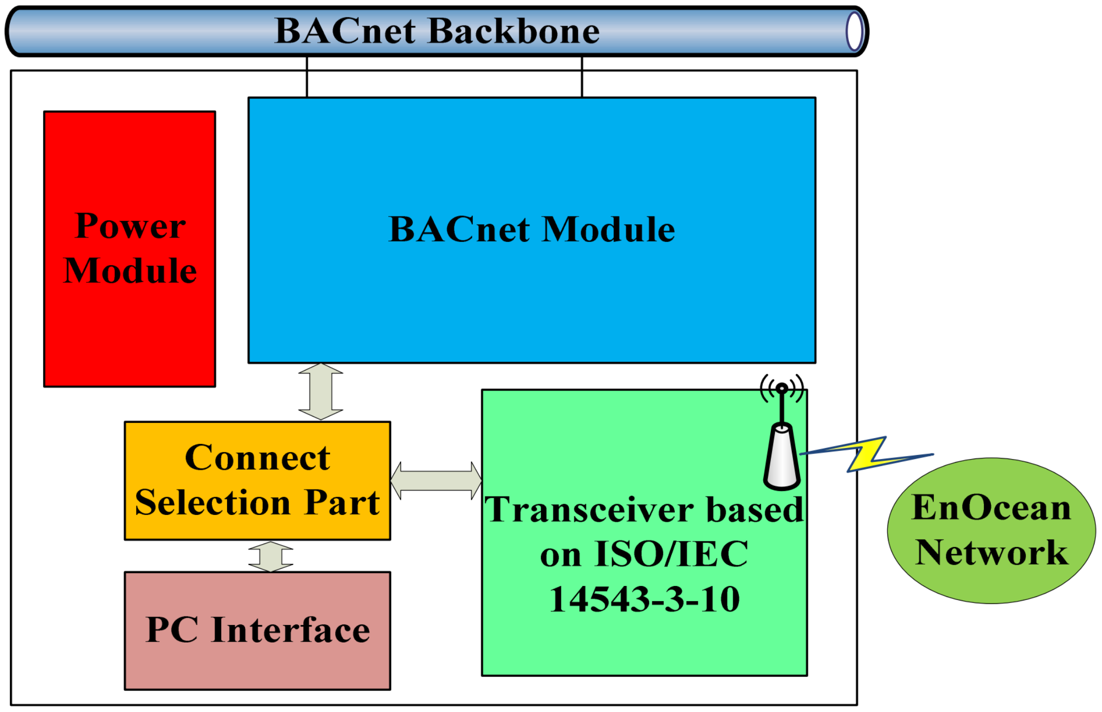
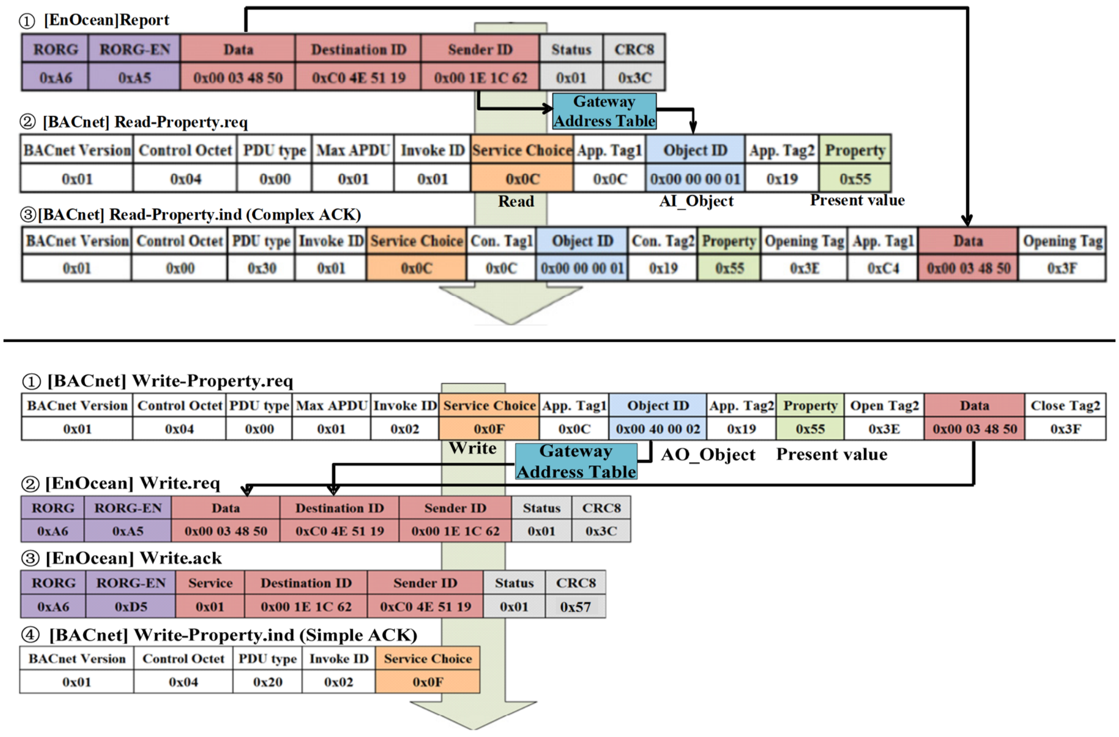


Figure 2.6: EnOcean to BACnet gateway hardware architecture [15]

A PoC setup with EnOcean devices and BACnet gateway have been used to test the function of the gateway. Input and output objects were accessed via BACnet and the EnOcean telegrams were analyzed with the analyzer DolphinView. As Figure 2.7 shows, the report-telegram from EnOcean was successfully transferred to BACnet read property. The same results were shown when testing a write-request. The data as well as the source and destination addresses were correctly translated. [15]



① [EnOcean]Report

RORG	RORG-EN	Data	Destination ID	Sender ID	Status	CRC8
0xA6	0xA5	0x00 03 48 50	0xC0 4E 51 19	0x00 1E 1C 62	0x01	0x3C

② [BACnet] Read-Property.req

BACnet Version	Control Octet	PDU type	Max APDU	Invoke ID	Service Choice	App. Tag1	Object ID	App. Tag2	Property
0x01	0x04	0x00	0x01	0x01	0x0C	0x0C	0x00 00 00 01	0x19	0x55

③[BACnet] Read-Property.ind (Complex ACK)

BACnet Version	Control Octet	PDU type	Invoke ID	Service Choice	Con. Tag1	Object ID	Con. Tag2	Property	Opening Tag	App. Tag1	Data	Opening Tag
0x01	0x00	0x30	0x01	0x0C	0x0C	0x00 00 00 01	0x19	0x55	0x3E	0xC4	0x00 03 48 50	0x3F

① [BACnet] Write-Property.req

BACnet Version	Control Octet	PDU type	Max APDU	Invoke ID	Service Choice	App. Tag1	Object ID	App. Tag2	Property	Open Tag2	Data	Close Tag2
0x01	0x04	0x00	0x01	0x02	0x0F	0x0C	0x00 40 00 02	0x19	0x55	0x3E	0x00 03 48 50	0x3F

② [EnOcean] Write.req

RORG	RORG-EN	Data	Destination ID	Sender ID	Status	CRC8
0xA6	0xA5	0x00 03 48 50	0xC0 4E 51 19	0x00 1E 1C 62	0x01	0x3C

③ [EnOcean] Write.ack

RORG	RORG-EN	Service	Destination ID	Sender ID	Status	CRC8
0xA6	0xD5	0x01	0x00 1E 1C 62	0xC0 4E 51 19	0x01	0x57

④ [BACnet] Write-Property.ind (Simple ACK)

BACnet Version	Control Octet	PDU type	Invoke ID	Service Choice
0x01	0x04	0x20	0x02	0x0F

Figure 2.7: EnOcean report-telegram to BACnet [15]

CHAPTER 3

Design and Implementation

This chapter will describe problems and solutions of the design and implementation process. First, the Beckhoff PLC will be described, followed by the Beckhoff PLC *Integral-Standardprotokoll over IP* (ISP-IP) API to access the FACP. After that, the Beckhoff BACnet stack and BACnet information model will be explained. Lastly, a C# BACnet stack to create LSP and LSZ objects will be shown. Figure 3.1 shows the system architecture, the green part is only needed for the C# BACnet stack solution. The C# BACnet stack is running on a development PC instead of the ultra-compact Industrial PC for testing purposes only. When using the Beckhoff BACnet stack, the BACnet server is running on the PLC.

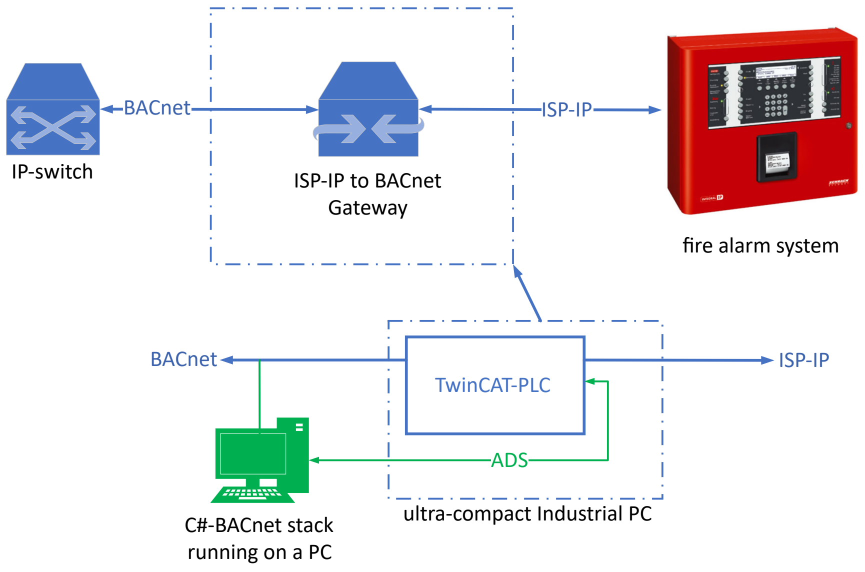


Figure 3.1: System Architecture

3.1 PLC - Fire Alarm System Communication

The Beckhoff C6030 is an ultra-compact Industrial PC (UcIPC) using Windows 10 IoT Enterprise as operating system. 8 GB of memory and an Intel x86 CPU power the UcIPC. [16] TwinCAT-PLC runs on the UcIPC to provide PLC functionality. Programming the PLC is done via the integrated development environment (IDE) called TwinCAT 3 Engineering. [17] The UcIPC combines the performance of a PC with the functionality of a PLC and is therefore targeted for IoT and Industrial 4.0 use cases. [16] Figure 3.2 shows the UcIPC. Especially, two of the four Ethernet ports are needed to connect to multiple networks, which is a necessity, in order to use the UcIPC as a protocol gateway between two networks.

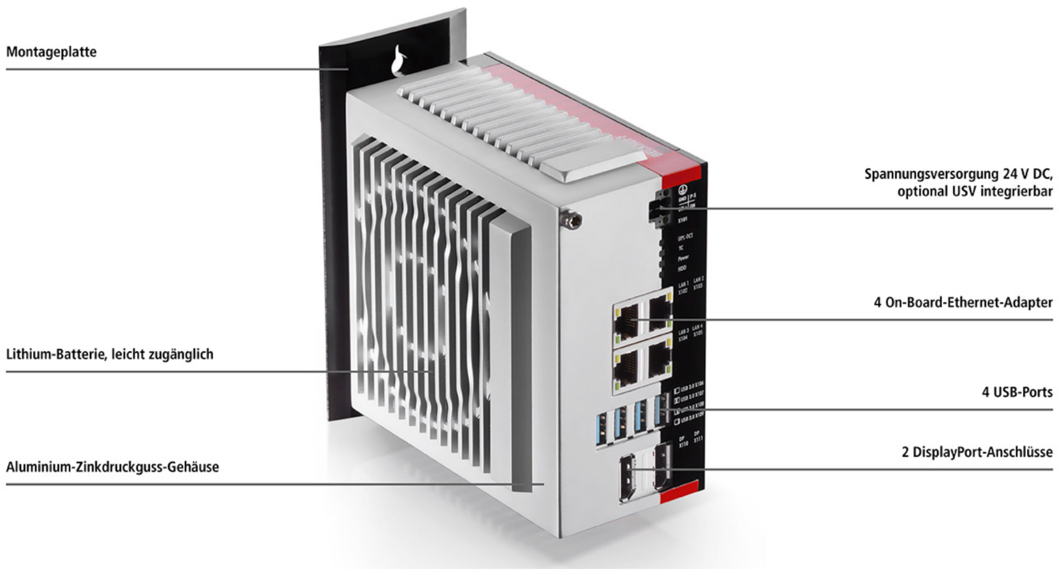

Montageplatte
Lithium-Batterie, leicht zugänglich
Aluminium-Zinkdruckguss-Gehäuse
Spannungsversorgung 24 V DC, optional USV integrierbar
4 On-Board-Ethernet-Adapter
4 USB-Ports
2 DisplayPort-Anschlüsse

Figure 3.2: UcIPC Beckhoff C6030 [16]

3.1.1 Beckhoff-PLC as Gateway

In order to allow a PLC to map a fire alarm system to a BACnet representation, the PLC needs to be connected to the FACP. The connection is done via IP over Ethernet, therefore the PLC can be directly connected to the FACP. The Windows 10 IoT Enterprise runs a service called *Integral-Standardprotokoll over IP* (ISP-IP) that is supplied from Schrack-Seconet AG [5] and can directly communicate with the FACP. An Application Programming Interface (API) is available to access the ISP-IP service from the Beckhoff PLC environment TwinCAT3. [18] The extracted data from the FACP is converted to BACnet objects with the Beckhoff BACnet library or transferred to a C# program that creates a BACnet/IP-Server with a third party BACnet library.

3.1.2 State Machine Procedure to Read From the FACP

The ISP-IP-Service has to be configured with the IP-address and an XML file that holds the description of the fire alarm system configuration. The IP configuration and XML file provided by Schrack-Seconet AG [5]. To access the FACP within TwinCAT3, the Tc_ISP-IP Library must be added first, after that the elements of the fire alarm system can be accessed via function blocks. The easiest way to access the FACP is to create a state machine that updates the BACnet objects repeatedly. An example for a state machine that reads from the FACP was provided by Schrack-Seconet [5]. Based on this state machine, the BACnet/IP gateway is created.

3.1.3 ISP-IP API for Beckhoff TwinCAT

The used state machine is shown in Figure 3.3. At the startup, the state **idle** waits for the timer to call for an update of the fire alarm system elements. When the call occurs, it prepares the ISP-IP function blocks to read from the FACP and switches to the **init** state, which resets all counters. **Init** is followed by a **start** state which checks if a FACP is connected. Furthermore, the number of elements from the fire alarm system is compared to the number set in the program, if it does not match, the PLC goes into error-mode. This is done to prevent configuration errors from going unnoticed. Unless an error occurred, the first 100 elements are read from the FACP. In case no further elements need to be fetched, the state **get_description** is entered to get the textual description of each fire alarm system element. If elements are missing, the state-machine goes to the state **getnext** to fetch 100 elements at a time till all elements are fetched. Only 100 elements are read per step to limit the duration of a state-machine cycle. Otherwise, the time per step can exceed the PLC cycle time. After all elements are read and received their description, the state-machine enters the **idle**-state and the procedure repeats itself anew. The elements' descriptions won't change while the fire alarm system is running. Therefore, descriptions are only fetched in the first cycle of the state machine.

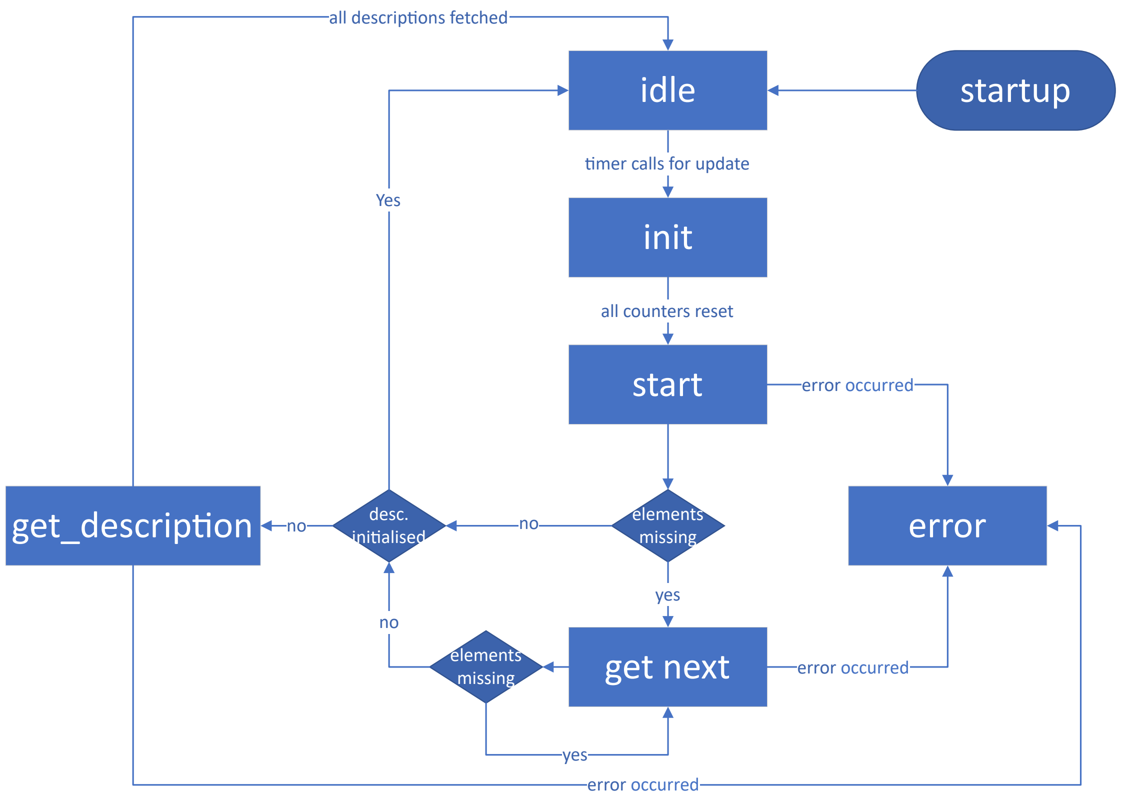


Figure 3.3: State machine

The ISP-IP API has a shortcoming, the description of the elements cannot handle special characters. This is a known issue and will be fixed in a future release. The states **start**, **getnext** and **get_description** need to be altered to map the elements to the BACnet information model, these changes are explained in section 3.3.3.

3.2 Beckhoff BACnet API

The first milestone is to use the BACnet API from Beckhoff to map the fire alarm system to the LSP and LSZ objects. But unfortunately, the BACnet implementation from Beckhoff does not support LSP and LSZ objects. Proprietary objects are not supported as well. Therefore, the fire alarm system needs to be modeled with the help of available BACnet objects. The Beckhoff-API only supports 38 object types, which are listed below (Figure 3.4). [8] To provide a BACnet information model with LSP and LSZ, TwinCAT-PLC needs to be connected to a third-party program that provides a BACnet implementation with LSZ and LSP objects. More detail on a C# implementation can be found in section 3.7.

- Analog-Input
- Analog-Output
- Analog-Value
- Averaging
- Binary-Input
- Binary-Output
- Binary-Value
- Calendar
- Command
- Device
- Event-Enrollment
- File
- Group
- Loop
- Multi-state Input
- Multi-state Output
- Multi-state Value
- Notification Class
- Schedule
- Structured View
- Program
- Pulse-Converter
- Trend-Log
- Trend-Log-Multiple
- Event-Log
- Bitstring-Value
- Character-String-Value
- Date-Pattern-Value
- Date-Value
- DateTime-Pattern-Value
- DateTime-Value
- Integer-Value
- LargeAnalog-Value
- OctetString-Value
- Positive-Integer-Value
- Time-Pattern-Value
- Time-Value

Figure 3.4: Supported object types

3.2.1 ISP-IP Datatype to Beckhoff BACnet Object Mapping

There are five different data types (DateTime, Byte, U/Integer, String) per node available from the ISP/IP API that need to be mapped to available BACnet object types. In addition, a BACnet object is needed to group objects from an ISP-IP Element. Table 3.1 shows the five data types and the potential BACnet object types that are suitable for mapping.

Description	Datatype	BACnet object types	BACnet Datatype
Timestamp	Date Time	Datetime Value	ST_BA_DateTime
Elementstate	Byte	Positive-Integer-Value	DINT
		Multi-state Value	UDINT
		Analog Value	REAL
Element Number, Subtype,...	Unsigned-Integer 16bit	Large-Analog-Value	LREAL
		Analog Value	REAL
		Positive-Integer-Value	DINT
Temperature, CO,...	Integer 32bit	Large-Analog-Value	LREAL
		Analog-Value	REAL
		Integer-Value	DINT
Description	String	BACnet Primitive String	T_MAXString
		Structured View ->Property:Description	T_MAXString
Grouping one fire alarm system Element	Group multiple BACnet objects	Group	-
		Structured View	-

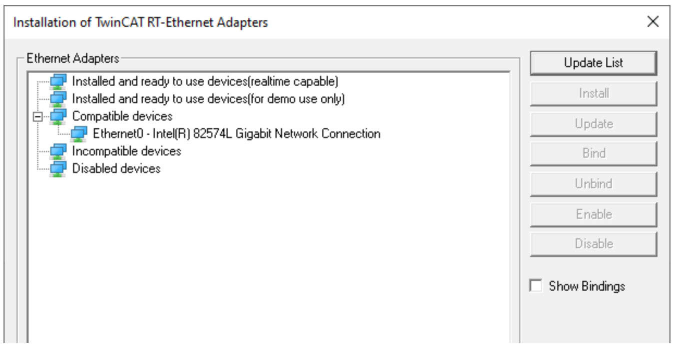
Table 3.1: ISP-IP Datatype to Beckhoff BACnet object mapping

3.2.2 Usage of Beckhoff BACnet API

The UcIPC must be configured so that it is compatible with the Beckhoff BACnet API. First, the BACnet/IP-Adapter must be configured and a BACnet/IP-Server added. After that, the PLC-program can be connected to the BACnet/IP-Server.

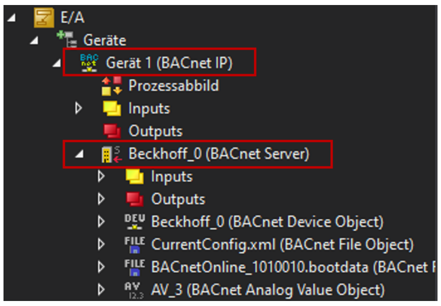
3.2.3 BACnet Network-Adapter and BACnet/IP-Server

A BACnet/IP-Adapter can be added in TwinCAT IDE at EA→Devices. When selecting the BACnet/IP-Adapter in the TwinCAT IDE, a real time network-adapter of the PLC can be assigned to the BACnet/IP-Adapter. The IDE must be connected to the PLC and a Beckhoff network real-time adapter driver must be installed for the network adapter to become available. To get the TwinCAT real time network driver running on the UcIPC, the driver needs to be installed directly on the UcIPC. This can be done via remote desktop, when connected, the driver can be installed via the tool **TcRteInstall** which can be found in the installation path of TwinCAT (e.g., C:\TwinCAT\3.1\System\TcRteInstall.exe). Figure 3.5a shows the RT-Ethernet Adapter's dialogue. After installation of the network driver, the network adapter can be selected in the TwinCAT IDE and assigned to the BACnet interface. Now, the BACnet/IP-Server can be added to the BACnet/IP-Adapter shown in Figure 3.5b.



Installed and ready to use devices(realtime capable)
Installed and ready to use devices(for demo use only)
Compatible devices
Ethernet0 - Intel(R) 82574L Gigabit Network Connection
Incompatible devices
Disabled devices

(a) Beckhoff TcRteInstall


E/A
Geräte
Gerät 1 (BACnet IP)
Prozessabbild
Inputs
Outputs
Beckhoff_0 (BACnet Server)
Inputs
Outputs
Beckhoff_0 (BACnet Device Object)
CurrentConfig.xml (BACnet File Object)
BACnetOnline_1010010.bootdata (BACnet F
AV_3 (BACnet Analog Value Object)

(b) BACnet/IP and BACnet/IP-Server

Figure 3.5: Beckhoff BACnet/IP-Server

3.2.4 Automapping BACnet Library Revision 12 and Revision 14

The Beckhoff BACnet library is available in two versions and is required to communicate with the BACnet/IP-Server already created. Revision 12 is available from the Beckhoff's Website, while revision 14 is not available online, yet. Both versions use completely different approaches to auto-generate and map BACnet objects, the mapping based on revision 12 did not work properly, therefore revision 14 was used. A more detailed explanation can be found in the next sections.

Automapping with BACnet Library Revision 12

Automapping with the BACnet library revision 12 is done by using special annotations after the variable declaration to automatically create a BACnet object. This enables the programmer to create BACnet objects from the PLC program instead of creating them by hand within the TwinCAT IDE. An example for such a special annotation can be found in the Beckhoff documentation and is shown in Algorithm 3.1. [19]

Algorithm 3.1: Automapping with revision 12 [19]

```
av0 AT %I* : REAL;          (* ~ (BACnet_ObjectType         :
AV             : NOLINK)
                                  (BACnet_ObjectIdentifier   :
100            : NOLINK)
                                  (BACnet_PresentValue
:                : )
                              *)
```

Before applying the program to the PLC, the BACnet objects need to be mapped to the BACnet/IP-Server module. That can be done via the mapping button in the BACnet/IP-

Server settings section, *PLC Automapping*. The problem with this approach is the lack of bidirectional updating of variables, regardless of settings. The values would only update from BACnet to the PLC program, but not vice versa. This problem could not be solved, therefore revision 14 was chosen. Moreover, the map-button from the BACnet/IP-Server needs to be pressed each time the program is changed and is not updated automatically at runtime.

Automapping with BACnet Library Revision 14

The BACnet library revision 14 was provided by the Beckhoff support team and creates BACnet objects via function blocks instead of special annotations. The function blocks are defined in the revision 14 library and can be used after importing the library. At the declaration, already known properties can be assigned to the BACnet object. If multiple objects are needed, an array can be used. The following code shows an example (Algorithm 3.2). [20]

Algorithm 3.2: Revision 14 function block declaration

```
state  :  FB_BACnet_MV := ( iParent  :=  element ,
                           bEnPgm  :=  TRUE ,
                           eNotifyType  :=  E_BACnet_NotifyType . eAlarm ,
                           aEventEnable  :=  [ 1 , 1 , 1 ] ,
                           aAlarmValues := [ 2 , 3 ] ,
                           nValPgm := 1
                           ) ;
```

The BACnet object is not online yet, to create the object the function block needs to be called. Hereby, it is important to call the function block cyclically to guarantee the update of properties. Furthermore, the function block should be called without property values. The values should be written directly to the function block to avoid unnecessary updates of the BACnet objects. [20] An example is shown in Algorithm 3.3.

Algorithm 3.3: Revision 14 function block updates

```
IF ( NOT  ( description_in  =  description_intern ) )  THEN
    description_intern  :=  description_in ;
    element . sDescription := description_intern ;
END_IF
```

This approach worked without a problem, and the values updated in the PLC program were also updated in the BACnet objects.

Space Limitations for BACnet Objects

The router cache from TwinCAT is used to store BACnet objects and is therefore limited. The TwinCAT router cache can be at most 1 GB in size, and only 60% of this memory can be used to store BACnet objects. One BACnet object requires an average of 20 KB, which results in a maximum of about 30 000 objects. [20] When the available memory is fully utilized and the program tries to create an additional object, an error message is shown, and no new object is created. This space limitation introduces a problem to the BACnet information model for large fire alarm systems. Upper limit of fire alarm elements depends on the used information model and can be found in Section 3.3. Moreover, the maximum number of objects is by default set to 10000, this number can be adjusted upwards at the BACnet/IP-Server-Adapter settings but is still limited by the available memory.

3.3 Beckhoff PLC BACnet Information Model

3.3.1 Large BACnet Information Model

The first BACnet information model design is based on all data points which are available from the ISP-IP API and are shown in Figure 3.6. Each Node within the fire alarm system network is represented by a structured view object called Elements [NUMBER] that groups all objects that are part of one fire alarm network node. A group consists of three large-analog-value objects that represent CO, the smoke level and temperature. The datetime-value object is used for the timestamp. Moreover, the group contains four additional structured view objects to provide an organized view. One structured view object is used to subgroup four positive-integer-value objects which hold information (network-number, element-type, element-number and sub-element-number) that belong to the key of the node. Each one of the other three structured view objects represents one state of a fire alarm system's element,where each node can have three states at the same time. One state contains six positive-integer-value objects to represent Function-Type, Sub-Type, State, Sub-State, Level and Information. This BACnet information model design would present a detailed view of each fire alarm system node, unfortunately the space requirements for this design are too large. Given a router cache of 1 GB and 20 KB per BACnet object, this design with 33 objects per Node would be limited to approximately 900 fire alarm system elements. That number is too small for large fire alarm systems, which can consist of multiple thousands of elements.

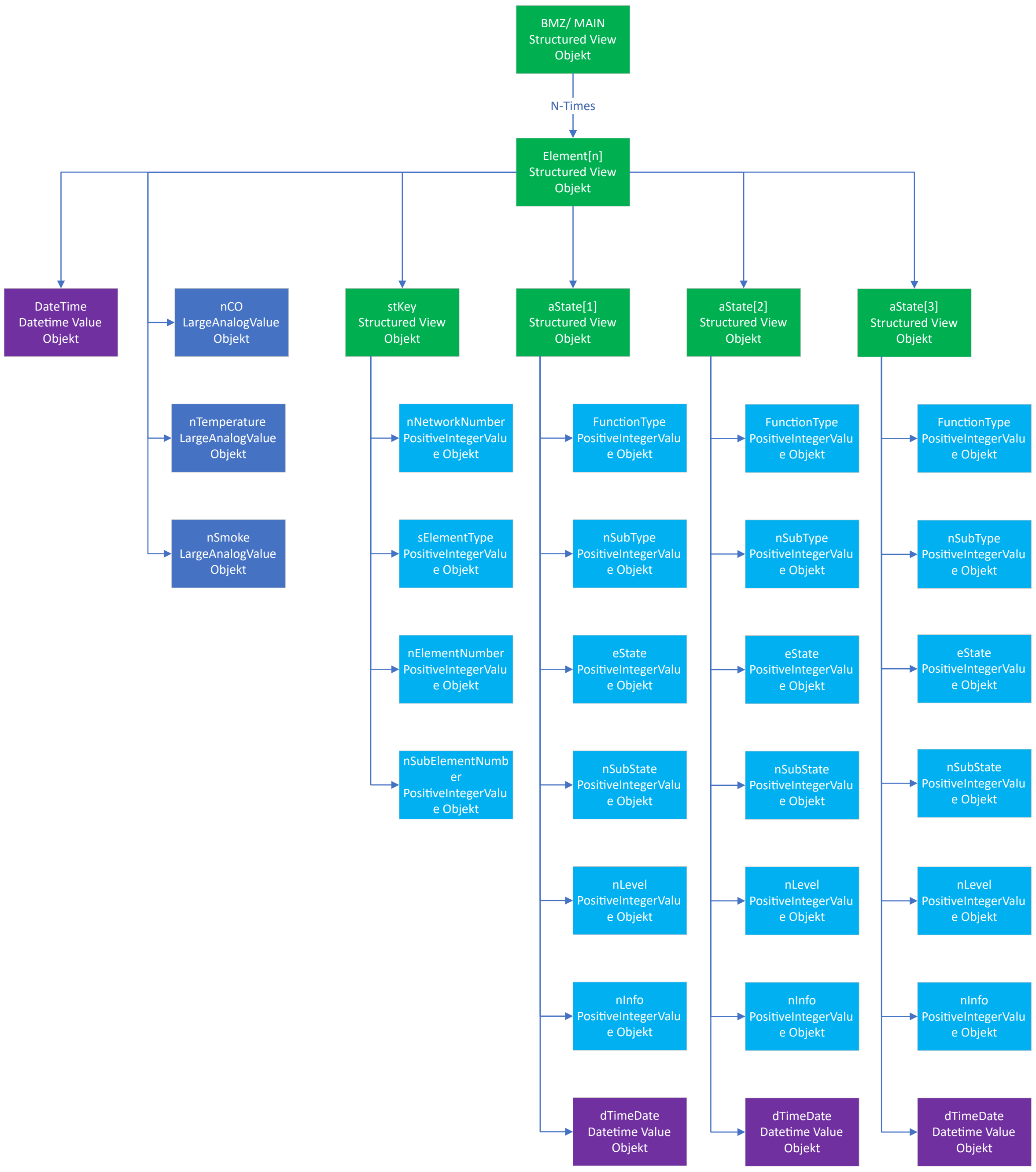
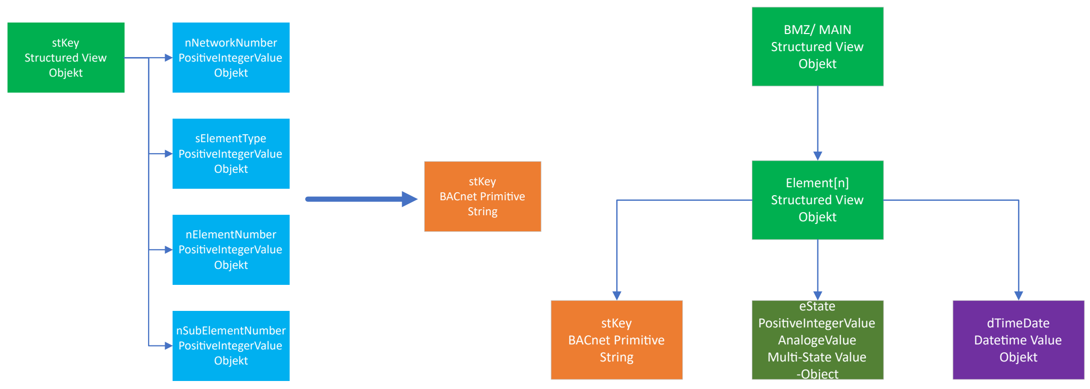

BMZ/ MAIN Structured View Objekt
N-Times
Element[n] Structured View Objekt
DateTime Datetime Value Objekt
nCO LargeAnalogValue Objekt
nTemperature LargeAnalogValue Objekt
nSmoke LargeAnalogValue Objekt
stKey Structured View Objekt
nNetworkNumber PositiveIntegerValue Objekt
sElementType PositiveIntegerValue Objekt
nElementNumber PositiveIntegerValue Objekt
nSubElementNumber PositiveIntegerValue Objekt
aState[1] Structured View Objekt
aState[2] Structured View Objekt
aState[3] Structured View Objekt
FunctionType PositiveIntegerValue Objekt
nSubType PositiveIntegerValue Objekt
eState PositiveIntegerValue Objekt
nSubState PositiveIntegerValue Objekt
nLevel PositiveIntegerValue Objekt
nInfo PositiveIntegerValue Objekt
dTimeDate Datetime Value Objekt

Figure 3.6: Large BACnet information model

3.3.2 Minimal BACnet Information Model

The space restrictions lead to a vastly reduced BACnet information model. The model has been reduced to a minimum by removing all optional objects that are not needed to report a fire alarm. Even the data points for CO, smoke and temperature had to be removed to gain room for more fire alarm system nodes. The downside of the new BACnet information model is, that information for Industry 4.0 use cases is not available due to space limitations. In case of only a few fire alarm system elements being needed, this model can be easily extended with temperature, CO and smoke values. Otherwise, a third party BACnet stack with more storage can be used to create proprietary BACnet objects or to use LSP and LSZ objects and extend them with analog sensor values.

The memory saving measures described above result in a model that requires only 4 objects per node of the fire alarm system, instead of 33. The structured view **stkey** with all its sub elements is summarized to a string and stored in a BACnet Primitive String. Figure 3.7a shows the translation. The timestamp stays unchanged from the previous model, and of the element-states, only the Positive-Integer-Value State from the structured view object State[1] is carried to the new model. The estate can be modeled via a Positive-Integer-Value, Analog-Value or Multi-State-Value object and all of them have their pros and cons. Figure 3.7b shows the new design.



(a) Stkey summarization (b) BACnet information model

Figure 3.7: Minimal BACnet information model

The primitive object has the advantage that it carries a minimum of unneeded properties, but the disadvantage is that it is not subscribable via BACnet and can't send alarm notifications. It can only be polled, which is not optimal for a big fire alarm system with multiple thousands of elements. When using the analog-value-object, the state can be subscribed and the value will be updated by the client automatically when changed. The downside is, that the object carries some unnecessary properties and the alarm notifications are based on a minimum and maximum threshold of the present value and is therefore not usable. The multi-state-value object also caries unneeded properties, but supports subscriptions and alarm notifications on a state basis. The only drawback is that the states start at 1 and the ISP-IP states starts at 0, therefore the states are offset by one which may cause confusion.

3.3.3 Mapping Implementation With TwinCAT

To translate the fire alarm system to BACnet/IP, the state machine from Section 3.1.3 needs to be extended with dynamic BACnet object creation based on the number of elements and a mapping function that updates the objects in case a value changes. To make the generation and update of the BACnet objects easier, a function block is created. This function-block can be initialized with an array to match the number of fire alarm system elements. The code section described by Algorithm 3.4 shows the function block for the minimal information model with the multi-value-state object for the fire alarm system state. When reading the elements from the FACP in the states **start**, **getnext** and **get_description**, the elements are created or updated.

Algorithm 3.4: Function block of the multi-value-state information model

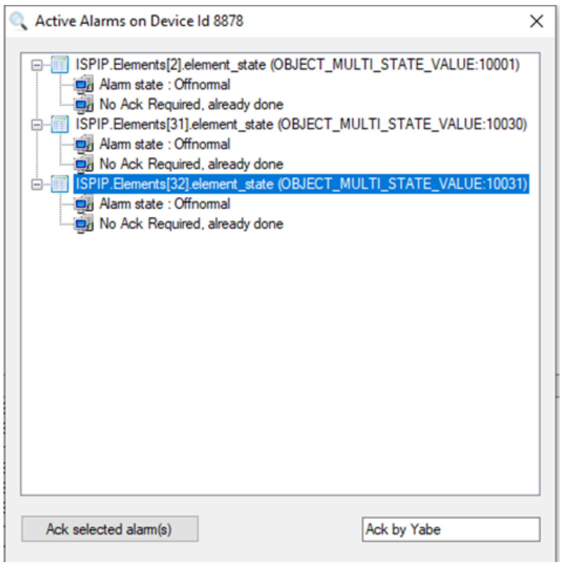
```
FUNCTION_BLOCK  FB_BACnet_BMZ_Element
VAR_INPUT
{attribute  'TcEncoding':='UTF-8'}
description_in  :  STRING := 'not  set' ;
st_Key_in  :  STRING := 'not  set' ;
dateTime_in  :  DATE_AND_TIME ;
state_in  :  BYTE := Tc3_IspIp . E_IspIpState . Fault ;
NotUsed  :  BOOL := TRUE ;
END_VAR
VAR_OUTPUT
END_VAR
VAR
    //Safe Values to check for changes
    {attribute  'TcEncoding':='UTF-8'}
    description_intern  :  STRING := 'not  set' ;
    dateTime_intern  :  DATE_AND_TIME ;
    state_intern  :  BYTE := Tc3_IspIp . E_IspIpState . Fault ;
    st_Key_intern  :  STRING := 'not  set' ;
    //BACnet-Objekte
    element  :  FB_BACnet_View := ( sDescription := 'Not  Used!' ) ;
    state  :  FB_BACnet_MV := ( iParent  :=  element ,
                                bEnPgm  :=  TRUE ,
                                eNotifyType  :=  E_BACnet_NotifyType . eAlarm ,
                                aEventEnable  :=  [ 1 , 1 , 1 ] ,
                                aAlarmValues := [ 2 , 3 ] ,
                                nValPgm := 1
                                ) ;
    dateTime  :  FB_BACnet_DateTime := ( iParent  :=  element ) ;
    stKey  :  FB_BACnet_String := ( iParent  :=  element ) ;
END_VAR

```

```
// If not Used don't create all Objects
IF ( NOT  NotUsed )  THEN
    element ( ) ;
    stKey ( ) ;
    state ( ) ;
    dateTime ( ) ;
    // Only update if changed
    IF ( NOT  ( description_in  =  description_intern ) )  THEN
        description_intern  :=  description_in ;
        element . sDescription := description_intern ;
    END_IF

    IF ( NOT  ( st_Key_in  =  st_Key_intern ) )  THEN
        st_Key_intern  :=  st_Key_in ;
        stKey . sValue := st_Key_intern ;
    END_IF

    IF ( NOT  ( state_in  =  state_intern ) )  THEN
        state_intern  :=  state_in ;
        state . nValPgm := state_intern + 1 ;  // Array starts with 1 not with 0
    END_IF
    IF ( NOT  ( dateTime_in  =  dateTime_intern ) )  THEN
        dateTime_intern  :=  dateTime_in ;
        dateTime . stValue := F_BA_ToSTDateTime ( dateTime_intern ) ;
```

3.4 Beckhoff BACnet Stack Results

3.4.1 BACnet Viewer

All three Information Models (positive-integer-value-, Analogue-Value- and multi-state-value-object) were tested to verify that the mapping is working and to evaluate the maximum number of fire alarm system elements. All models worked with, *Yet Another BACnet Explorer* (Yabe) [21], *Cimetrics Free BACnet Explorer* [22] and *Inneasoft Free BACnet Explorer* [23]. The alarm notification of the multi-state-value-object is also working, Figure 3.8 shows the active alarm summary in Yabe when a manual call point is pressed. The positive-integer-value and analog-value model can have a maximum of approximately, 6560 elements. The multi-state-value-object model can represent up to 5470 elements which is a bit less than the other two models but still much larger compared to the first information model.


Active Alarms on Device Id 8878

ISPIP.Elements[2].element_state (OBJECT_MULTI_STATE_VALUE:10001)
Alarm state : Offnormal
No Ack Required, already done
ISPIP.Elements[31].element_state (OBJECT_MULTI_STATE_VALUE:10030)
Alarm state : Offnormal
No Ack Required, already done
ISPIP.Elements[32].element_state (OBJECT_MULTI_STATE_VALUE:10031)
Alarm state : Offnormal
No Ack Required, already done

Figure 3.8: Active alarms multi-state-value-object model

3.4.2 Open Points and Weaknesses

The use cases for Industry 4.0 are very limited with this BACnet information model. No analog values are available for smart climate control or building automation. The analog values can easily be added, but would require three additional BACnet objects with approximately 20 KB each. This would increase the number of required objects per fire alarm system element from 4 to 7, that nearly doubles the needed memory per element and therefore reduces the number of supported fire alarm system elements drastically. Another possibility would be to serialize the three analog values into a string and include it in the BACnet information model. In this way, the additional memory required is less, but usability would suffer.

Some properties are still writable via a BACnet viewer, that problem can be resolved by adding constrains by the BACnet object generation. Writing to the BACnet server will have no effect on the fire alarm system and only affects the BACnet objects and is therefore irrelevant for the PoC.

The multi-value-state information model supports alarm and fault notifications based on the present_value property. There are two arrays, one for alarms and one for faults, that specify which present_values triggers a notification. The ISP-IP states that cause alarm and faults still need to be selected and added to the respective arrays. For the PoC, the BACnet Life Safety State **alarm**(2) has been added to the alarms array.

3.5 Life Safety Point and Life Safety Zone Information Model

Before creating the LSP and LSZ with the C# Library, an information model that defines the mapping from ISP-IP Elements to LSP and LSZ, is needed. This model also includes optional properties that can be used for intrinsic reporting, though reporting will not be implemented in PoC.

The mapping is done in two steps, first the ISP-IP Elements are mapped to Life Safety data points by TwinCAT. Secondly, the Life Safety data points are mapped to LSP and LSZ objects by C#. Tables 3.2 and 3.3 show the first step of the mapping. Most mappings are straight forward values of ISP-IP Elements, but a few need special mapping. Some properties are required by the BACnet standard but not needed for the mapping of the fire alarm system elements, therefore these values will be mapped to a static value.

LSP required properties		
Source	**Mapping**	**Output**
-	Static: None accepted	Accepted_Modes
-	Static: None supported	Mode
-	Static: Indicates event state when intrinsic reporting is used, otherwise “Normal”	Event_State
-	Static: “Normal”	Operation_Expected
eState, nSubState	FB_GET_Reliability see 3.5	Reliability
eState	eState==Disablement ⇒ TRUE eState==Disablementpublic ⇒ TRUE else ⇒ FALSE	Out_of_Service
eState	eState==Deactivation ⇒„all_silenced“ else ⇒”unsilenced”	Silenced
eElementTypeClass	eElementTypeClass	Device_Type
description_in	description_in	Description
-	Event_State==’Normal’ ⇒ FALSE else ⇒ TRUE	Status_Flag: IN_ALARM

eState	estate=FAULT & Reliability!=’General’ ⇒ TRUE
	else ⇒ FALSE
		Status_Flag: FAULT

-	Reliability==Simulation ⇒TRUE else ⇒ FALSE	Status_Flag: OVERRIDDEN
-	Out_of_Service ⇒ TRUE else ⇒ FALSE	Status_Flag: OUT_OF_SERVICE
eState	LSS_MAP(eState) see 3.4	Present_Value
-	Same as Present_Value	Tracking_Value
nSmoke	nSmoke	Direct_Reading
nElementNumber	nElementNumber	ElementNumber
nSubElementNumber	nSubElementNumber	subElementNumber

Table 3.2: Life Safety Point required properties

LSP properties that are only needed for intrinsic reporting		
Source	**Mapping**	**Output**
-	Depends on BACnet network	Notification_Class
-	List of Present_Values that should trigger Life Safety Alarms	Life_Safety_Alarm_Values
-	List of Present_Values that should trigger Alarm_Values	Alarm_Values
-	List of Present_Values that should trigger Fault_Values	Fault_Values
-	0	Time_Delay
-	(TRUE, TRUE, TRUE)	Event_Enable
-	Should be handled by the BACnet stack	Acked_Transitions
-	ALARM	Notify_Type
-	Should be handled by the BACnet stack	Event_Time_Stamps
-	TRUE	Event_Detection_Enable

Table 3.3: Life Safety Point intrinsic reporting properties

Reliability mapping is based on the state and substate of an ISP-IP Element, which follows no easy programmable logic. Therefore, the mapping is done with the help of a two-dimensional array. The first dimension is the state and the second dimension is the substate. The reliability mapping function block is shown in Algorithm 3.5,the array is only partially visible due to its size. According to the fire alarm system description, the state “Enabled” contains 255 substates, Level1 to Level255, in increasing order. Creating the mapping per hand would be impractically and unnecessary, Level14 to Level255 can be created by a loop at the startup of the PLC due to the simple naming scheme. Level1 to Level13 will be coded by hand like the other reliability values to simplify the loop.

Algorithm 3.5: Reliability mapping function block

```
FUNCTION_BLOCK Get_IsP_Ip_BACnet_Reliability
VAR_INPUT
eState : INT ;
nSubState : INT ;
END_VAR
VAR_OUTPUT
Reliability : STRING ;
END_VAR
VAR
     Reliability_values : ARRAY [ 0 .. 27 ] OF ARRAY [ 0 .. 255 ] OF STRING :=
[ [ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not
[  'General' ,  'Simulation' ,  'Smoke' , 'Temp' , 'Sabotage' , 'Check' , 'Unconfirmed' , 'CO' , 'SmokeTemp' , 'SmokeC
[ 'General' , 'Not_defined' , 'Smoke' , 'Temp' , 'Sabotage' , 'Not_defined' , 'Not_defined' , 'CO' , 'SmokeTemp' , 'Sm
[ 'General' , 'ShortCircuit' , 'OpenCircuit' , 'Simulation' , 'External' , 'Overload' , 'NotPresent' , 'EarthFault'
[ 'General' , 'ShortCircuit' , 'OpenCircuit' , 'Not_defined' , 'Not_defined' , 'Overload' , 'NotPresent' , 'EarthFau
[ 'General' , 'KeySwitch' , 'Automatic' , 'FBP' , 'Emergency' , 'Limited' , 'Not_defined' , 'Not_defined' , 'Not_defi
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Automatic' , 'External' , 'FBP' , 'Unconfirmed' , 'Confirmed General' , 'Confirmed Automatic' , 'Conf
[ 'General' , 'FBP' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined'
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Level1' , 'Level2' , 'Level3' , 'Level4' , 'Level5' , 'Level6' , 'Level7' , 'Level8' , 'Level9' , 'Level
[ 'General' , 'Simulation' , 'Automatic' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defi
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'External' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defi
[ 'General' , 'Automatic' , 'External' , 'FBP' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Simulation' , 'Smoke' , 'Temp' , 'Sabotage' , 'Not_defined' , 'Not_defined' , 'CO' , 'SmokeTemp' , 'Sm
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'External' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defi
[ 'General' , 'Simulation' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_de
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'Not_
[ 'General' , 'Not_defined' , 'Smoke' , 'Temp' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'CO' , 'SmokeTemp' ,
[ 'General' , 'Not_defined' , 'Smoke' , 'Temp' , 'Not_defined' , 'Not_defined' , 'Not_defined' , 'CO' , 'SmokeTemp' ,
[ 'General' , 'Simulation' , 'Smoke' , 'Temp' , 'Sabotage' , 'Not_defined' , 'Not_defined' , 'CO' , 'SmokeTemp' , 'Sm
] ;      // Not_defined needs to be added to 255


count : INT ;
subcount : INT ;
init : BOOL := FALSE ;
END_VAR

```

```
IF ( NOT init ) THEN

    FOR count := 0 TO 27 BY 1 DO
            FOR subcount := 14 TO 255 BY 1 DO
                IF ( count = E_IspIpState . Enabled ) THEN
                    Reliability_values [ count ] [ subcount ] := CONCAT ( 'Level' , TO_STRING ( subcount ) ) ;
                ELSE
                    Reliability_values [ count ] [ subcount ] := 'Not_defined' ;
                END_IF
            END_FOR
            END_FOR
    init := TRUE ;
END_IF

Reliability := Reliability_values [ eState ] [ nSubState ] ;

```

The Present Value is represented by the APDU **BACnetLifeSafetyState**. Values 0-23 are defined and 24-255 are reserved by the American Society of Heating, Refrigerating and Air-Conditioning Engineers(ASHRAE) [6]. 256-65635 can be used to add proprietary states. The states of the ISP IP Elements were mapped to the existing values where possible, otherwise proprietary states were added. Table 3.4 shows the mapping of ISP-IP Element states to the **BACnetLifeSafetyState**. Dark green entries have been mapped to existing values; light green values have not been used. Light blue values are mapped to proprietary **BACnetLifeSafetyState** values.

BACnet Life Safety State	Nummer	ISP-IP-State	ISP-IP-Number
quiet	0	Idle	0
pre-alarm	1	PreAlarm	27
alarm	2	Alarm	1
fault	3	Fault	3
fault-pre-alarm	4		
fault-alarm	5		
not-ready	6		
active	7	Active	1
tamper	8		
test-alarm	9	RevisionAlarm	2
test-active	10		
test-fault	11	RevisionFault	4
test-fault-alarm	12		
holdup	13		
duress	14		
tamper-alarm	15		
abnormal	16		
emergency-power	17		
delayed	18		
blocked	19		
local-alarm	20		
general-alarm	21		
supervisory	22		
test-supervisory	23		
Custom-Disablement	256	Disablement	5
Custom-Disablement public	257	Disablement public	6
Custom-Revision	258	Revision	7
Custom-Activation	259	Activation	8
Custom-Deactivation	260	Deactivation	9
Custom-HardAlarm	261	HardAlarm	10
Custom-Enabled	262	Enabled	11
Custom-RevisionActive	263	RevisionActive	13

Table 3.4 continued from previous page

BACnet Life Safety State	Nummer	ISP-IP-State	ISP-IP-Number
Custom-Warning	264	Warning	14
Custom-Explore	265	Explore	15
Custom-Terminated	266	Terminated	16
Custom-RevisionActivation	267	RevisionActivation	17
Custom-PaperFeed	268	PaperFeed	18
Custom-SilentAlarm	269	SilentAlarm	19
Custom-PreActivation	270	PreActivation	20
Custom-Release	271	Release	21
Custom-PreActive	272	PreActive	22
Custom-RevisionPreActive	273	RevisionPreActive	23
Custom-LowBat	274	LowBat	24
Custom-Presignal	275	Presignal	25
Custom-Revision Presignal	276	RevisionPresignal	26

Table 3.4: Bacnet Life Safety State mapping

The Life Safety data points are mapped to LSZ and LSP objects by C#. The LSZs contain lists of the LSPs that are part of the LSZ. The data points are grouped based on *Device_Type*, *ElementNumber*, and *subElementNumber*. Figure 3.9 shows the grouping logic. First, the *Device_Type* is used to verify that the data point is of the type *zone*. If not, the element must be an LSP. However, the *Device_Type* is not a sufficient condition for LSZ, therefore the *subElementNumber* must also be equal to 255. Otherwise, the data point is an LSP as well, if the LSP is Part of a Zone, it is added to the corresponding LSZ *Member_List*. The *ElementNumber* is used to generate the *object_name* property.

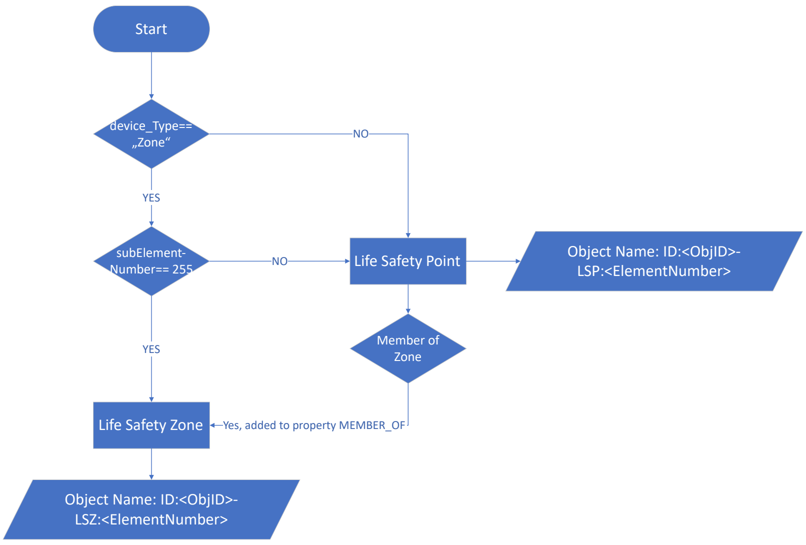


Figure 3.9: LSP and LSZ grouping

3.6 Data Transfer TwinCAT to C#

The communication protocol Automation Device Specification (ADS) enables the data exchange from the PLC to C# and vice versa. The advantage of ADS is that the C# program can be executed on the PLC itself or on another computer that is connected to the PLC via Ethernet. Furthermore, PLC variables are accessible via ADS without modifications to the PLC-program. [24]

3.6.1 TwinCAT First Version

Figure 3.10 shows the first version of the PLC-state machine that groups the ISP-IP Elements into arrays which can be accessed via ADS. The state machine is based on the Beckhoff BACnet implementation and is altered to create LSP and LSZ data points instead of Beckhoff BACnet objects. Three flags are used to ensure safe reads between the PLC and C# program. *Csc_ready* is set to true by the PLC when the C# program can safely read from the PLC. *Value_change* is set to true by the PLC when the list of LSP and LSZ data points have changed, and is set to false by the C# program when the data points have been read. *Reading* is set to true by the C# program when it starts reading the LSP and LSZ data point arrays, and is set to false when finished. Initially, *csc_ready* is set to **false** and *value_change* is set to **true**. When the state machine states *start* or *GetNext* are reached, the function block *update* is called. The function block creates and updates the LSP and LSZ data points following the first mapping step 3.2 without intrinsic reporting. The LSP and LSZ data points are function blocks, one function block is created per LSP and LSZ. In case of value changes, the flag *value_change* is set to **true**. The initial interaction of state machine will also fetch the element descriptions and add them to the LSP and LSZ data points. After that, LSP and LSZ data points are converted to arrays by the PLC if the C# program is not currently reading the arrays. Arrays are used to transfer the data points to C#, the reason for using arrays is that the ADS-library from Beckhoff provides no easy way to transfer arrays of function blocks to C#. During the update of the data point arrays, the flag *csc_ready* is set to **false** to prevent inconsistent data from being read. The state machine repeats this procedure if the timer calls for an update.

The drawback of this approach is the limited amount of LSP and LSZ data points due to space limitations of the PLC. A maximum of 1842 LSP and LSZ data points can be created. The problem is the largest continuous memory block of 982.250.463 bytes (reported by the compiler). Therefore, the array of data points approach is limited by the available continuous memory. For large fire alarm systems, this state machine does not scale well, and a different approach is required. A solution to this problem is presented in Section 3.6.2.

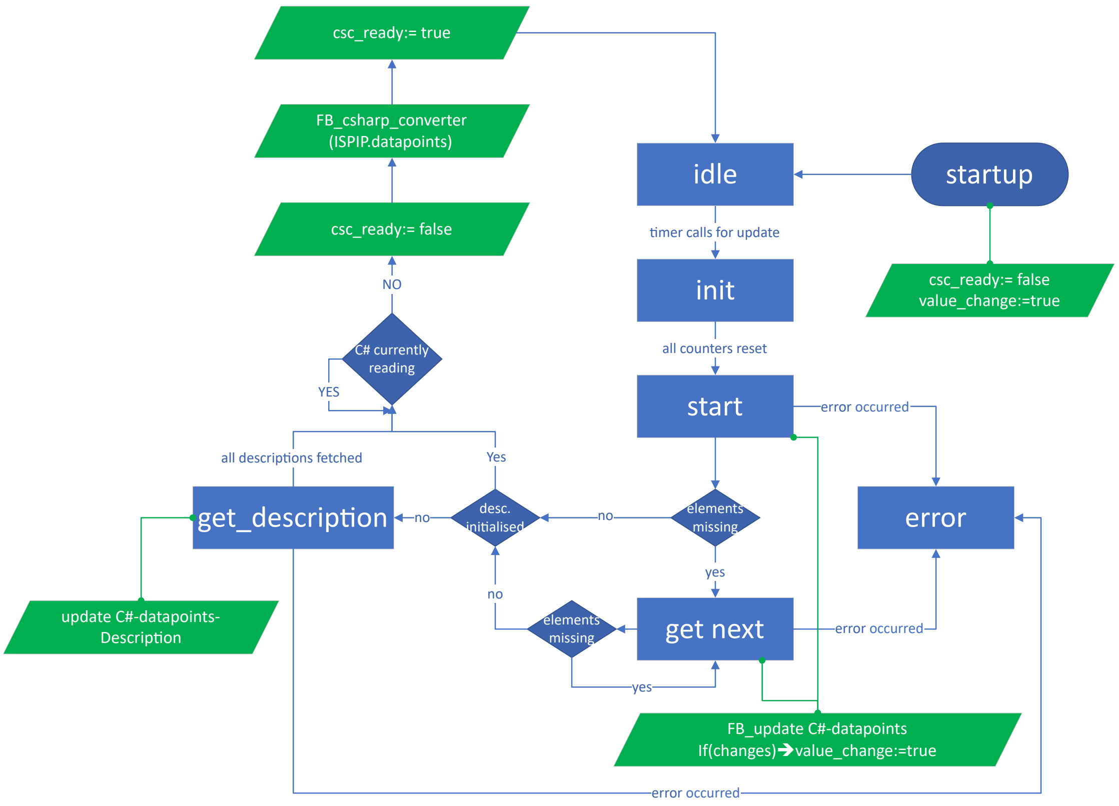


Figure 3.10: ISP-IP to C# first version state machine

3.6.2 TwinCAT Second Version

To allow for larger number of LSP and LSZ data points, the function block for the data points needs to be broken up into smaller pieces. Data point arrays need to be divided into individual arrays per property to be transmitted to the C# program. Therefore, the function block array, whose main purpose is structuring the data, is removed completely and the LSP and LSZ data points are stored in separate arrays. The new state machine is shown in Figure 3.11. The changes increase the maximum amount of LSP and LSZ data points significantly from 1842 to over 2800000. One must keep in mind, that the larger the number of ISP-IP Elements, the more PLC cycles are needed to update the LSP and LSZ data points. Especially the first iteration which fetches the descriptions of LSP and LSZ will take some time.

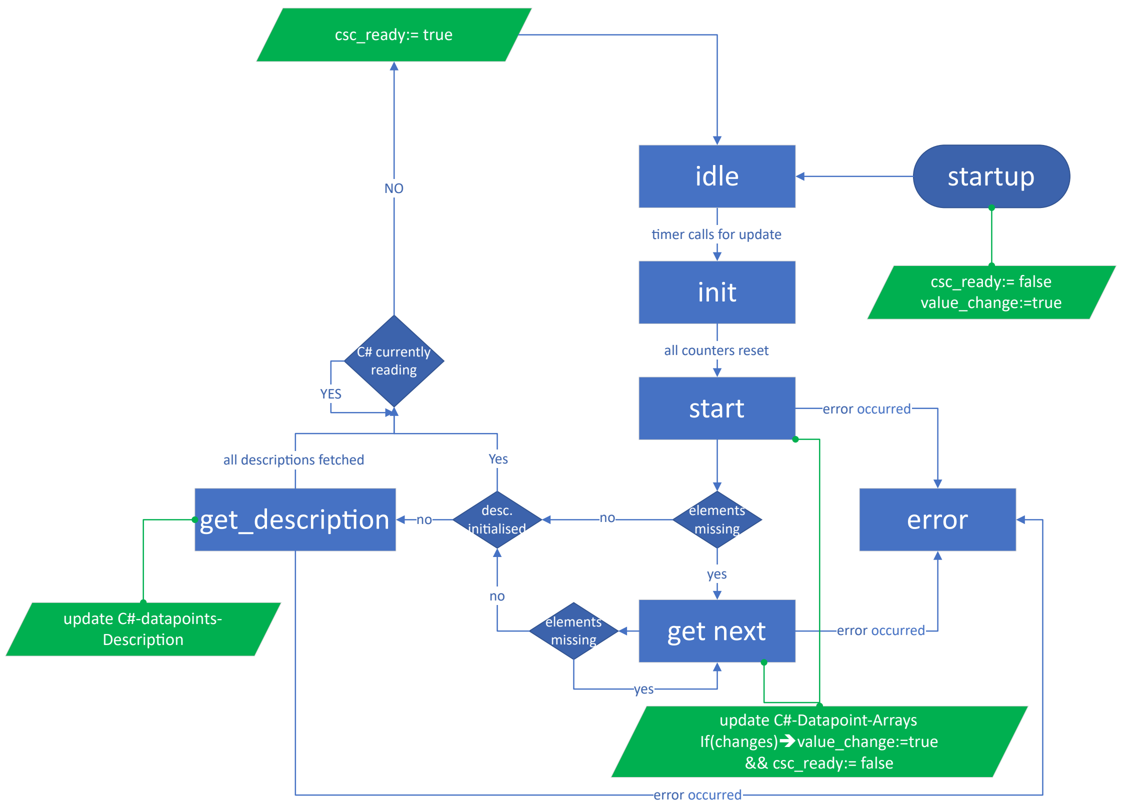


Figure 3.11: ISP-IP to C# second version state machine

3.6.3 C# Access and Write Data

The C# program connects to TwinCAT via ADS at startup and can then access the PLC variables.

A class for data exchange is created to structure the communication between TwinCAT and C#. To establish the connection from C# to the PLC, only a *port* and *netID* are needed. When using TwinCAT3, the port 851 is used and the *netID* is the **AMSNetId** of the PLC.

When the connection is established, the variables can be accessed in multiple ways. Variables can be accessed via an address or the variable name, for better readability variable names are used. The reading of values can be done event driven or via a read function. This PoC will poll the read function and use flags to mark value changes. The reason for manual CoV tracking is to gain full control of the update process to prevent changes on C# BACnet objects while the PLC is still processing ISP-IP Elements. Reading all ISP-IP Elements from the fire alarm system can take multiple PLC cycles, the C# program should only read the element list when all elements are updated, therefore manual flags are used to indicate if the ISP-IP Elements are ready to be read.

The data exchange class contains functions to read single *bool*, *int16* and *String* variables as well as arrays of *bool*, *int16*, *String*, *unit16* and *int32* data types. One write-function for *bool* variables is available as well to set the data exchange flags. To simplify the read access and grouping of LSP and LSZ data points into template classes, a function is available as well. The LSZ and LSP are stored into two lists, one containing LSZs with their LSP-members and a second list containing LSPs without an LSZ. These LSZ and LSP template classes are independent of the used BACnet stack and allow easy switching of BACnet stacks in the future.

3.7 Life Safety Objects with C# BACnet Library

Due to the lack of LSP and LSZ support of Beckhoff PLC, another solution is needed. One is to use an external C# BACnet stack to generate the LSP and LSZ objects and access the data from the PLC via ADS protocol from Beckhoff.

3.7.1 C# Library

The ISP-IP Elements are mapped to LSP and LSZ template classes within C# and must be turned into real BACnet objects with help of a BACnet stack. Most of the available BACnet stacks for C# are only available for purchase by request, and are therefore not suited for this PoC implementation. Hence, the freely available C# BACnet stack maintained at GitHub [25] by ela-compil and downloadable at NuGet [26] is used. This library supports the generation of BACnet objects at runtime, but needs a C# template-class per BACnet object. The class must define all object properties, the encoding of the properties and intrinsic reporting. Some examples for C# BACnet classes are given [27], but most of them are over six years old and are thus not fully compatible with the newest version of the BACnet stack. Moreover, LSP and LSZ are not utilized in the examples at all. Therefore, LSP and LSZ C# template classes must be implemented based on the DIN EN ISO 16484-5:2017-12 [6]. For the proof of concept, the intrinsic reporting part of the LSP and LSZ will not be implemented with C# BACnet stack, only the required properties and some optional properties along with CoV notification will be realized. Before creating the LSP and LSZ template classes, some missing APDUs need to be defined and proprietary values must be added. After the APDUs *BACnetReliability*, *BACnetLifeSafetyMode*, *BACnetLifeSafetyStates*, *BACnetSilencedState* and *BACnetLifeSafetyOperation* are defined the templates can be created. The class templates use the *BACSharpObject* [28] class from GitHub as the parent class, which defines fundamental functions and properties. When defining a property, the encoding type and CoV management must be defined. An example for the property *PRESENT_VALUE* of the LSP and property *Zone_Members* of the LSZ is shown below.

Algorithm 3.6: C# BACnet property definition

```
public BacnetLifeSafetyStates m_PROP_PRESENT_VALUE;
[BaCSharpType(BacnetApplicationTags.BACNET_APPLICATION_TAG_ENUMERATED)]
public virtual BacnetLifeSafetyStates PROP_PRESENT_VALUE
{
    get { return m_PROP_PRESENT_VALUE; }
    set
    {
        m_PROP_PRESENT_VALUE = value;
        ExternalCOVManagement(BacnetPropertyIds.PROP_PRESENT_VALUE);
    }
}
protected List<BacnetValue> m_PROP_ZONE_MEMBERS = new List<BacnetValue>();
[BaCSharpType(BacnetApplicationTags.BACNET_APPLICATION_TAG_OBJECT_ID)]
public virtual List<BacnetValue> PROP_ZONE_MEMBERS
{
    get { return m_PROP_ZONE_MEMBERS; }
}

```

The constructor of the LSP and LSZ templates takes an LSP and LSZ data point that has been created by the TwinCAT-read function described in section 3.6.3 and turns it into a BACnet object. Additionally, an update function is created to update the BACnet object in case of changes.

Algorithm 3.7: C# LSP constructor

```
public LifeSafetyZone(uint objid, TW_LifeSafetyZone lsp)
            : base(new BacnetObjectId(BacnetObjectTypes.
    OBJECT_LIFE_SAFETY_ZONE, objid), ("ID:" + objid.ToString() + "-LSZ-" +
    lsp.ElementNumber.ToString()), lsp.Description)
        {
            m_PROP_STATUS_FLAGS.SetBit((byte)0, lsp.Status_Flags_IN_ALARM);
            m_PROP_STATUS_FLAGS.SetBit((byte)1, lsp.Status_Flags_FAULT);
            m_PROP_STATUS_FLAGS.SetBit((byte)2, lsp.Status_Flags_OVERRIDDEN);
            m_PROP_STATUS_FLAGS.SetBit((byte)3, lsp.
    Status_Flags_OUT_OF_SERVICE);
            PROP_PRESENT_VALUE = (BacnetLifeSafetyStates)lsp.Present_Value;
            m_PROP_TRACKING_VALUE = m_PROP_PRESENT_VALUE;
            m_PROP_DEVICE_TYPE = lsp.Device_Type;
            m_PROP_EVENT_STATE = 0;
            Enum.TryParse<BACnetReliability>(lsp.Reliability, out
    m_PROP_RELIABILITY);
            PROP_OUT_OF_SERVICE = lsp.Out_of_Service;
            Enum.TryParse<BACnetSilencedState>(lsp.Silenced, out
    m_PROP_SILENCED);
            m_PROP_DIRECT_READING = lsp.Direct_Reading;
        }


```

To create and update all LSP and LSZ, a state machine is created in C#. Figure 3.12 illustrates the broad functionality of the state machine. At the startup, a connection to the PLC is established. Afterwards, if the *csc_ready* is set to **true**, the LSP and LSZ objects are read and grouped. If the flag *csc_ready* remains **false** for multiple seconds, the program enters an error state. After the objects have been successfully loaded, the BACnet objects are created. From there on, the program queries the value_change flag at intervals of 500ms. In case the flag is set to **true**, the flag will be set to **false** and the LSP and LSZ data points will be read from the PLC and BACnet objects will be updated. During read operations from the PLC, the program will set the *reading* flag to **true** to prevent inconsistent data reads.

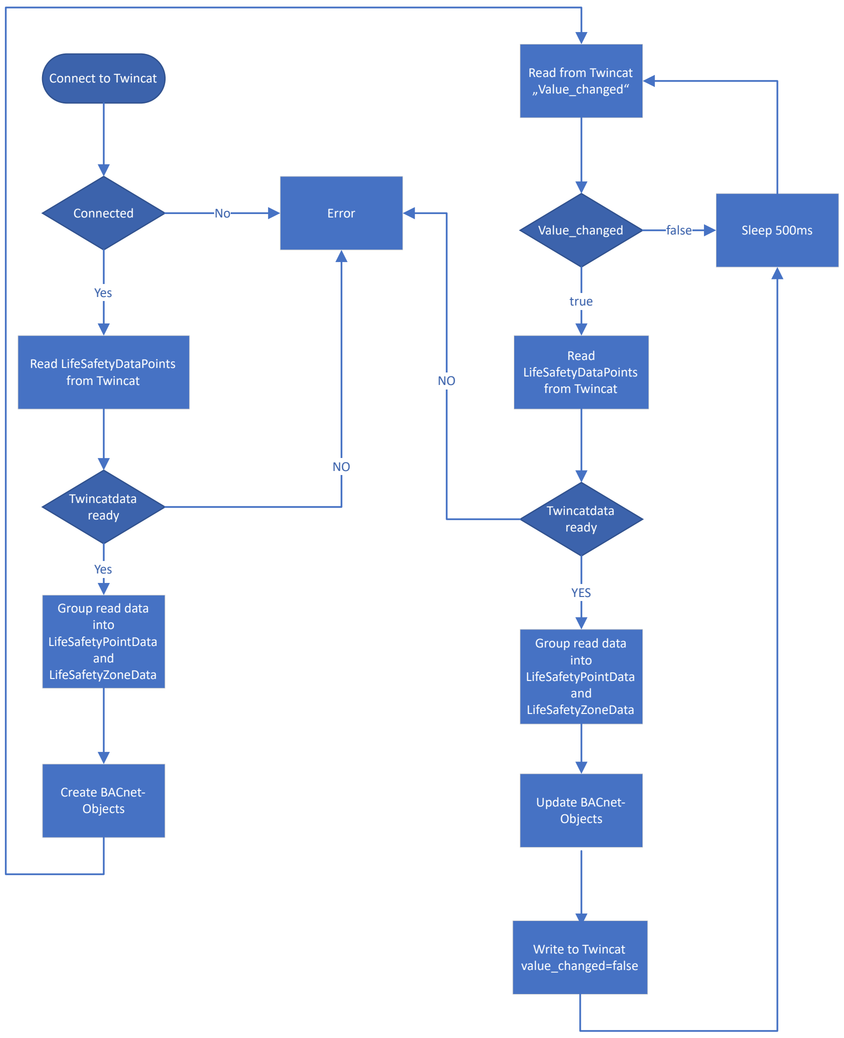
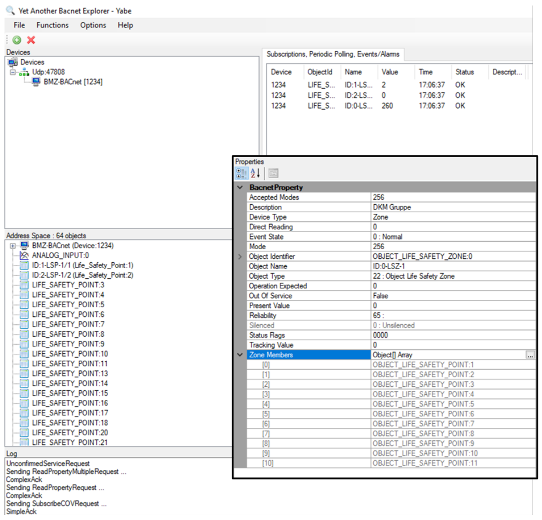

Connect to Twincat
Connected
No
Error
Yes
Read LifeSafetyDataPoints from Twincat
Twincatdata ready
NO
Yes
Group read data into LifeSafetyPointData and LifeSafetyZoneData
Create BACnet-Objects
Read from Twincat „Value_changed“
Value_changed
false
Sleep 500ms
true
Read LifeSafetyDataPoints from Twincat
NO
Twincatdata ready
YES
Group read data into LifeSafetyPointData and LifeSafetyZoneData
Update BACnet-Objects
Write to Twincat value_changed=false

Figure 3.12: C# BACnet object generation

3.8 C# BACnet Server Results

3.8.1 LSP and LSZ Read Access via BACnet Viewer

The same three BACnet viewers as before are used to test the BACnet objects. Yabe and the Inneasoft Free BACnet Explorer work fine, all properties are correctly encoded, readable and CoV notifications work for the PoC fire alarm system. The free viewer from Cimetrics has problems with the *Member_List* property of the LSZ. Figure 3.13 shows the three BACnet Viewer side by side.


Yet Another Bacnet Explorer - Yabe
File
Functions
Options
Help
Devices
Devices
Udp:47808
BMZ-BACnet [1234]
Subscriptions, Periodic Polling, Events/Alarms
Address Space : 64 objects
Properties
Log

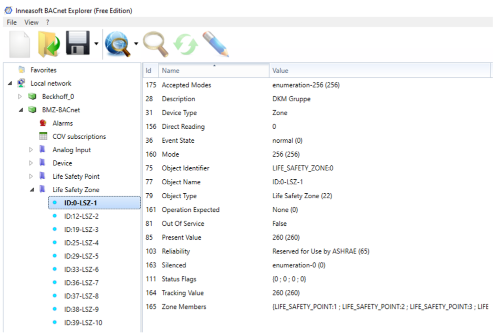
Device	ObjectId	Name	Value	Time	Status	Descript...
1234	LIFE_S...	ID:1-LS...	2	17:06:37	OK	
1234	LIFE_S...	ID:2-LS...	0	17:06:37	OK	
1234	LIFE_S...	ID:0-LS...	260	17:06:37	OK	

BMZ-BACnet (Device:1234)
ANALOG_INPUT:0
ID:1-LSP-1/1 (Life_Safety_Point:1)
ID:2-LSP-1/2 (Life_Safety_Point:2)
LIFE_SAFETY_POINT:3
LIFE_SAFETY_POINT:4
LIFE_SAFETY_POINT:5
LIFE_SAFETY_POINT:6
LIFE_SAFETY_POINT:7
LIFE_SAFETY_POINT:8
LIFE_SAFETY_POINT:9
LIFE_SAFETY_POINT:10
LIFE_SAFETY_POINT:11
LIFE_SAFETY_POINT:13
LIFE_SAFETY_POINT:14
LIFE_SAFETY_POINT:15
LIFE_SAFETY_POINT:16
LIFE_SAFETY_POINT:17
LIFE_SAFETY_POINT:18
LIFE_SAFETY_POINT:20
LIFE_SAFETY_POINT:21

BacnetProperty	
Accepted Modes	256
Description	DKM Gruppe
Device Type	Zone
Direct Reading	0
Event State	0 : Normal
Mode	256
Object Identifier	OBJECT_LIFE_SAFETY_ZONE:0
Object Name	ID:0-LSZ-1
Object Type	22 : Object Life Safety Zone
Operation Expected	0
Out Of Service	False
Present Value	0
Reliability	65 :
Silenced	0 : Unsilenced
Status Flags	0000
Tracking Value	0
Zone Members	Object[] Array
[0]	OBJECT_LIFE_SAFETY_POINT:1
[1]	OBJECT_LIFE_SAFETY_POINT:2
[2]	OBJECT_LIFE_SAFETY_POINT:3
[3]	OBJECT_LIFE_SAFETY_POINT:4
[4]	OBJECT_LIFE_SAFETY_POINT:5
[5]	OBJECT_LIFE_SAFETY_POINT:6
[6]	OBJECT_LIFE_SAFETY_POINT:7
[7]	OBJECT_LIFE_SAFETY_POINT:8
[8]	OBJECT_LIFE_SAFETY_POINT:9
[9]	OBJECT_LIFE_SAFETY_POINT:10
[10]	OBJECT_LIFE_SAFETY_POINT:11

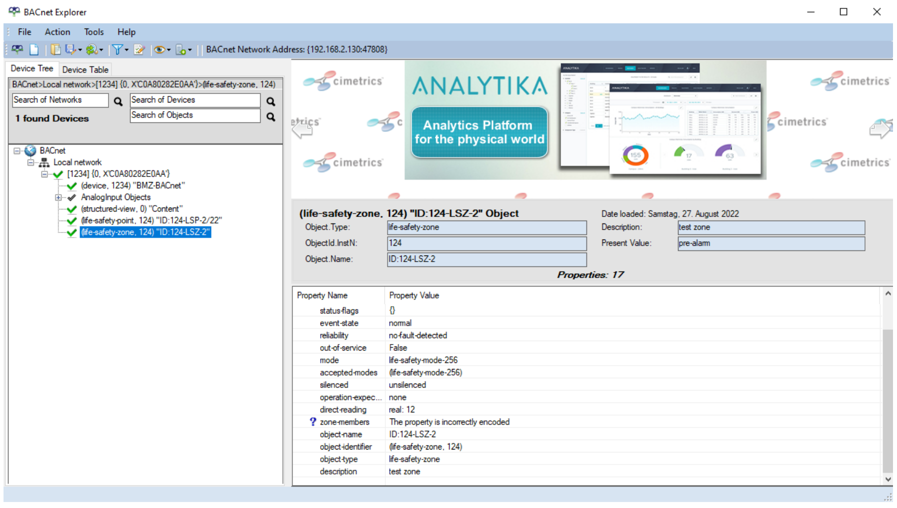
```
UnconfirmedServiceRequest
Sending ReadPropertyMultipleRequest ...
ComplexAck
Sending ReadPropertyRequest ...
ComplexAck
Sending SubscribeCOVRequest ...
SimpleAck
```

(a) BACnet viewer Yabe



Id	Name	Value
175	Accepted Modes	enumeration-256 (256)
28	Description	DKM Gruppe
31	Device Type	Zone
156	Direct Reading	0
36	Event State	normal (0)
160	Mode	256 (256)
75	Object Identifier	LIFE_SAFETY_ZONE:0
77	Object Name	ID:0-LSZ-1
79	Object Type	Life Safety Zone (22)
161	Operation Expected	None (0)
81	Out Of Service	False
85	Present Value	260 (260)
103	Reliability	Reserved for Use by ASHRAE (65)
163	Silenced	enumeration-0 (0)
111	Status Flags	{0 ; 0 ; 0 ; 0}
164	Tracking Value	260 (260)
165	Zone Members	{LIFE_SAFETY_POINT:1 ; LIFE_SAFETY_POINT:2 ; LIFE_SAFETY_POINT:3 ; LIFE

(b) BACnet viewer Inneasoft



BACnet Network Address: {192.168.2.130:47808}

BACnet>Local network>[1234] {0, X'C0A80282E0AA'}>(life-safety-zone, 124)

1 found Devices

BACnet
Local network
[1234] {0, X'C0A80282E0AA'}
(device, 1234) "BMZ-BACnet"
AnalogInput Objects
(structured-view, 0) "Content"
(life-safety-point, 124) "ID:124-LSP-2/22"
(life-safety-zone, 124) "ID:124-LSZ-2"

(life-safety-zone, 124) "ID:124-LSZ-2" Object

Date loaded: Samstag, 27. August 2022

Object.Type: life-safety-zone

ObjectId.InstN: 124

Object.Name: ID:124-LSZ-2

Description: test zone

Present Value: pre-alarm

Properties: 17

Property Name	Property Value
status-flags	{}
event-state	normal
reliability	no-fault-detected
out-of-service	False
mode	life-safety-mode-256
accepted-modes	(life-safety-mode-256)
silenced	unsilenced
operation-expec...	none
direct-reading	real: 12
? zone-members	The property is incorrectly encoded
object-name	ID:124-LSZ-2
object-identifier	(life-safety-zone, 124)
object-type	life-safety-zone
description	test zone

(c) BACnet viewer Cimetrics

Figure 3.13: BACnet viewer comparison

3.8.2 C# Program Hardware Requirements

The C# program does not consume many resources when idle. Only when new BACnet viewers connect, thousands of LSP and LSZ objects are read or a CoV event occurs, CPU consumption peaks may occur.Memory consumption is low at around 30 MB for 2000 BACnet objects. Figure 3.14 shows the memory and CPU consumption for 2000 objects. The CPU spikes occurs when a manual call point is pressed and BACnet objects must be updated and the CoV notifications are sent.

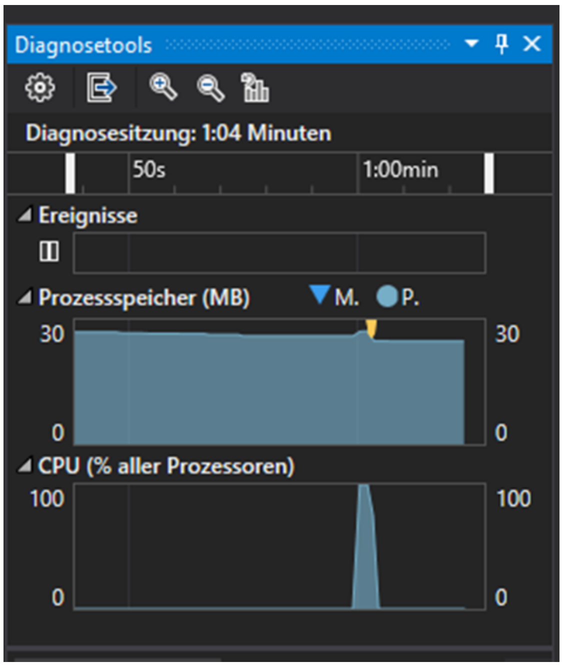


Figure 3.14: C# program hardware requirements

3.8.3 Open Points of the C# BACnet Server

The C# BACnet stack approach is not perfect, for a large number of BACnet objects the CoV notifications stop working reliably when lots of BACnet objects get updated at once, some notifications won't arrive at all or are delayed. However, manually refreshing the BACnet objects always works.

One time, the whole C# BACnet server went offline. After adding additional synchronization primitives, the problem could not be reproduced. This problem still needs to be further investigated.

Moreover, intrinsic reporting is not supported out of the box by the library for LSP and LSZ and is not implemented yet, therefore it must still be added.

The Cimetrics Free BACnet Explorer [22] has problems with the encoding of the LSZ property *Member_List*, the other two BACnet viewers had no problems. This issue still continues to exist and could not be solved, and also exists with the structured view example available at GitHub [29]. It could not be identified whether the viewer, BACnet server or a combination of both is the cause of the problem.

CHAPTER 4

Conclusion and Further Work

4.1 Beckhoff vs C# BACnet Stack

Both the Beckhoff and C# stack can be used to create a BACnet information model, although the minimal Beckhoff model is very limited. Only limited Industry 4.0 applications are possible due to the lack of raw sensor values. The C# Model at least contains one sensor value that can be used for advanced use cases. The C# Library should support the addition of proprietary properties, but to do so proprietary property-IDs must be added to the enum *BacnetPropertyIds*.

Large fire alarm systems with over 7000 fire alarm elements can not be represented with the Beckhoff BACnet stack due to space limitations (max 6500), therefore the C# stack (theoretically PLC limit of 2800000, C# stack tested up to 200000) is better suited for larger implementations. When looking at interoperability, the C# BACnet server with LSP and LSZ is a much better option. LSP and LSZ are standardized and therefore most likely supported by other BACnet devices. The information model of the Beckhoff BACnet stack is less compatible with other system due to grouping of standardized objects to represent one fire alarm system element instead of using LSP and LSZ.

The advantage of the Beckhoff stack is the BACnet conformance certificate [30] which could not be found for the open source C# BACnet stack. Especially for life safety applications like fire alarm system, certificates are necessary to deploy the BACnet stack in real world applications. Another downside of the C# BACnet option is, that two applications need to be maintained, the PLC program and C# program. This also increases the risk of security vulnerabilities and complicates the deployment of the system.

4.2 Conclusion

This thesis started with an analysis of available BACnet protocol-gateways, revealing that translations can be achieved in multiple ways, running on an MCU or PC and using different mapping strategies. These steps were followed by an analysis of the usability of a PLC to act as a Multiprotocol-Gateway for Fire Alarm Systems. The gathered information was used to build two BACnet information models, one for the Beckhoff PLC BACnet stack using grouped standardized BACnet objects and one for an open-source C# BACnet stack utilizing LSP and LSZ. Both running on a UcIPC with a software PLC to function as a BACnet Multiprotocol-Gateway. The PoC implementations show, that both gateways are viable solutions, although the C# model supports a larger number of BACnet objects and has a higher interoperability due to LSP and LSZ support. Unfortunately, the C# stack is not running as stable as the Beckhoff BACnet server and CoV notifications sometimes stop working properly when a high number of elements gets updated at once. The Beckhoff BACnet server has the advantage of having a BACnet conformance certificate and no need to operate an additional C# program. However, the lack of support for LSP and LSZ, the limitation to approx. 5000-6000 elements, and the lack of access to raw sensor values mean only limited suitability for Industry 4.0 use cases.

4.3 Further work

Both models can be further improved. The Beckhoff model can be expanded with string encoded analog values that are added to a description of an already used BACnet object. That way, memory usage can be kept at a minimum while increasing the usability for Industry 4.0. Nevertheless, the decoding of the analog value string would be more complicated for other BACnet devices reading the string compared to accessing an analog value property.

The used C# BACnet library may be sourced from GitHub instead of NuGet and extended with proprietary property-IDs to expand the LSP and LSZ with raw sensor readings. Moreover, intrinsic reporting could be added to the LSP and LSZ.

Additionally, proprietary C# BACnet stacks may be used to create the BACnet objects. For instance, the proprietary BACnet stack Chipkin [31] or Cimetrics [32] can be evaluated. These have the advantage of support and better documentation. Even different programming languages compatible with TwinCAT ADS may be tested to host the BACnet server.

List of Figures

2.1 Zettlerfire UC-8112-LX BACnet converter workflow [10] 7
2.2 NOTIFIER BACnet gateway system architecture [12] 8
2.3 ZigBee BACnet gateway mapping table [13] 9
2.4 BACnet gateway hardware structure and architecture [13] 9
2.5 BMS client and gateway system architecture [14] 10
2.6 EnOcean to BACnet gateway hardware architecture [15] 11
2.7 EnOcean report-telegram to BACnet [15] 12

3.1 System Architecture 13
3.2 UcIPC Beckhoff C6030 [16] 14
3.3 State machine 16
3.4 Supported object types 17
3.5 Beckhoff BACnet/IP-Server 19
3.6 Large BACnet information model 23
3.7 Minimal BACnet information model 24
3.8 Active alarms multi-state-value-object model 27
3.9 LSP and LSZ grouping 33
3.10 ISP-IP to C# first version state machine 35
3.11 ISP-IP to C# second version state machine 36
3.12 C# BACnet object generation 39
3.13 BACnet viewer comparison 40
3.14 C# program hardware requirements 41

List of Tables

2.1 Life Safety Point properties explanation [6] 5

3.1 ISP-IP Datatype to Beckhoff BACnet object mapping 18
3.2 Life Safety Point required properties 29
3.3 Life Safety Point intrinsic reporting properties 30
3.4 Bacnet Life Safety State mapping 33

List of Algorithms

3.1 Automapping with revision 12 [19] 19

3.2 Revision 14 function block declaration 20

3.3 Revision 14 function block updates 20

3.4 Function block of the multi-value-state information model 26

3.5 Reliability mapping function block 31

3.6 C# BACnet property definition 38

3.7 C# LSP constructor 38

Bibliography

[1] Z. Jiang, Y. Chang, and X. Liu, "Design of software-defined gateway for industrial interconnection," *Journal of Industrial Information Integration*, vol. 18, 6 2020.

[2] B. Dong, V. Prakash, F. Feng, and Z. O'Neill, "A review of smart building sensing system for better indoor environment control," pp. 29–46, 9 2019.

[3] Hekatron Vertriebs GmbH, "Leistungserklärung Brandmelder Nr.305/2011."

[4] K. Jakubowski, J. Paś, S. Duer, and J. Bugaj, "Operational analysis of fire alarm systems with a focused, dispersed and mixed structure in critical infrastructure buildings," *Energies*, vol. 14, no. 23, 12 2021.

[5] Beckhoff Automation GmbH & Co. KG., "Fire Alarm Systems | Schrack Seconet AG." [Online]. Available: https://www.schrack-seconet.com/firealarm/

[6] DIN Deutsches Institut für Normung e.V., "DIN EN ISO 16484-5." [Online]. Available: www.din.de

[7] Thomas Hansemann and Christof Hübner, *Gebäudeautomation.* Hanser, 2021.

[8] Beckhoff Automation GmbH & Co. KG., "Beckhoff BACnet IP Protocol Implementation Conformance Statement," 2021. [Online]. Available: https://www.bacnetinternational.net/catalog/manu/beckhoff%20automation%20gmbh/Beckhoff_BACnetIP_PICSen_Rev14_Ver4.0.pdf

[9] Johnson Controls Zettler, "Zettler MXZ BACnet Interface MZX BACnet Converter," 2017. [Online]. Available: https://www.zettlerfire.com/Products/Fire/ZettlerBACNet.asp

[10] ——, "Zettler MXZ BACnet Interface," 2017. [Online]. Available: https://www.tycoemea.com/pdf13/datasht/fire/Zettler/PSF247ZT.pdf

[11] Notifier by Honeywell, "Notifier BACnet Gateway Installation and Operation Manual," 2009. [Online]. Available: https://cgproducts.johnsoncontrols.com/MET_PDF/53372.pdf

[12] ——, “Notifier Network Systems.” [Online]. Available: https://www.securityandfire.honeywell.com/notifier/en-us/-/media/Files/Notifier/Data-Sheets/DN_6877_pdf.pdf

[13] S. C. Park, W. S. Lee, S. H. Kim, S. H. Hong, and P. Palensky, “Implementation of a BACnet-ZigBee gateway,” in *IEEE International Conference on Industrial Informatics*, 2010, pp. 40–45.

[14] C. V. Bharadwaj, M. Velammal, and M. Raju, “A BMS Client and Gateway Using BACnet Protocol,” in *Advances in Computing and Information Technology*, D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, and D. Nagamalai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 437–449.

[15] Y. C. Li, S. H. Hong, X. H. Li, Y. C. Kim, and M. Alam, “Implementation of a BACnet-EnOcean gateway in buildings,” in *Proceedings of 2014 International Conference on Intelligent Green Building and Smart Grid, IGBSG 2014*. IEEE Computer Society, 2014.

[16] Beckhoff Automation GmbH & Co. KG., “C6030-0060 | Ultra-Kompakt-Industrie-PC | Beckhoff Österreich.” [Online]. Available: https://www.beckhoff.com/de-at/produkte/ipc/pcs/c60xx-ultra-kompakt-industrie-pcs/c6030-0060.html

[17] ——, “TE1000 | TwinCAT 3 Engineering | Beckhoff Österreich.” [Online]. Available: https://www.beckhoff.com/de-at/produkte/automation/twincat/texxxx-twincat-3-engineering/te1000.html

[18] Schrack Seconet, “TwinCAT ISP-IP Guide.”

[19] Beckhoff Automation GmbH & Co. KG, “Beckhoff information system bacnet sps-automapping rev12.” [Online]. Available: https://infosys.beckhoff.com/index.php?content=../content/1031/tcbacnet/html/bacnet_automappingplc.htm&id=

[20] Beckhoff Automation GmbH & Co. KG., “TwinCAT 3 Library BACnet Revision 14 Documentation,Version: 1.0.0.”

[21] Morten Kvistgaard, Frédéric Chaxel, Adam Guzik, Christopher Günther, Thamer Al-Salek, Lance Tollenaar, and Frank Schubert, “Yet Another Bacnet Explorer| SourceForge.net.” [Online]. Available: https://sourceforge.net/projects/yetanotherbacnetexplorer/

[22] Cimetrics Inc., “Cimetrics BACnet Explorer Free.” [Online]. Available: https://www.cimetrics.com/products/bacnet-explorer?variant=31119750987870

[23] Inneasoft, “Inneasoft BACnet Explorer Free.” [Online]. Available: https://www.inneasoft.com/en/bacnet-explorer/

[24] Beckhoff Automation GmbH & Co. KG., “TC1000 | TwinCAT 3 ADS.” [Online]. Available: https://www.beckhoff.com/de-at/produkte/automation/twincat/tc1xxx-twincat-3-base/tc1000.html

[25] Ela-compil sp. z o.o., “ela-compil/BACnet: BACnet protocol library for .NET | GitHub.” [Online]. Available: https://github.com/ela-compil/BACnet

[26] ——, “NuGet C#-BACnet Stack.” [Online]. Available: https://www.nuget.org/packages/BACnet/2.0.4

[27] ——, “BACnet-Example AnotherStorageImplementation | GitHub.” [Online]. Available: https://github.com/ela-compil/BACnet.Examples/tree/master/AnotherStorageImplementation

[28] ——, “BACnet-Example BaCSharpObject.cs | GitHub.” [Online]. Available: https://github.com/ela-compil/BACnet.Examples/blob/master/AnotherStorageImplementation/BacnetObjects/BaCSharpObject.cs

[29] ——, “BACnet-Example StructuredView.cs | GitHub.” [Online]. Available: https://github.com/ela-compil/BACnet.Examples/blob/master/AnotherStorageImplementation/BacnetObjects/StructuredView.cs

[30] Beckhoff Automation GmbH & Co. KG., “Beckhoff BACnet conformance certificate.”

[31] Chipkin Automation Systems, “BACnet Stack | Chipkin.” [Online]. Available: https://www.bacnetstack.com/

[32] Cimetrics Inc., “BACnet Protocol Cimetrics.” [Online]. Available: https://www.cimetrics.com/collections/bacstac