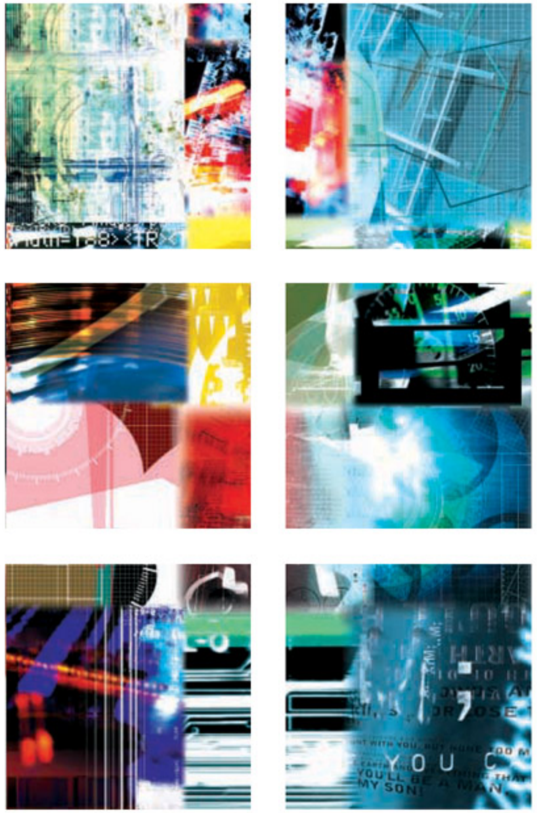


www.knx.org

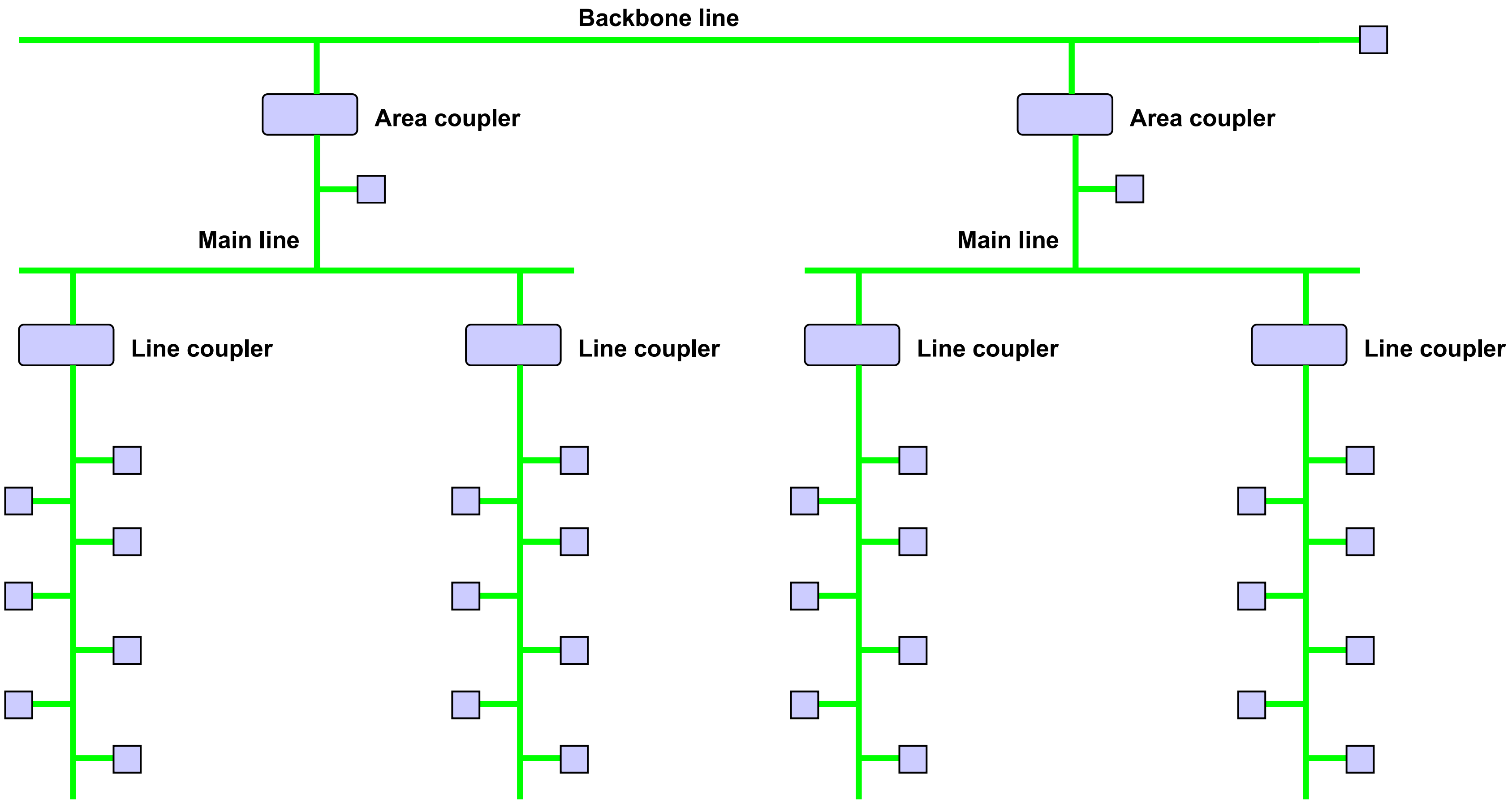
KNX IP – using IP networks as KNX medium

Dipl.-Ing. Hans-Joachim Langels
Siemens AG
Industry Sector
Building Technologies
Control Products and Systems
Regensburg
hans-joachim.langels@siemens.com

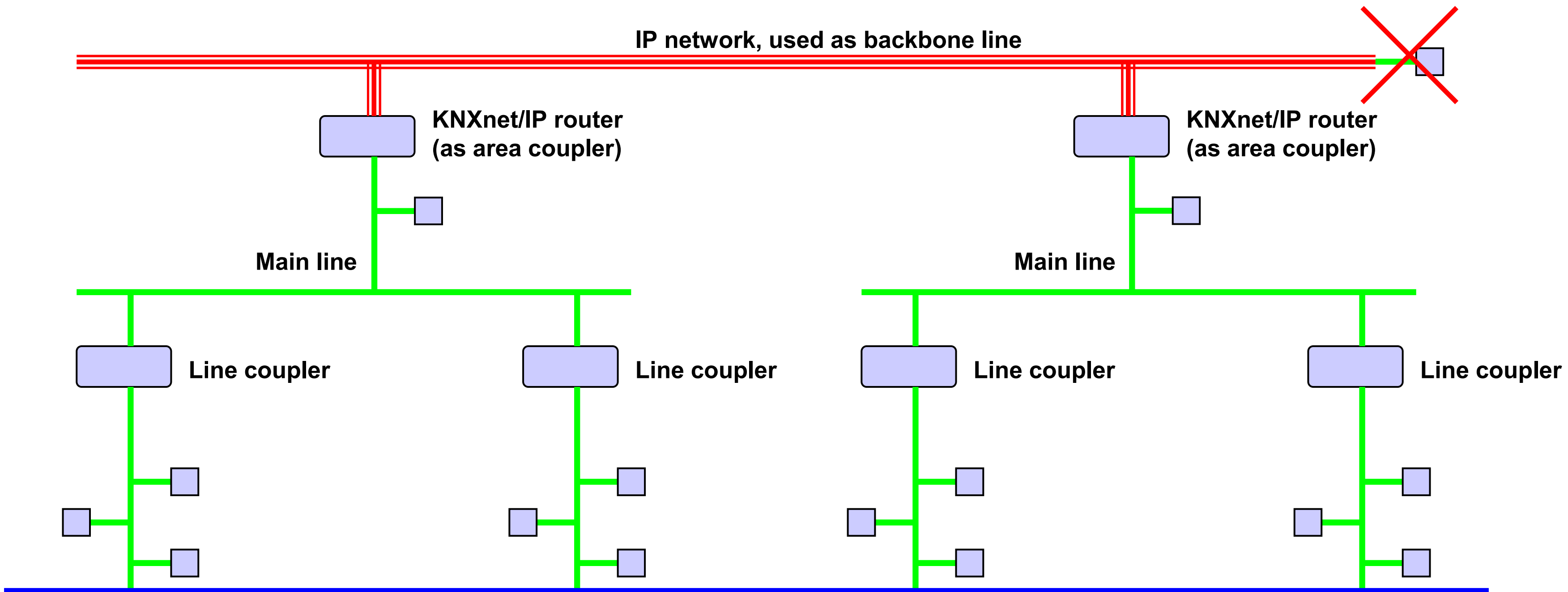
Overview

- From KNX to KNXnet/IP
- KNX IP
  - Concept and System Rules
  - Addressing potential issues
    - Device Requirements
    - Device Design Recommendations
  - ROUTING_BUSY
- Conclusion and Outlook

From KNX to KNXnet/IP



From KNX to KNXnet/IP



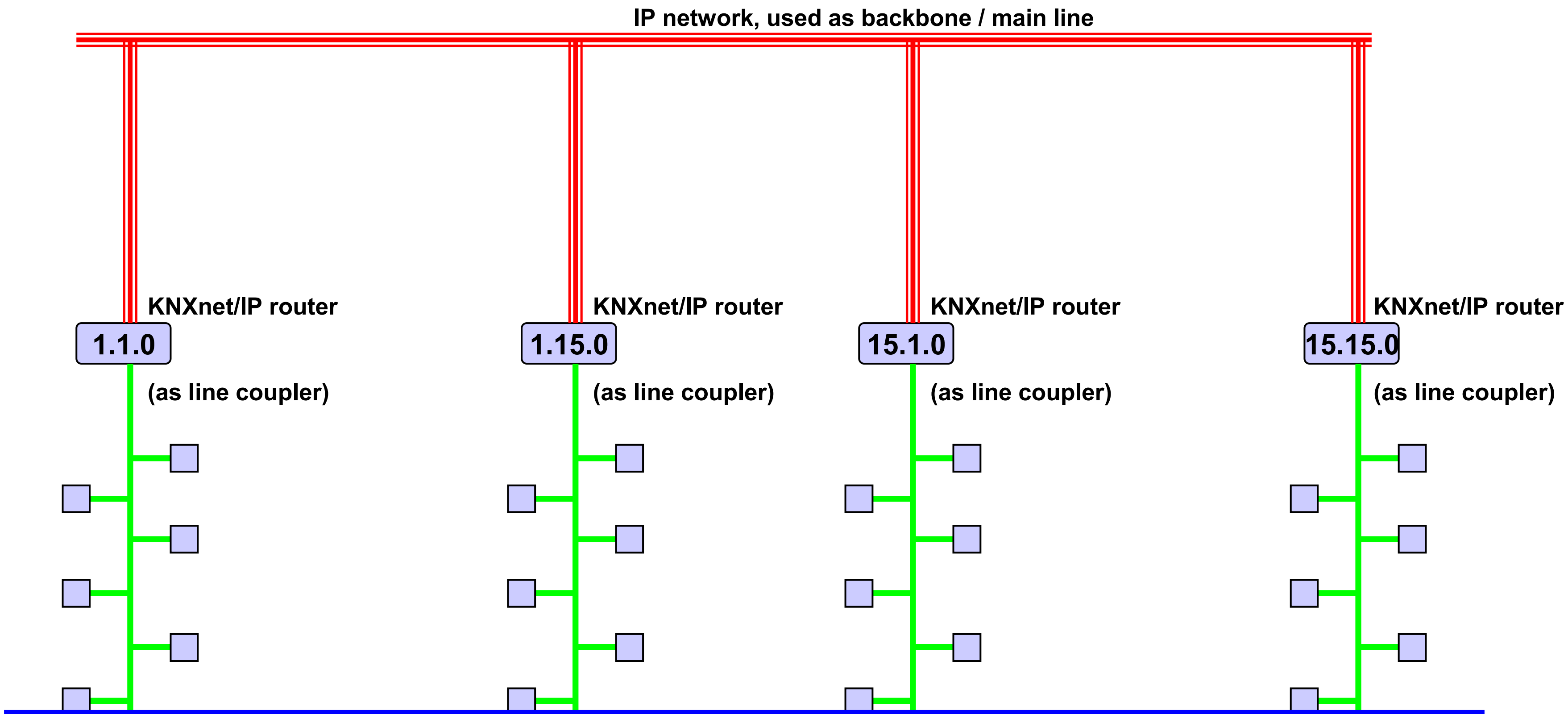
Rule 1

In general a KNXnet/IP Router may be used as a Line Coupler or a Backbone Coupler.
The Individual Address has the format x.y.0, with x = 1 to 15 and y = 0 to 15.

Rule 2:

If a KNXnet/IP Router is applied as a Backbone Coupler with the Individual Address x.0.0 then no other KNXnet/IP Router with the Line Coupler Individual Address x.y.0 (y = 1 to 15) shall be placed topologically „below“ this KNXnet/IP Router.

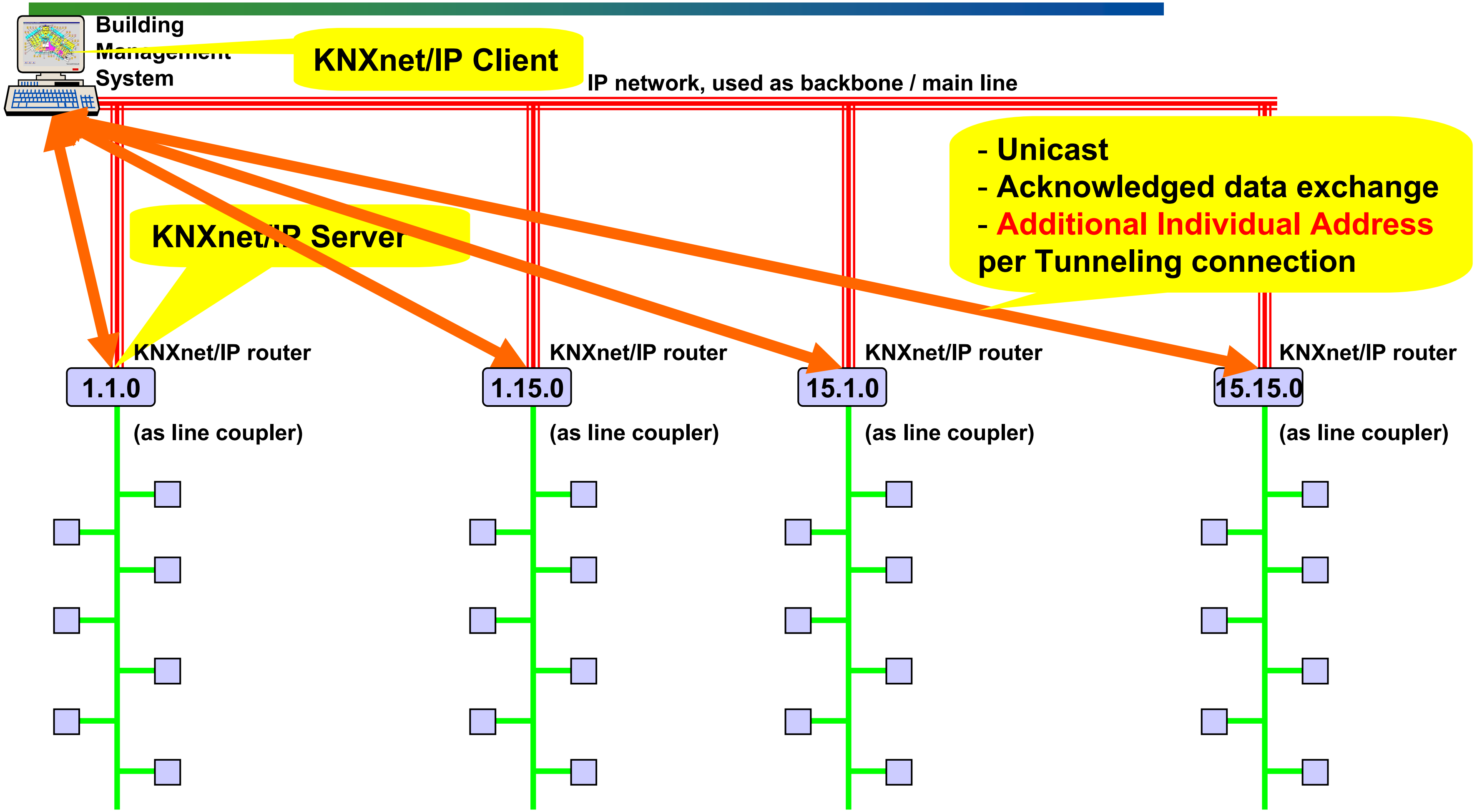
From KNX to KNXnet/IP



Rule 3:

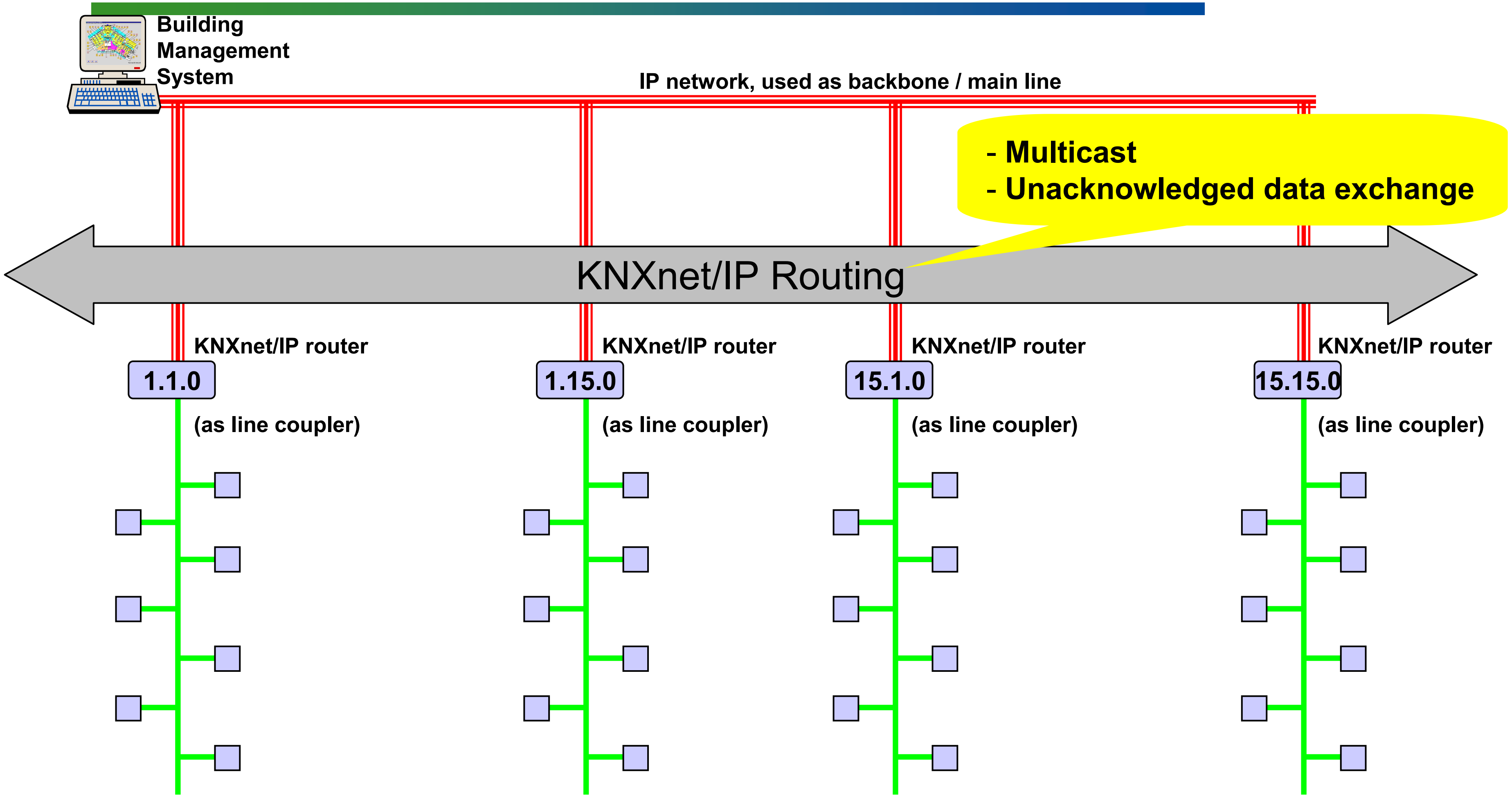
If a KNXnet/IP Router is applied as a Line Coupler (e.g. with Individual Address 1.2.0) then no other KNXnet/IP Router shall be used with a superior Backbone Coupler Individual Address (e.g. 1.0.0) in this installation.

KNXnet/IP Tunneling



- Unicast
- Acknowledged data exchange
- Additional Individual Address per Tunneling connection

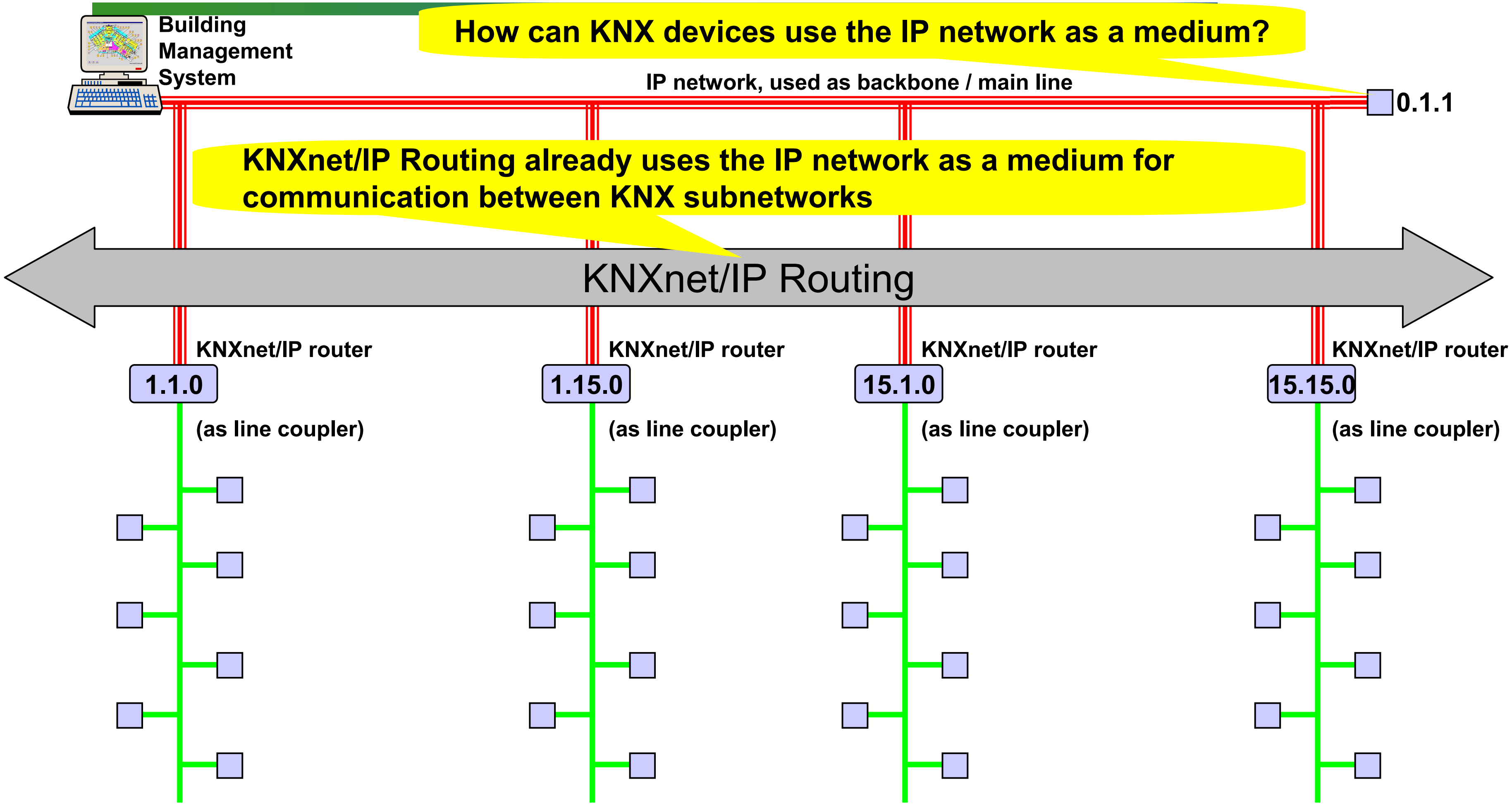
KNXnet/IP Routing



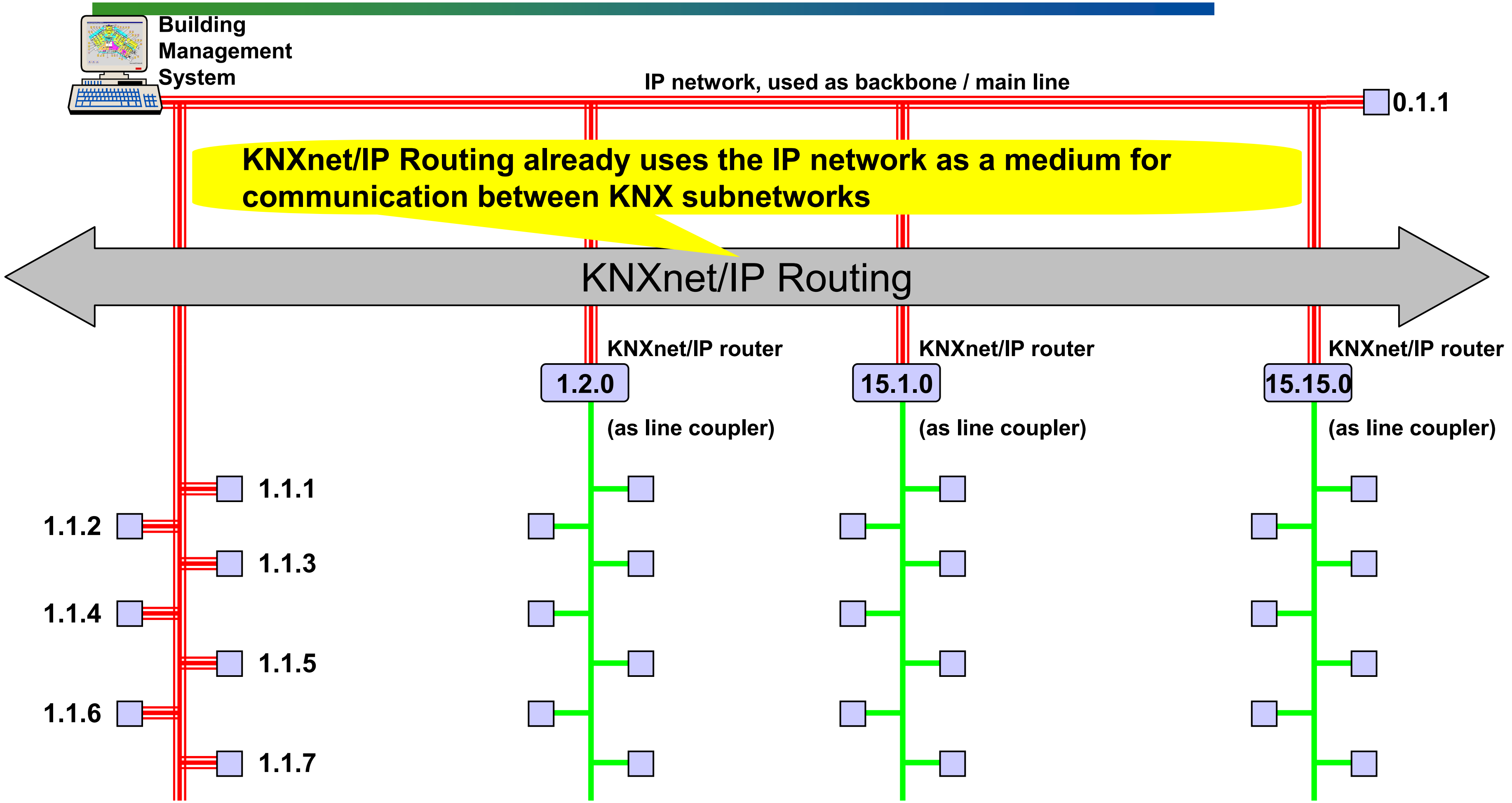
Using KNXnet/IP Routing for KNX IP

How can KNX devices use the IP network as a medium?

KNXnet/IP Routing already uses the IP network as a medium for communication between KNX subnetworks



Using KNXnet/IP Routing for KNX IP



KNXnet/IP Routing already uses the IP network as a medium for communication between KNX subnetworks

Rules

Rule 1

In general a KNXnet/IP Router may be used as a Line Coupler or a Backbone Coupler. The Individual Address has the format x.y.0, with x = 1 to 15 and y = 0 to 15.

Rule 2:

If a KNXnet/IP Router is applied as a Backbone Coupler with the Individual Address x.0.0 then no other KNXnet/IP Router with the Line Coupler Individual Address x.y.0 (y = 1 to 15) shall be placed topologically „below“ this KNXnet/IP Router.

Rule 3:

If a KNXnet/IP Router is applied as a Line Coupler (e.g. with Individual Address 1.2.0) then no other KNXnet/IP Router shall be used with a superior Backbone Coupler Individual Address (e.g. 1.0.0) in this installation.

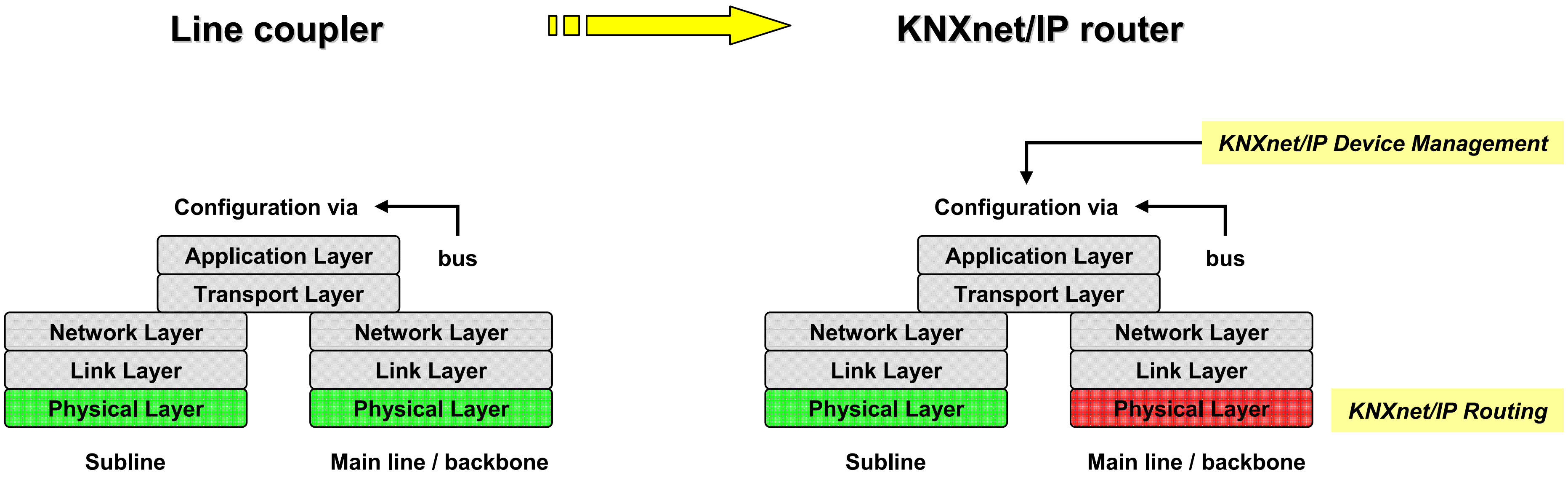
Rule 4:

If a KNX IP device is assigned to a Subnetwork as a simple device (e.g. with Individual Address 1.0.1) then that Subnetwork and any Subnetwork higher in the system structure shall contain KNX IP devices only.

KNX Task Force IP Design Objectives

- Maintain the simplicity and scalability of the KNX system
- Use management procedures already supported by ETS
- Enable reusing existing stack implementations whereever possible
- Add capabilities that enhance the KNX system for advanced applications

From KNX to KNXnet/IP



Line coupler

KNXnet/IP router

Configuration via

bus

Application Layer	
Transport Layer	
Network Layer	Network Layer
Link Layer	Link Layer
Physical Layer 🟩	Physical Layer 🟩
Subline	Main line / backbone

KNXnet/IP Device Management

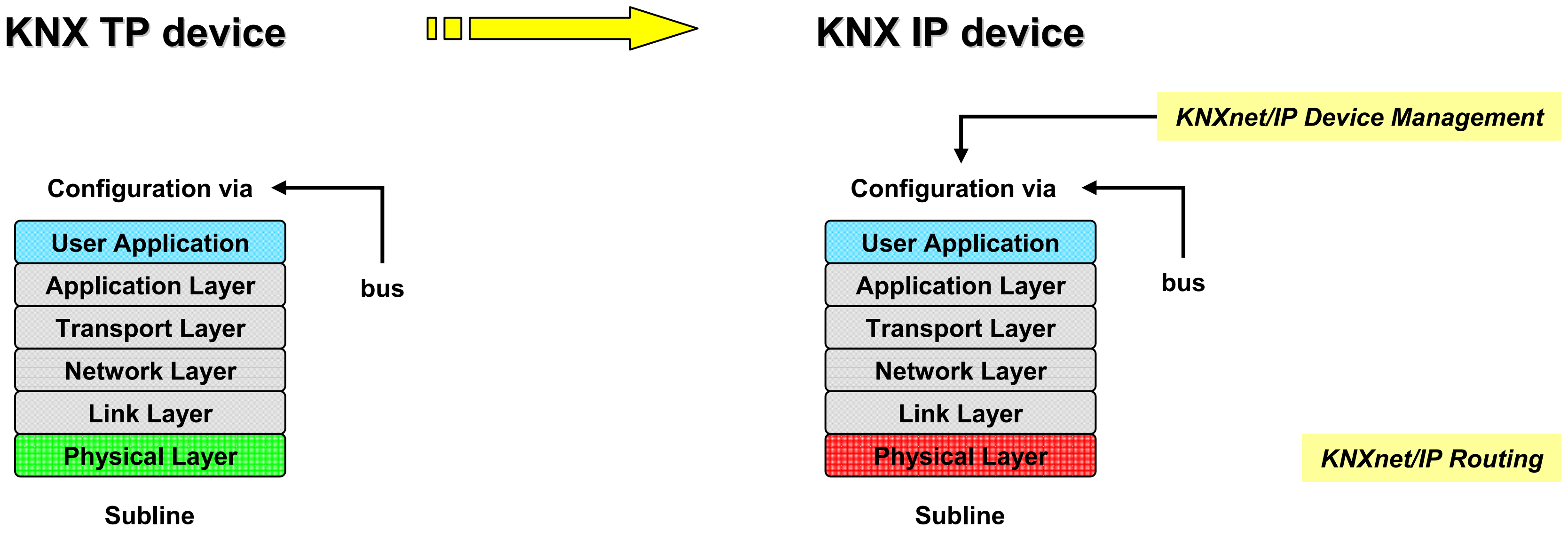
Configuration via

bus

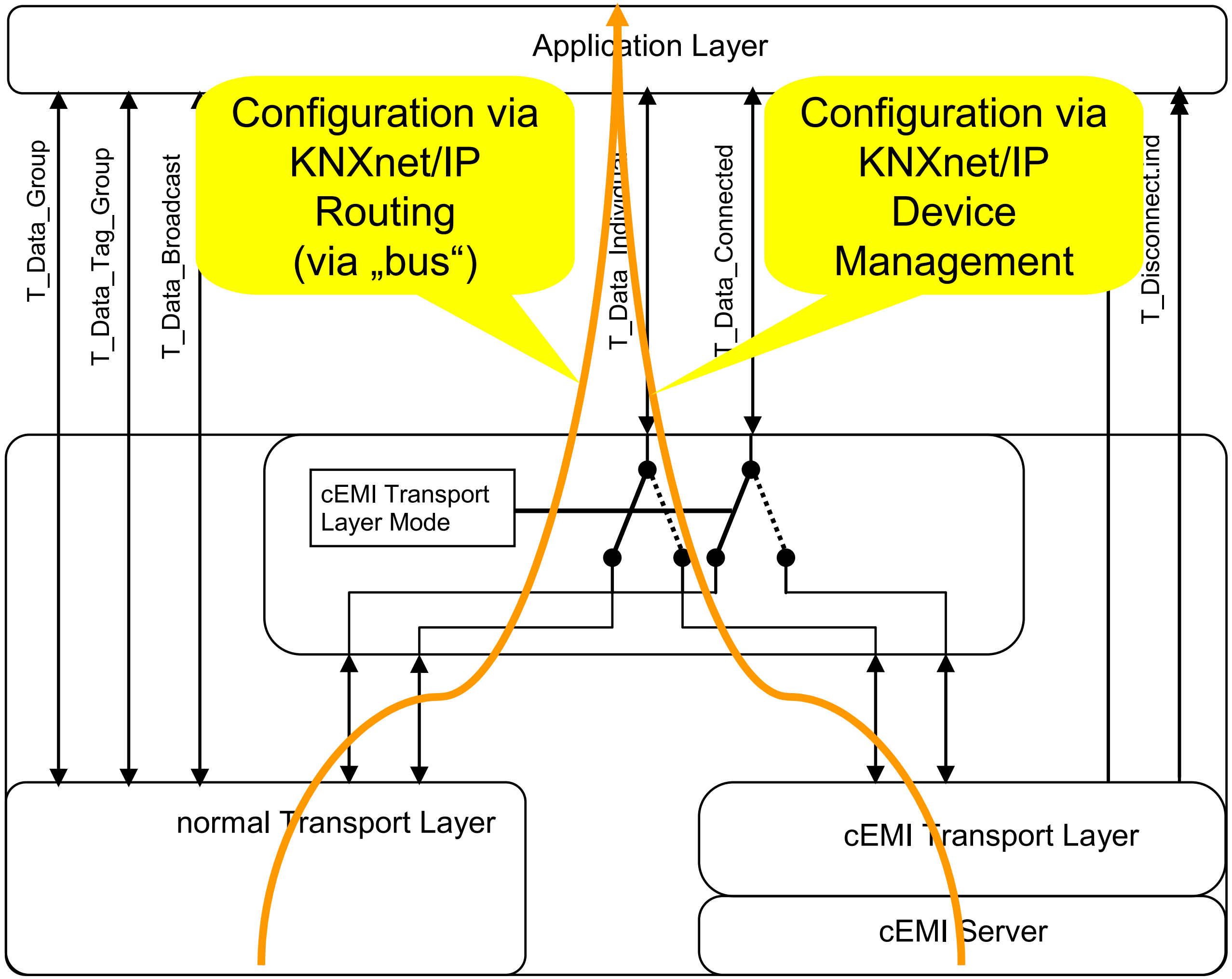
Application Layer	
Transport Layer	
Network Layer	Network Layer
Link Layer	Link Layer
Physical Layer 🟩	Physical Layer 🟥
Subline	Main line / backbone

KNXnet/IP Routing

From KNX TP to KNX IP



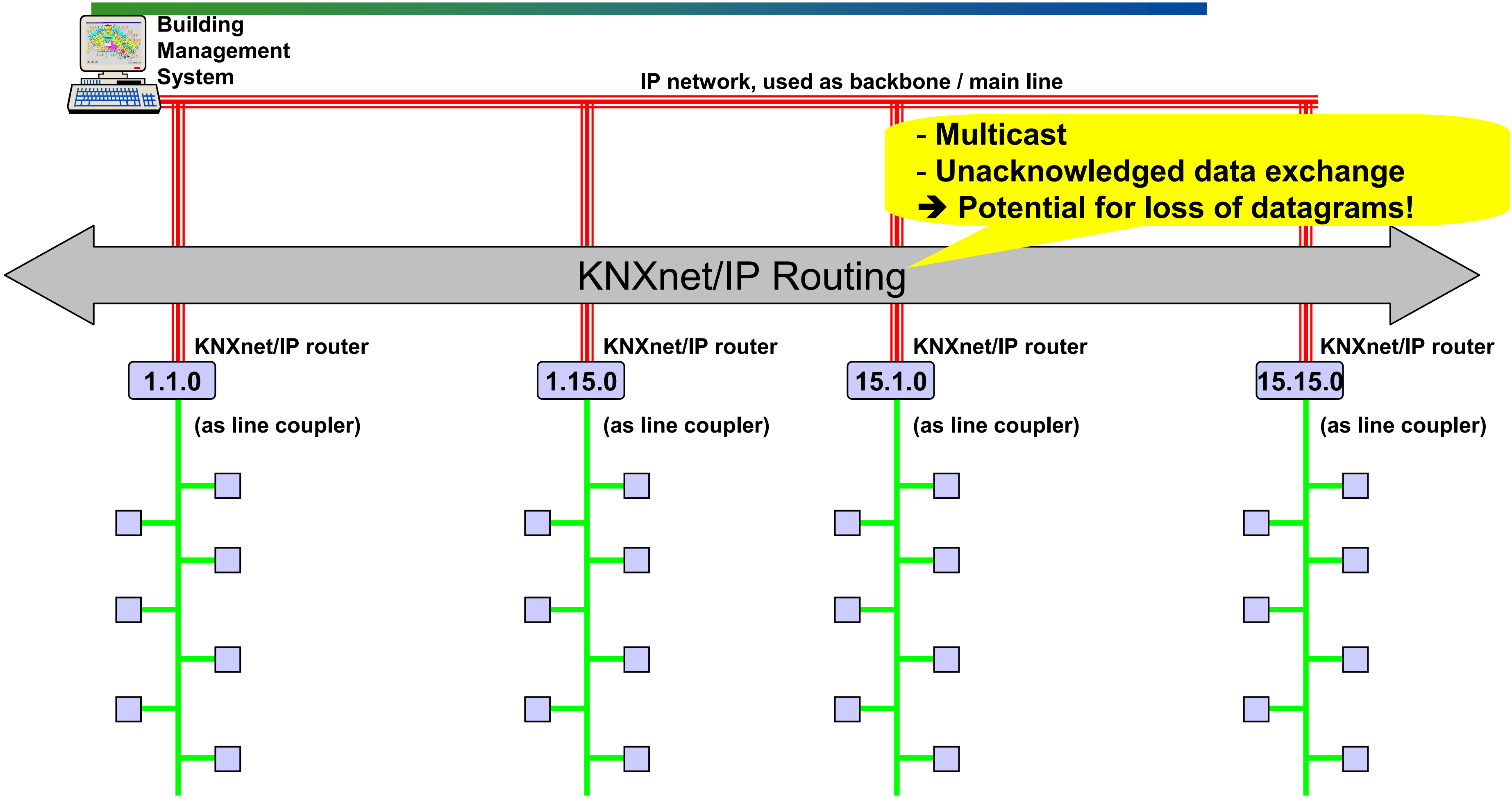
Configuration of KNX IP devices



KNXnet/IP Device Mangement

- KNXnet/IP Device Management defines a DEVICE_CONFIGURATION_REQUEST, which carries a cEMI frame with the configuration message.
- The original cEMI configuration message for the cEMI server in the KNXnet/IP device defines property services (M_PropRead, M_PropWrite, ...) that use Interface Object Type and Interface Object Instance.
- To use existing management procedures cEMI Transport Layer for „local management“ was defined.
- KNXnet/IP devices implementing these cEMI services SHALL announce Device Management service version 1.1 when responding to a SEARCH_REQUEST.

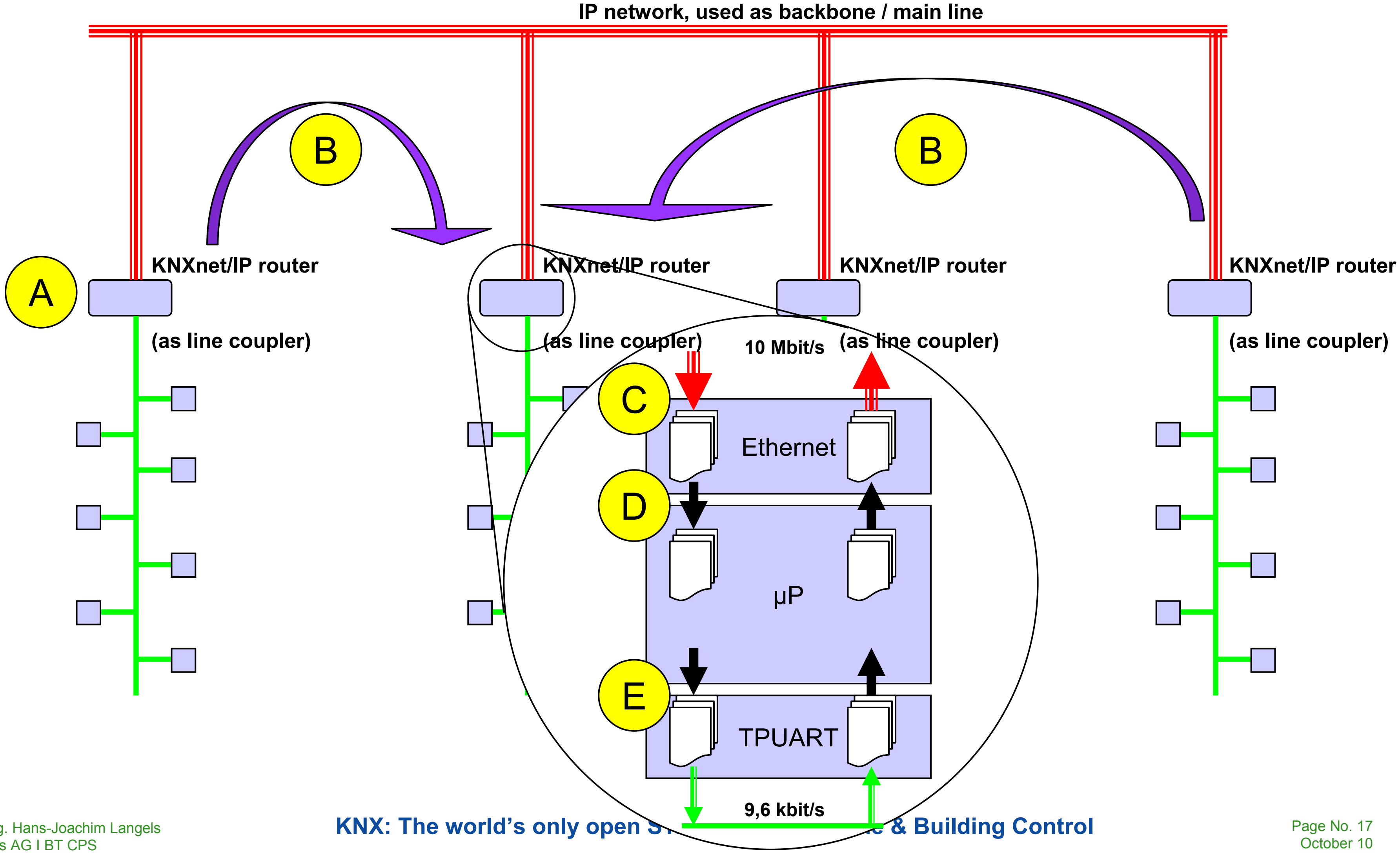
Potential issues for KNX IP



- Multicast
- Unacknowledged data exchange

➔ Potential for loss of datagrams!

Potential issues for KNX IP



Potential issues for KNX IP

A

- Routing from KNX subnetwork to KNXnet/IP Routing Multicast Address

B

- Transmission across the IP network

C

- Reception from the IP network

D

- Interface within KNXnet/IP router or KNX IP device between Ethernet interface and microprocessor

E

- Interface within KNXnet/IP router to KNX subnetwork

or

- Interface within KNX IP device between KNX IP communication stack and application

Potential issues for KNX IP

A

- **Routing from KNX subnetwork to KNXnet/IP Routing Multicast Address**

- **Because Ethernet has at least 1000 times the transmission capacity of a KNX TP (PL, RF) subnetwork,
routing TO KNXnet/IP is not an issue.**
- **KNXnet/IP routers can only transmit a maximum of 50 telegrams per second to their subnetwork (KNX TP, PL, RF).**

➢ ***"Any KNX IP device or KNXnet/IP router (including ETS) SHALL limit the transmission of KNX IP ROUTING_INDICATION datagrams to a maximum of 50 datagrams per second within one second."***

➢ ***"A KNX IP device or KNXnet/IP Router SHALL always pause its transmission on an assigned multicast address for at least 5ms after it transmitted a ROUTING_INDICATION datagram."***

Potential issues for KNX IP

B
- **Transmission across the IP network**

- **If KNX IP datagrams are lost in an IP network this cannot be detected by the sender because multicast datagrams are not acknowledged.**

➢ **Any communication with supervisory systems (including visualizations) or ETS should use KNXnet/IP Tunneling.**

Potential issues for KNX IP

C
- Reception from the IP network

- An issue with the Ethernet interface performance cannot be removed by whatever protocol measures would be applied.

➢ It is the responsibility of the manufacturer to select an Ethernet interface that is suitable for the network data rate (e.g. 10 Mbit/s).

Potential issues for KNX IP

D

- Interface within KNXnet/IP router or KNX IP device between Ethernet interface and microprocessor

- Datagrams may be lost without detection by the KNXnet/IP router or KNX IP device if datagrams are received by the Ethernet interface but cannot be quickly enough transmitted to or processed by the microprocessor.

➢ It is the responsibility of the manufacturer to select and design hardware, firmware, operating system, and/or application software suitable for KNX IP performance.

Potential issues for KNX IP

- C Reception from the IP network
- D Interface within KNXnet/IP router or KNX IP device between Ethernet interface and microprocessor

➢ Test cases to measure the performance of KNX IP devices and KNXnet/IP routers are defined.

- **Requirement:**
  "A KNXnet/IP Router or KNX IP device SHALL be able to receive and process up to the KNX Network - respectively Application Layer at least 1000 ROUTING_INDICATION frames per second."
- **Recommendation:**
  "Any KNX IP device or KNXnet/IP Router SHOULD be capable of receiving and processing at least 12750 ROUTING_INDICATION datagrams per second on an assigned multicast address."

Potential issues for KNX IP

E

- Interface within KNXnet/IP router to KNX subnetwork

  or
- Interface within KNX IP device between KNX IP communication stack and application

- Flow control needs to be introduced for KNXnet/IP routers and KNX IP devices to avoid the loss of datagrams due to overflowing queues in KNXnet/IP routers and KNX IP devices.

➢ ROUTING_BUSY is introduced for a receiving device to indicate to all other devices that its incoming queue is filling up and it may loose datagrams if they do not stop sending.

ROUTING_BUSY

- **ROUTING_BUSY is intended to take care of potential datagram losses due to temporary datagram rate differences between the IP network and a KNX subnetwork.**

KNXnet/IP header

```
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   HEADER_SIZE_10              |   KNXNETIP_VERSION            |
|   (06h)                       |   (10h)                       |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   ROUTING_BUSY                                                |
|   (0532h)                                                     |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   HEADER_SIZE_10 + 6                                          |
|                                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

BusyInfo

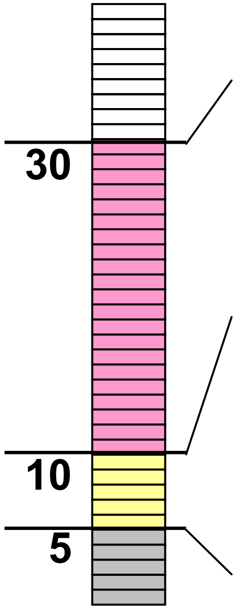
```
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   Structure Length            |   DeviceState                 |
|   (1 octet)                   |   (1 octet)                   |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   ROUTING_BUSY_WAIT_TIME (in milliseconds)                    |
|   (2 octets)                                                  |
+---+---+---+---+---+---+---+---+---+---+---+--
|   ROUTING_BUSY_CONTROL_FIELD
|   (2 octets)
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Minimum: 20ms
Maximum: 100ms (typical)

Default value: 0000h (= all KNX IP devices)
Other values: reserved for future use

ROUTING_BUSY Flow Control

Recommendations based on a system simulation assuming that up to 255 devices send 50 ROUTING_INDICATION datagrams per second:



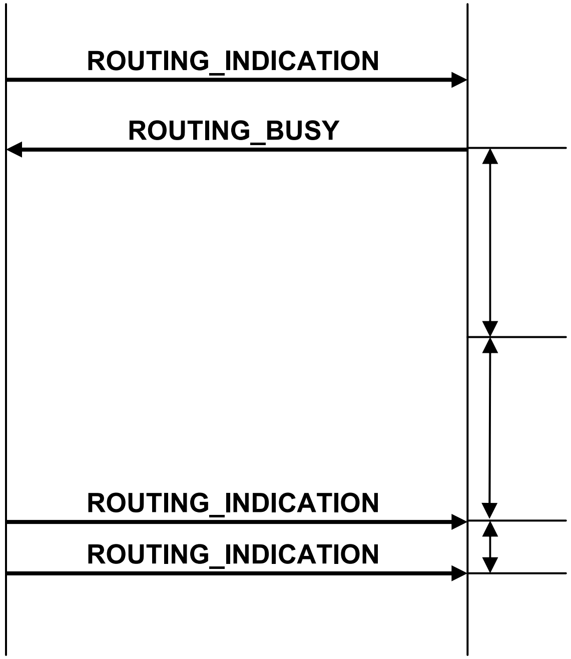
The incoming (from KNX IP) queue SHOULD be able to hold at least 30 messages.

The threshold for sending a ROUTING_BUSY frame to all KNX/IP devices and KNXnet/IP Routers SHOULD be set at ten messages in the incoming queue.

The threshold for sending a ROUTING_BUSY frame with the individual address from the last ROUTING_INDICATION frame SHOULD be set at five messages in the incoming queue.

ROUTING_BUSY

- **ROUTING_BUSY provides**
  - time to empty the queue from KNX IP to a KNX subnetwork
  - a random restart of transmissions to avoid a flooding effect after the wait time t_w has expired



t_w <= 100 ms
(depends on ROUTING_BUSY_WAIT_TIME received with ROUTING_BUSY)

t_{random} <= $t_{random, max}$ = N * 50 ms
(N = number of ROUTING_BUSY received;
N is decremented every 5 ms after [t_{random} + N * 100 ms] expired)

t >= 5 ms

- **Filter tables in KNXnet/IP Routers MUST be activated!**

Conclusions and Outlook

- KNX IP is defined based on the existing KNXnet/IP Routing protocol.
- cEMI Transport Layer has been defined for configuration of KNX IP devices.
- Requirements and recommendations for the design and behavior of KNX IP devices address the fact that network performance and KNX IP device performance influence the overall KNX IP system performance.

- KNX IP further expands the foundation for active participation of KNX in environments increasingly driven by the Internet Protocol.

KNX IP – using IP networks as KNX medium

Dipl.-Ing. Hans-Joachim Langels
Siemens AG
Siemensstraße 10, 93055 Regensburg
Tel: 0941-790-2992
E-Mail: hans-joachim.langels@siemens.com
www.siemens.de/gamma
www.din-bauportal.de/siemens

Abstract

The versatility of KNX is based on its protocol but also on its different media: twisted pair (TP), power line (PL), and radio frequency (RF).

KNXnet/IP as standardized in EN 13321-2 was introduced in 2004 to enable usage of the ubiquitous IP networks for data transmission between KNX subnetworks (→ KNXnet/IP Routing), with supervisory systems (→ KNXnet/IP Tunneling), and with configuration tools (→ KNXnet/IP Device Management).

Since then products using KNXnet/IP, especially KNXnet/IP routers and interfaces, have become part of the standard product offering.

As an answer to the desire to have KNX devices use existing IP networks as a "medium" for peer-to-peer communication KNX Task Force IP has defined KNX IP, which is based on KNXnet/IP Routing.

KNX/IP is another step in expanding the deployment of KNX into new application fields and interfacing easily with entertainment and information communication technology.

This paper summarizes the definitions for KNX IP.

Introduction

The ubiquitous presence of the Internet Protocol (IP) has led to the definition of KNXnet/IP. KNXnet/IP provides the means for point-to-point connections (KNXnet/IP Tunneling) for ETS and/or between a supervisory system and a KNX installation.

KNXnet/IP Device Management provides configuration of KNXnet/IP devices through the network effectively reducing the time required for configuration.

It also defines how lines or areas may interconnect using IP networks via KNXnet/IP Routing. Yet, KNXnet/IP Routing in itself only defines how KNXnet/IP routers communicate with each other using IP networks.

The desire is to have KNX devices use the IP networks as a "medium" for peer-to-peer communication.

The KNX system topology for twisted pair is hierarchical as depicted in figure 1.

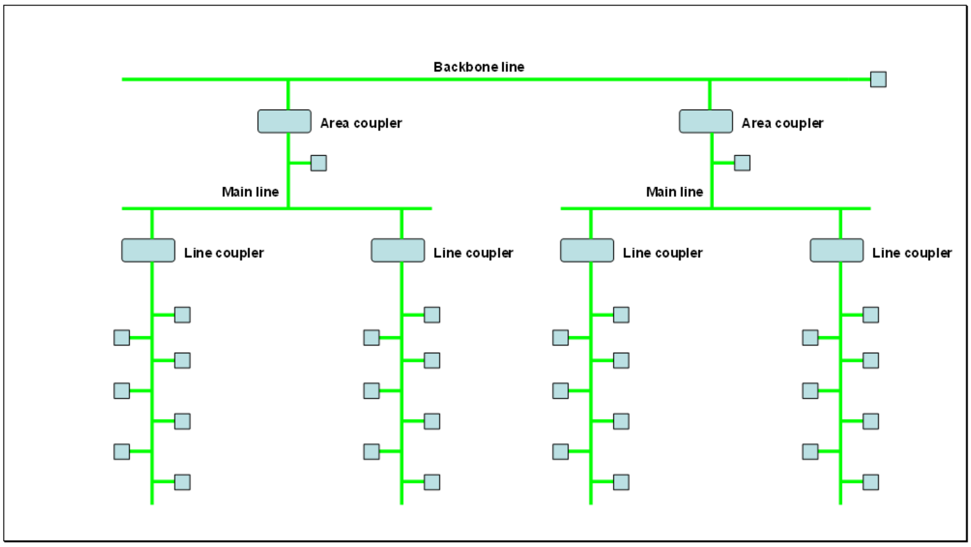


Figure 1

With KNXnet/IP routers replacing the area couplers the topology changes as depicted in figure 2.

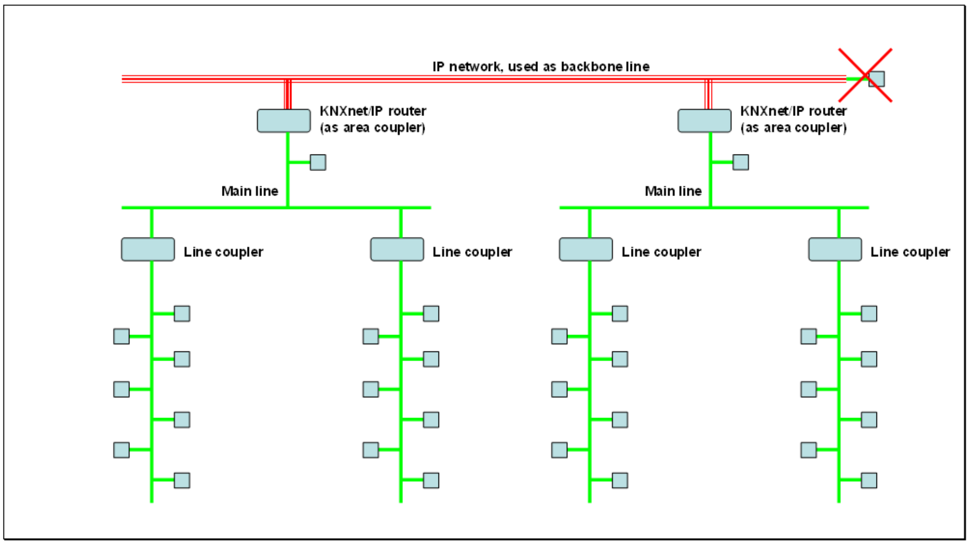


Figure 2

The IP network is used as the backbone for KNX. The communication protocol between KNXnet/IP routers is the KNXnet/IP Routing protocol. A KNXnet/IP router is defined as a device that is on one side connected to a KNX subnetwork (either KNX TP, KNX PL, or KNX RF) and on the other side is connected to an IP network, that is used as the backbone line.

Further collapsing the system topology, the line couplers are replaced by KNXnet/IP routers and the backbone couplers are omitted as depicted in figure 3.

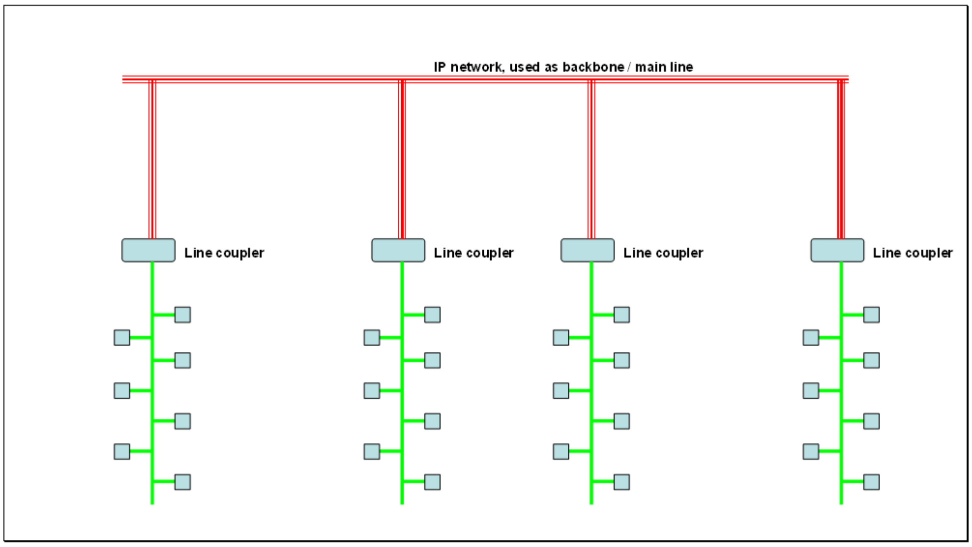


Figure 3

While the traditional KNX TP topology allows for KNX devices to be on the KNX backbone line or the main line, the initial KNXnet/IP topology does not define KNX devices other than KNXnet/IP routers or interfaces to be connected to the IP network. It also assumes that

KNXnet/IP devices have two interfaces: one to the KNX subnetwork and one to the IP network.

Users and manufacturers expressed the desire to use the IP network as a native medium for KNX. Consequently, these KNX IP devices would only have one physical interface i.e. the interface to the IP network as depicted in figure 4.

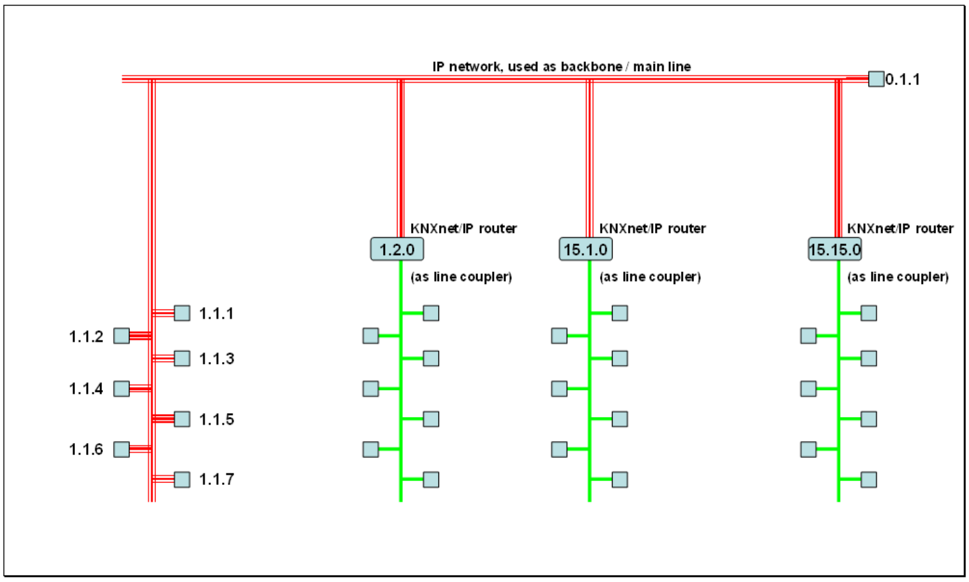


Figure 4

Figures 2, 3 and 4 also depict a set of rules for KNXnet/IP routers and KNX IP devices.

Rule 1

In general a KNXnet/IP Router may be used as a Line Coupler or a Backbone Coupler. The Individual Address has the format x.y.0, with x = 1 to 15 and y = 0 to 15.

Rule 2:

If a KNXnet/IP Router is applied as a Backbone Coupler with the Individual Address x.0.0 then no other KNXnet/IP Router with the Line Coupler Individual Address x.y.0 (y = 1 to 15) shall be placed topologically „below“ this KNXnet/IP Router.

Rule 3:

If a KNXnet/IP Router is applied as a Line Coupler (e.g. with Individual Address 1.2.0) then no other KNXnet/IP Router shall be used with a superior Backbone Coupler Individual Address (e.g. 1.0.0) in this installation.

Rule 4:

If a KNX IP device is assigned to a Subnetwork as a simple device (e.g. with Individual Address 1.0.1) then that Subnetwork and any Subnetwork higher in the system structure shall contain KNX IP devices only.

KNX Task Force IP Design Objectives

KNX Task Force IP developed the KNXnet/IP protocol following a few design objectives:

- maintain the simplicity and scalability of the KNX system
- use management procedures already supported by ETS
- enable reusing existing stack implementations whereever possible
- add capabilities that enhance the KNX system for advanced applications

Following these design objectives the KNXnet/IP Routing protocol allowed quick development of KNXnet/IP routers that operate like a line coupler. This enables replacing line couplers with KNXnet/IP routers while simplifying the system topology as depicted in Figure 3. The changes to ETS were minimal, which means that installers could start using KNXnet/IP routers quickly. Yet, the KNXnet/IP protocol defines a number of features that may be used by future ETS versions.

KNX IP

As a first step a few terms need to be defined before going into further details on how to use IP networks as a KNX medium.

- A KNXnet/IP Server is a KNX device that has physical access to a KNX network and implements the KNXnet/IP Server protocol to communicate with KNXnet/IP Client or other KNXnet/IP Servers (in case of routing) on an IP network channel. A KNXnet/IP Server is by design always also a KNX node.
- A KNXnet/IP Client is an application that implements the KNXnet/IP Client protocol to get access to a KNX Subnetwork over an IP network channel.
- A KNXnet/IP Router is a special type of KNXnet/IP device that routes KNX protocol packets between KNX Subnetworks.
- KNX IP is the term used when the Internet Protocol (IP) is utilized as a KNX medium.
- KNX IP device describes a KNX device using the Internet Protocol (IP) as the only KNX medium.
- A KNX IP Router is a special type of KNX IP device that routes KNX protocol packets between its associated KNX IP Subnetwork and other KNX Subnetworks.

Following the design objectives KNX Task Force IP made a fundamental decision to use the existing KNXnet/IP Routing protocol as the basis for KNX IP.

Why?

KNX does not require the source of a KNX telegram to know who is supposed to receive that telegram. Based on KNX group addressing it allows the sender to transmit to an unknown number of recipients, which can be added or removed any time without affecting the sender configuration. Likewise, the number of sources sending to the same group address is not limited either, again without affecting the configuration of the recipients. How can this simplicity and scalability be transposed to IP networks?

In principle, either of these two IP addressing methods could be used: unicast or multicast.

If unicasting were used any KNXnet/IP router or KNX IP device would have to be configured for sending its data to a pre-defined list of other IP devices. This increases the engineering effort (new management procedures) and requires more resources in the devices but also of the network. Sending telegrams via unicast datagrams multiplies the number of datagrams by the number of recipients. Depending on the IP network characteristics and the number of recipients scalability is an issue as depicted in figure 5.

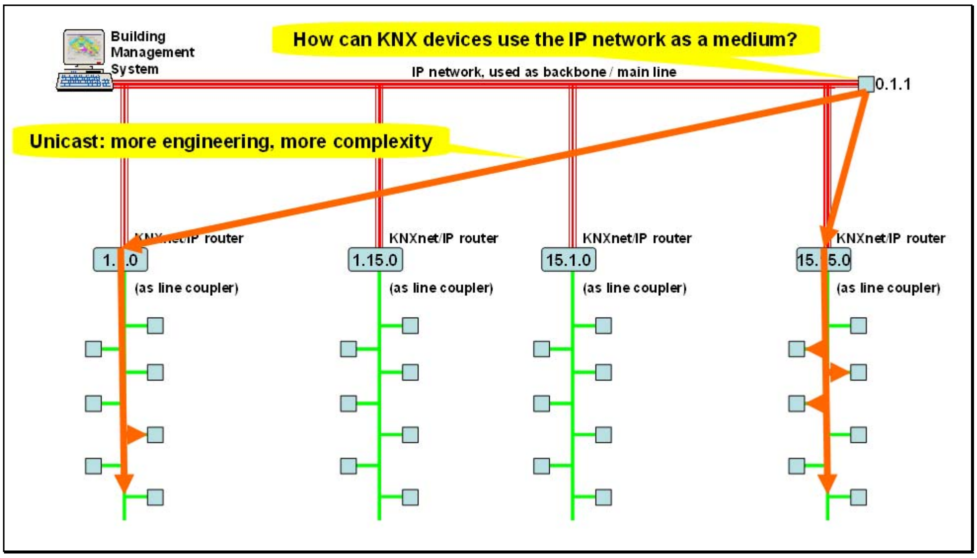


Figure 5

Unicast datagrams may be acknowledged, which ensures delivery across the network under different network conditions. This is why KNXnet/IP Tunneling and KNXnet/IP Device Management protocols use IP unicast addressing. Tracking acknowledgements for multiple unicast telegrams carrying the same information would further burden the sending devices and increase engineering effort far beyond what KNX requires for KNX TP (RF, PL) installations. This would breach the design objectives of scalability and simplicity but also require new management procedures and stack implementations.

IP multicast offers features similar to KNX group addressing. A multicast datagram is sent onto the network and the same multicast datagram may be received by one or many recipients at the same time. Hence, using multicast allows for scalability and retains the simplicity of the KNX system. This is why KNXnet/IP Routing uses multicast addressing. Following the objective of reusing existing stack implementations KNX Task Force IP decided to use KNXnet/IP Routing as the foundation for KNX IP communication (Figure 6).

KNX IP – using IP networks as KNX medium

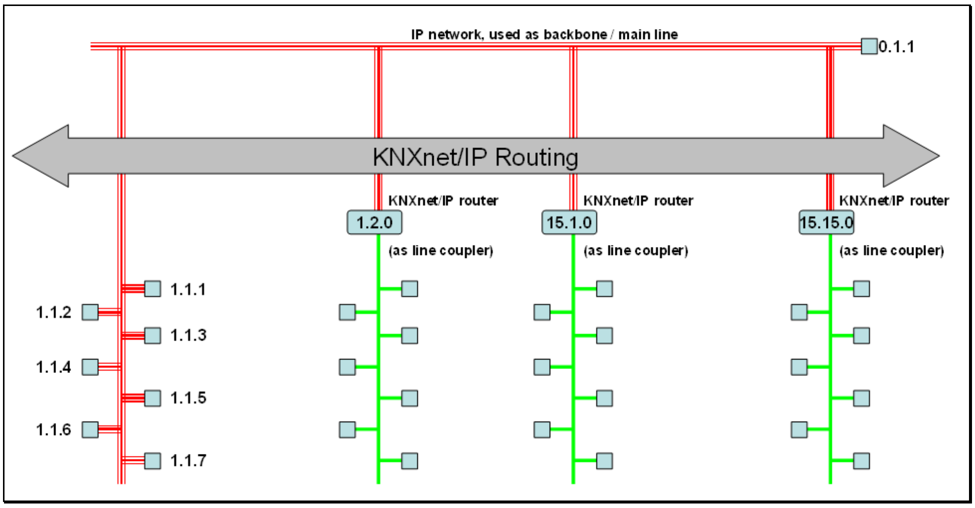


Figure 6

Figure 7 shows the evolution of the KNXnet/IP router from the line coupler.

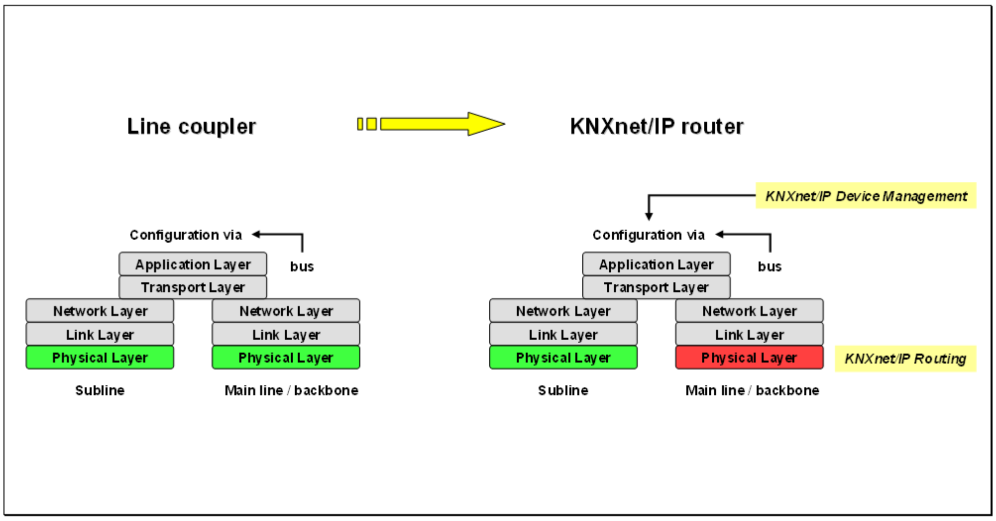


Figure 7

In a similar way KNX IP devices evolve from KNX devices (Figure 8). The existing stack and management procedures are retained but the transport has been changed from TP to KNXnet/IP. Configuration can be achieved through the KNX stack (via “bus”). In this case these configuration telegrams are sent via KNXnet/IP Routing. The second configuration option is available via KNXnet/IP Device Management. Originally, KNXnet/IP Device Management used cEMI property services for configuration. This is not sufficient to support existing management procedures for configuration. To retain these management procedures and the associated telegrams cEMI was enhanced by a cEMI Transport Layer (cEMI TL).

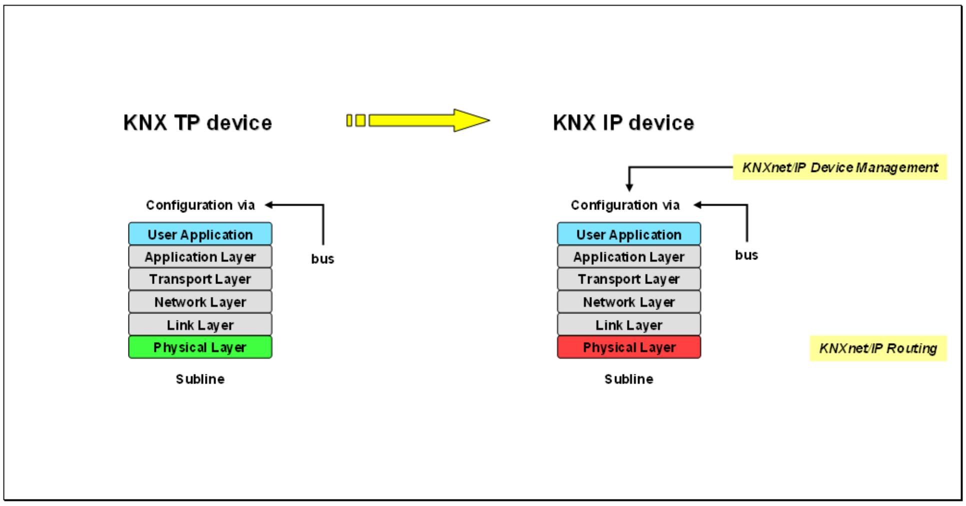


Figure 8

Figure 9 shows a schematic of a KNX IP device with the cEMI Transport Layer. The device may be either configured via the normal Transport Layer or the cEMI Transport Layer. The switch-over between these two Transport Layers is dependent on the connection established for KNXnet/IP Device Management. As long as such a connection is established the cEMI Transport Layer is activated.

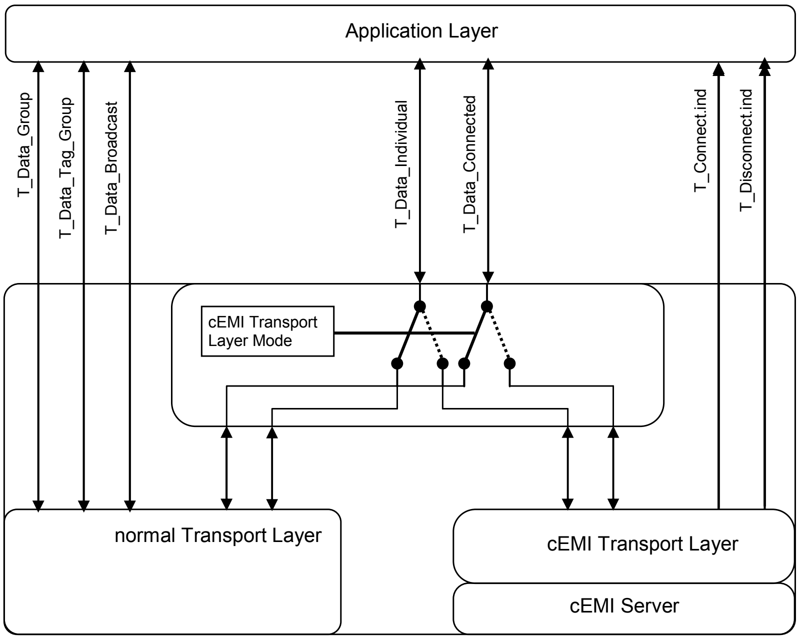


Figure 9

Configuration via KNXnet/IP Device Management is advantageous compared to configuration via KNXnet/IP Routing and the normal Transport Layer because KNXnet/IP Device Management establishes a point-to-point connection between ETS and the KNX IP device. While configuration via KNXnet/IP Tunneling has a one second time-out and configuration via KNXnet/IP Routing with normal Transport Layer has a three second time-out, configuration via KNXnet/IP Device Management has a time-out of ten (10) seconds making it suitable for long-distance configuration of KNX IP devices.

KNX IP solutions addressing potential issues

As described above KNX IP uses KNXnet/IP Routing as the communication basis. This has a number of implications that will be dealt with in this section.

Figure 10 depicts a schematic of a KNXnet/IP router that shows possible communication bottle-necks in a KNXnet/IP router or KNX IP device.

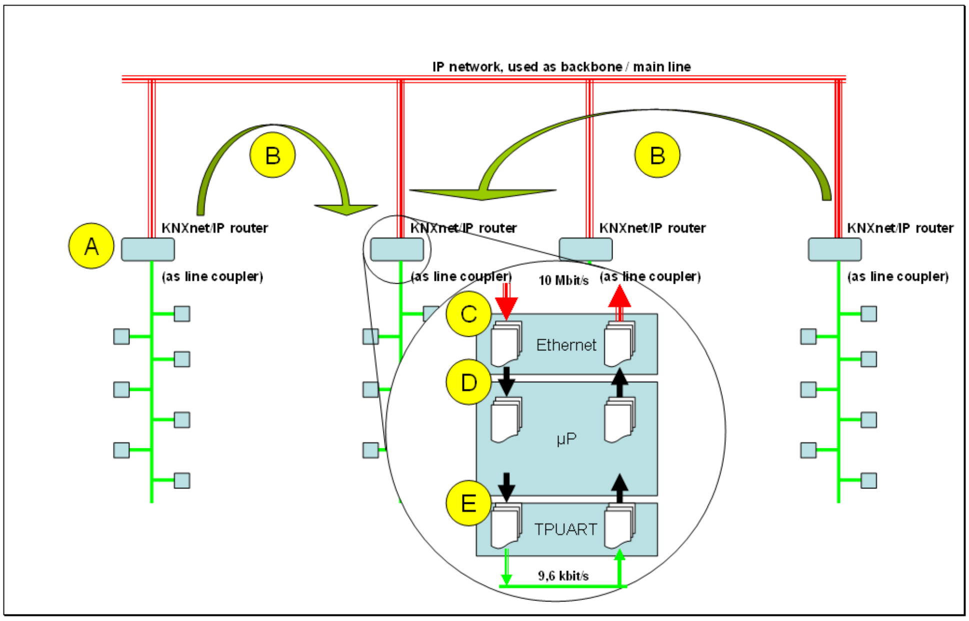


Figure 10

Following the direction of communication there are these possible locations for loss of telegrams / datagrams:

(A) Routing from KNX subnetwork to KNXnet/IP Routing Multicast Address

(B) Transmission across the IP network

(C) Reception from the IP network

(D) Interface within KNXnet/IP router or KNX IP device between Ethernet interface and micro processor

(E1) Interface within KNXnet/IP router to KNX subnetwork

(E2) Interface within KNX IP device between KNX IP communication stack and application

(A) Routing from KNX subnetwork to KNXnet/IP Routing Multicast Address

A KNXnet/IP router forwards telegrams from its KNX subnetwork to the IP network using the KNXnet/IP Routing Multicast Address to route the telegrams to other KNXnet/IP routers. Except for occasional network load conditions, forwarding KNX telegrams to the IP network is not critical as the IP network speed is at least 1000 times higher than that of the KNX subnetwork. Because of the capabilities of the KNX TP subnetwork the maximum number of telegrams forwarded from a KNX subnetwork to KNXnet/IP Routing Multicast Address is limited to 50 telegrams per second.

Under a more general system performance perspective it was decided to limit the transmission rate of KNX IP devices to 50 ROUTING_INDICATION datagrams per second.

(B) Transmission across the IP network

KNXnet/IP Routing is based on IP multicast addressing. Under certain network configurations or load conditions of devices or the network, datagrams in an IP network may be lost or even discarded by network components (switches, routers). If KNX IP datagrams are lost in an IP network this cannot be detected by the sender because multicast datagrams are not acknowledged.

Different transmission rates (10 Mbit/s, 100 Mbit/s, 1000 Mbit/s) may lead to traffic conditions in network routers and switches that can only be addressed by existing network methods (e.g. 8802.3x). If these network traffic conditions cannot be addressed then potentially KNX IP datagrams may be lost.

The KNXnet/IP Routing protocol itself has no influence on the performance or behavior of network components. Thus these are out of scope with respect to KNX IP.

Yet, there are measures that can be taken by the system integrator.

Any communication with supervisory systems (including visualizations) or ETS should use KNXnet/IP Tunneling, which is an acknowledged point-to-point connection. This excludes loss of datagrams due to the network. KNXnet/IP routers by default support KNXnet/IP Tunneling. Although KNXnet/IP Tunneling is not required for KNX IP devices it is strongly recommended to implement KNXnet/IP Tunneling in all KNX IP devices.

(C) Reception from the IP network

In general, the hardware design of a KNXnet/IP router or KNX IP device includes a physical 8802.3 (Ethernet) interface (often a specific Ethernet chip), a microprocessor, and in case of a KNXnet/IP router a KNX subnetwork interface like the TPUART.

If the Ethernet interface selected by a manufacturer is not suitable for the network data rate (e.g. 10 Mbit/s) this is an issue that is in the responsibility of the manufacturer. An issue with the Ethernet interface performance cannot be removed by whatever protocol measures would be applied.

(D) Interface within KNXnet/IP router or KNX IP device between Ethernet interface and microprocessor

Datagrams are exchanged between each of the three design units depicted in Figure 10 as datagrams are received from the IP network.

KNX IP devices receive datagrams via the IP network transceiver ("Ethernet chip"), which forwards them to the microprocessor. Depending on the hardware and software design of the interface between the network transceiver and the microprocessor the effective data transmission rate between these two parts inside a KNX IP device may be lower than the actual transmission rate on the communication network. Consequently, datagrams may queue up in the Ethernet chip before getting to the microprocessor when the data rate from the IP network exceeds the data rate to the microprocessor. This internal receiving transmission rate limitation may cause the loss of datagrams between network transceiver and microprocessor. This is a hardware, firmware, operating system, and/or application software issue that cannot be solved in general.

KNX Task Force IP came to the conclusion that using 8802.3x, which pauses communication between a switch and an end device (e.g. a KNX IP device) is not a viable solution.

Hence, a deficiency of the KNX device design cannot be removed by protocol measures unless the system performance as a whole is reduced. This would be against the intention using the IP network to enhance overall KNX system performance.

As product design is a manufacturer specific responsibility and because of existing devices in the market, KNX IP can only gradually achieve higher system performance by encouraging product improvements from today's performance levels to future higher performance levels.

Supporting such an approach is the definition of different test that intend to give an indication of the device performance with respect to receiving and processing datagrams.

To ensure a minimum system performance any KNX IP device or KNXnet/IP Router must be capable of receiving and processing the minimum number of 1 000 ROUTING_INDICATION datagrams per second on an assigned multicast address. Ideally, any KNX IP device or KNXnet/IP Router should be capable of receiving and processing at least 12 750 ROUTING_INDICATION datagrams [1)] per second on an assigned multicast address. This number enables KNX IP devices or KNXnet/IP Routers to receive and process datagrams

[1] IP datagram length: 64 octets.

sent by up to 255 KNX IP devices or KNXnet/IP Routers transmitting at a rate of 50 ROUTING_INDICATION datagrams per second.
A definition of performance classes was dropped leaving it up to the market to sort out performance evaluation most likely based on the tests defined for KNX IP.

(E1) Interface within KNXnet/IP router to KNX subnetwork

Because KNXnet/IP Routing datagrams are not acknowledged there is no indication on a datagram-by-datagram level about possible overload conditions in one or more KNXnet/IP routers. Because the data rate from the IP network exceeds the data rate to the KNX subnetwork and depending on the configuration a KNXnet/IP Router could receive more datagrams from the LAN than it can send to the KNX Subnetwork. This could lead to an overflow of the LAN-to-KNX queue and subsequent loss of one or more KNXnet/IP telegrams because they could not be transferred from the network buffer to the queue.
Flow control needs to be introduced for KNXnet/IP Routers and KNX IP devices to avoid the loss of datagrams due to overflowing queues in KNXnet/IP Routers and KNX IP devices.
Limiting the data rate of sending devices is not a solution for flow control as it does not guarantee that the incoming queue on a specific device (e.g. a KNXnet/IP Router) does not overflow because it is receiving telegrams to be sent onto the local Subnetwork from more than one sending device. The solution is for a receiving device to indicate to (all) other devices that its incoming queue is filling up and it may loose datagrams if they do not stop sending.
In this case the KNXnet/IP router needs to be able to signal other KNXnet/IP routers or KNX IP devices that its internal buffer to the KNX subnetwork is about to overflow. This signal has been introduced as ROUTING_BUSY, which is sent to the KNXnet/IP Routing Multicast Address.
The buffer overflow warning indication consists of a fixed length data field of six octets. It is used to indicate that the IP receive buffer has filled up to a point where the buffered incoming messages may take at least 100 ms to be sent to the KNX Subnetwork. The structure of the buffer overflow warning indication frame is shown in Figure 11.

KNX IP – using IP networks as KNX medium

```
KNXnet/IP header
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   HEADER_SIZE_10              |   KNXNETIP_VERSION            |
|   (06h)                       |   (10h)                       |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   ROUTING_BUSY                                                |
|   (0532h)                                                     |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   HEADER_SIZE_10 + 6                                          |
|                                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
BusyInfo
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   Structure Length            |   DeviceState                 |
|   (1 octet)                   |   (1 octet)                   |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|   ROUTING_BUSY_WAIT_TIME (in milliseconds)                    |
|   (2 octets)                                                  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   ROUTING_BUSY_CONTROL_FIELD                                  |
|   (2 octets)                                                  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 11 – ROUTING_BUSY frame binary format

If the incoming queue (e.g. KNX IP to KNX TP1) of a KNXnet/IP Router or KNX IP device exceeds the number of datagrams that can be processed (e.g. sent to the local KNX Subnetwork by a KNXnet/IP Router) within a period of $T_{PROCESS}$ then this device SHALL send a ROUTING_BUSY frame with a wait time t_w. The default value for $T_{PROCESS}$ SHOULD be 100 ms and MAY be greater than 100 ms. t_w SHOULD resemble the time required to empty the incoming queue.

The ROUTING_BUSY_WAIT_TIME contains the time that the receiving KNX IP device or KNXnet/IP router shall stop sending to the multicast address.

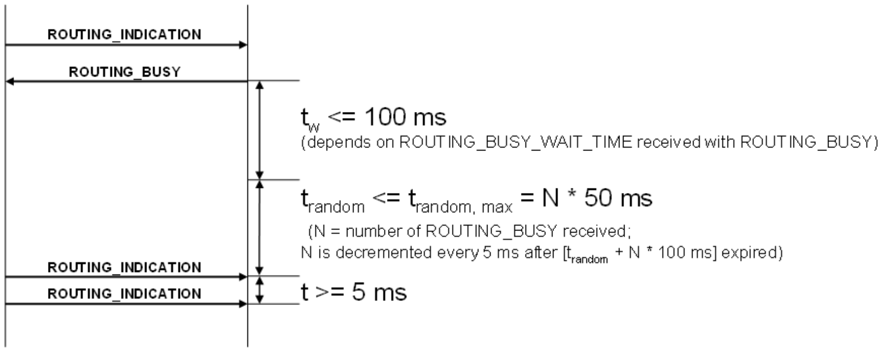

ROUTING_INDICATION
ROUTING_BUSY
t_w <= 100 ms
(depends on ROUTING_BUSY_WAIT_TIME received with ROUTING_BUSY)
t_{random} <= $t_{random, max}$ = N * 50 ms
(N = number of ROUTING_BUSY received;
N is decremented every 5 ms after [t_{random} + N * 100 ms] expired)
ROUTING_INDICATION
ROUTING_INDICATION
t >= 5 ms

Figure 12 – ROUTING_BUSY timing

ROUTING_BUSY provides time to empty the queue from KNX IP to a KNX subnetwork and a random restart of transmissions to avoid a flooding effect after the wait time t_w has expired. Typically setting the wait time t_w to 100 ms allows sending five messages to a KNX TP subnetwork. After sending of ROUTING_INDICATION messages resumes, a KNX IP device or KNXnet/IP router may send another ROUTING_BUSY if its queue has not been emptied below its threshold for sending ROUTING_BUSY. The number of ROUTING_BUSY received is then incremented leading to a longer random wait time for transmissions to resume. The intention is to dynamically scale the random wait time with the number of senders and the message traffic. The ROUTING_BUSY counter is decremented every 5 ms once the time [t_{random} + N * 100 ms] has passed without new ROUTING_BUSY messages.

The ROUTING_BUSY_CONTROL_FIELD by default is set to 0000h, which means that any KNX IP device or KNXnet/IP router must follow this request. If the ROUTING_BUSY frame contains a routing busy control field value not equal to 0000h then any device that does not interpret this routing busy control field SHALL stop sending for the time t_w (busy wait time).

What is the purpose of the ROUTING_BUSY_CONTOL_FIELD? In case the overflow of an internal buffer to the KNX Subnetwork of the KNX IP device or KNXnet/IP router sending the ROUTING_BUSY message is just caused by one or a few other devices sending ROUTING_INDICATION messages, stopping all ROUTING_INDICATION messaging unnecessarily reduces the overall system performance. KNX Task Force IP discussed several options for solving this situation. One possible solution is to set the value of this field to the individual address of the sender of the ROUTING_INDICATION message that led to the buffer overflow. A single device or line flooding the network would be slowed down but all other participants on KNX IP could still communicate. At this point, the routing busy control field is reserved for future use.

ROUTING_BUSY is intended to take care of potential datagram losses due to temporary datagram rate differences between the IP network and a KNX subnetwork. Permanent or frequent datagram flooding of a KNXnet/IP router is a matter of system configuration, which is the responsibility of the system integrator. In particular, this requires applying and downloading filter tables into KNXnet/IP routers.

ROUTING_BUSY does not take care of product performance issues due to product design limitations with regards to data exchange between network transceiver (Ethernet chip) and microprocessor. This is the responsibility of the manufacturer of a KNXnet/IP router or KNX IP device.

(E2) Interface within KNX IP device between KNX IP communication stack and application

The interface within a KNX IP device between the KNX IP communication stack and the application may be prone to similar performance issues as described under (E1) for KNXnet/IP routers. The solution for this situation is the same i.e. ROUTING_BUSY.

In general, it is expected that this case is a rare exception that manufacturers would endeavor to avoid.

Design recommendations

Technical University of Vienna provided KNX Task Force IP with a tool for simulating KNX IP system performance. This tool was used to simulate how the busy wait time, the size of the buffer for messages from KNX IP to KNX Subnetwork, the number of KNX IP devices, the number of ROUTING_INDICATION messages per second, and the threshold for sending ROUTING_BUSY influence the KNX IP system performance.

The simulations revealed valuable insights. A key factor for not loosing datagrams is the size of the buffer for messages from KNX IP to KNX Subnetwork in conjunction with the threshold for sending ROUTING_BUSY. For a KNX IP system with up to 255 devices sending a maximum of 50 messages per second the recommendations are:

- The threshold for sending a ROUTING_BUSY frame with the individual address from the last ROUTING_INDICATION frame SHOULD be set at five messages in the incoming queue.
- The threshold for sending a ROUTING_BUSY frame to all KNX/IP devices and KNXnet/IP Routers SHOULD be set at ten messages in the incoming queue.
- The incoming queue SHOULD be able to hold at least 30 messages.

Based on the simulations, KNX IP can be used reliably at least for up to 255 KNX IP devices and KNXnet/IP routers if all KNX IP devices and KNXnet/IP routers

- are capable of receiving and processing at least 12 750 ROUTING_INDICATION datagrams per second on an assigned multicast address,
- implement ROUTING_BUSY, and
- follow the recommendations for sending ROUTING_BUSY and for the buffer size.

Conclusions and Outlook

Following these few design objectives

- maintain the simplicity and scalability of the KNX system
- use management procedures already supported by ETS
- enable reusing existing stack implementations whereever possible
- add capabilities that enhance the KNX system for advanced applications

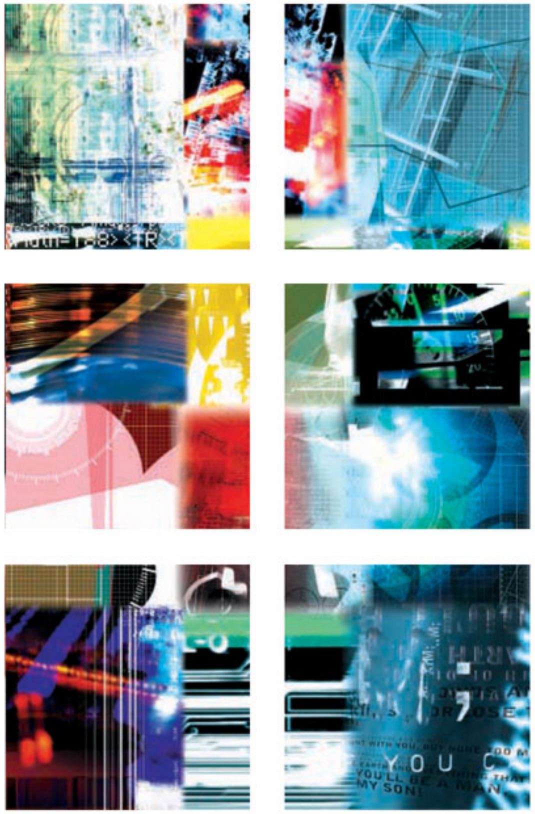
KNX Task Force IP defined KNX IP starting from the existing KNXnet/IP Routing protocol.

The influence of IP network performance on the KNX IP network performance and the KNX IP device performance may be dependent on the transmission rate, on network components (switches, routers), or on traffic on the network shared with other users (e.g.office applications).

Despite these potentially disturbing influences, KNX IP can be run reliably if a few design recommendations and configuration principles are followed.

The current definition allows for future enhancements as experience with KNX IP grows.

KNX IP further expands the foundation for active participation of KNX in environments increasingly driven by the Internet Protocol.



www.knx.org

Congestion Analysis in KNXnet/IP

Salvatore Cavalieri and Giovanni Cutuli

University of Catania, Faculty of Engineering,
Department of Computer Science and
Telecommunications Engineering
Viale A.Doria, 6 95125 Catania (ITALY)

Aim of the paper

- **The paper deals with the analysis of congestion in KNXnet/IP Router**

- **The analysis is aimed to point out:**
  - the information flow scenario which may cause congestion
  - the evaluation of the impact of congestion on the information flow, in terms of number of telegrams lost.
  - the impact of the forwarding rules of the KNXnet/IP Router on the congestion

Forwarding Rules in KNXnet/IP Router

- The KNXnet/IP Router may support two forwarding policies in case two or more telegrams are waiting in the incoming queue.

- Priority/FIFO mode foresees that the telegrams with the highest priority shall be routed first.

- Normal FIFO mode foresees the transmission of the telegram that was queued first, regardless of the KNX priority.

- Implementation of Priority/FIFO mode is not mandatory, so a KNXnet/IP Router must guarantee at least the FIFO mode functionality.

KNXnet/IP Model

- Evaluation of the impact of the congestion on the KNX communication system, has been realised through the definition of a model of the KNXnet/IP system and its simulation.

- Current literature offers many frameworks able to offer a set of ready to use useful objects and tools for creating a custom model and to simulate it.
  - Among them, OMNeT++ framework has been chosen

KNXnet/IP Model

- Different modules linked to the main KNXnet/IP features, have been defined inside OMNeT++.
- KNX Device. It models all the five layers belonging to the KNX stack.
- KNX subnetwork. It models a TP1 KNX bus to which several KNX devices can be attached.
- IP network. Transmission rate inside IP has been fixed at 100 Mbit/s.
  - KNX subnetworks are connected to the IP backbone

KNXnet/IP Model

- KNXnet/IP Router. Two different interchangeable objects have been defined for the KNXnet/IP Router:
  - The first foresees the use of the normal FIFO mode for realising the forwarding rule of the router;
    - a single queue has been modelled to collect telegrams coming from the IP backbone and routed to the KNX subnetwork according to the only arrival time.
  - The other KNXnet/IP Router model foresees telegrams priority-based forwarding rule (according to the priority/FIFO mode)
    - priority management of the telegrams incoming from the IP side has been realised using three separate FIFO queues, one for each of the three Data Link layer priorities.

KNXnet/IP Model

- The size of the queue used to collect telegrams coming from KNX subnetworks towards IP network, has been set to 2 that is the minimum requirement for a KNXnet/IP Router.

- The size of the queue/s used to collect telegrams coming from IP network toward KNX subnetwork is a configurable parameter and different simulation scenarios have been simulated with different KNX queue length values.

KNXnet/IP Model

- A one-to-many data exchange has been assumed to model the information flow between KNX devices connected to different KNX subnetworks and exchanging data through IP backbone.

- A KNX device placed into a KNX subnetwork sends information to KNX devices placed in the existing KNX subnetworks, through IP.

- Transmissions are realised using the Application layer “group objects”, considering a Group address for each KNX device sender and the associated KNX device receivers.

KNXnet/IP Model

- Another hypothesis done in the model is related to the kind of information transmitted by each single KNX device;
- it has been assumed that each information sent by a KNX device is a value of variables periodically updated.
- In particular, for each KNX device sender, a process producing periodic information has been considered;
- The length of each information produced may be configured in the model.

KNXnet/IP Model

- Priorities (urgent, normal and low) have been considered for the KNX sending devices.

- For each priority, it is possible to set the number of variables whose values must be transmitted using high priority telegrams (NH), the number of variables whose values must use the normal priority (NM) and that using the low priority (NL).

- All the variables featuring the same priority are updated (i.e. a new value is produced) in a periodic fashion with the same period;
  - so three periods have been defined for each priority: TH, TM and TL.

KNXnet/IP Model

- Other behaviours have been modelled.
- KNX traffic generators have been modelled; they are optionally located in each KNX subnetworks and send messages in the subnewtork they belong to, adding additional traffic in order to reduce the available bandwidth;
  - it is possible to enable or disable the presence of KNX traffic generators in the simulation model.
- IP traffic generator has been defined. A couple of sender/receiver may generate IP traffic in order to increase information flow in the IP network;
  - the default configuration foresees the presence of the IP traffic generator.

Performance Evaluation

- **Performance evaluation has been finalised to analyse:**
  - the impact of the congestion on the information flow exchanged by KNX devices through the IP network, and
  - the impact of the forwarding policies used inside KNXnet/IP Router on the congestion itself.

- **Performance evaluation has been based on the measure of percentage of telegrams lost, evaluated taking into account:**
  - the number of telegram incoming from IP network to the KNXnet/IP Router, that cannot be enqueued, as the incoming queue is full;
  - the number of telegram already enqueued and waiting to be transmitted into a KNX subnetwork, which cannot be transmitted (e.g. due to high traffic in the KNX subnetwork) and are deleted from the queue.

Performance Evaluation

- **The following hypotheses have been considered in the performance evaluation**
- **The KNX traffic generator has been configured in such a way to produce an additional amount of traffic equal to the 65% of the available throughput; this has been achieved by sending 6150 bits per second on the KNX subnetwork.**
- **3 KNX subnetworks have been considered.**
- **The number of bits used to code each variable value transmitted by each KNX device, has been fixed to 16 bits.**
- **For each KNX sending device, NH=3, NM=5 and NL=20.**
- **It was assumed that TH<TM<TL, setting TM=TL/2 and TH=TL/3.**
  - different values of TL have been considered during the performance evaluations

Results

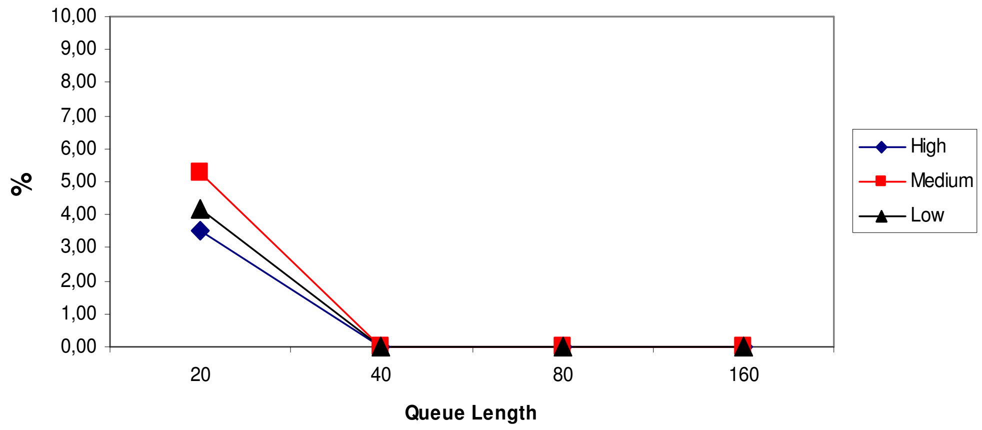


Figure 2. normal FIFO mode

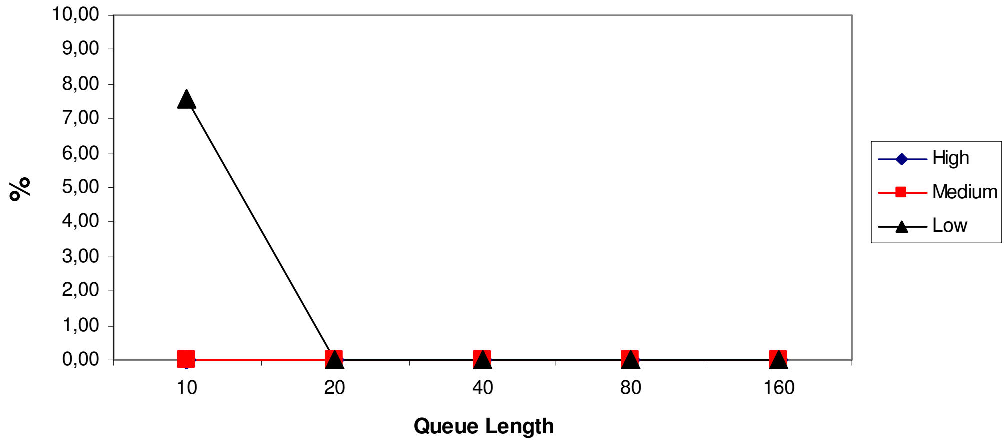


Figure 3. priority/FIFO mode

- TL=15s, TM=TL/2 and TH=TL/3
- The queue length in the abscissa of Figure 3 refers to the size of each of the three queues considered in the priority/FIFO mode scenario.
- No congestion
- The low percentages of telegrams lost are mainly due to a wrong dimension of the incoming queue.
- The adoption of the priority/FIFO mode allows to avoid loss of high and medium priority telegrams, also when small queue sizes are considered.

Results

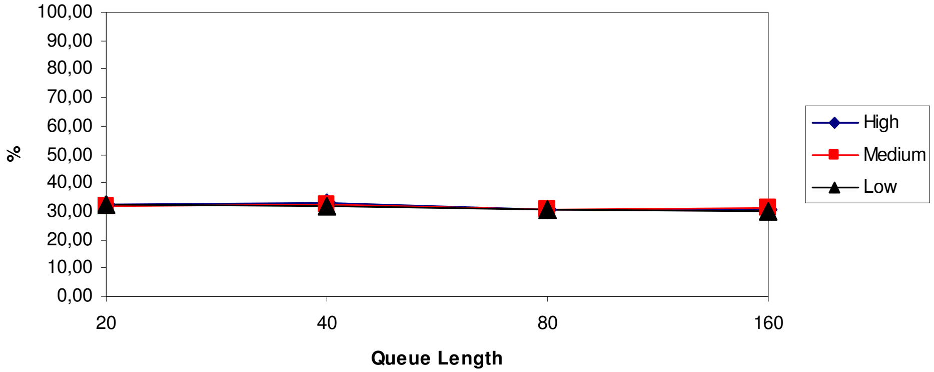


Figure 4. normal FIFO mode

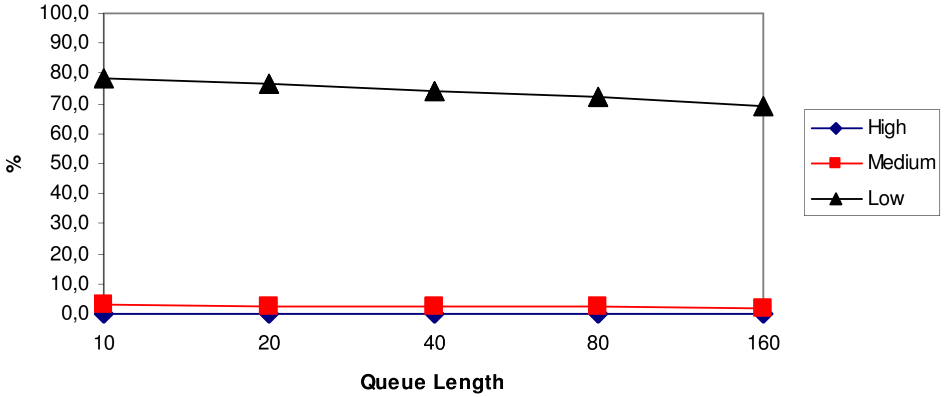
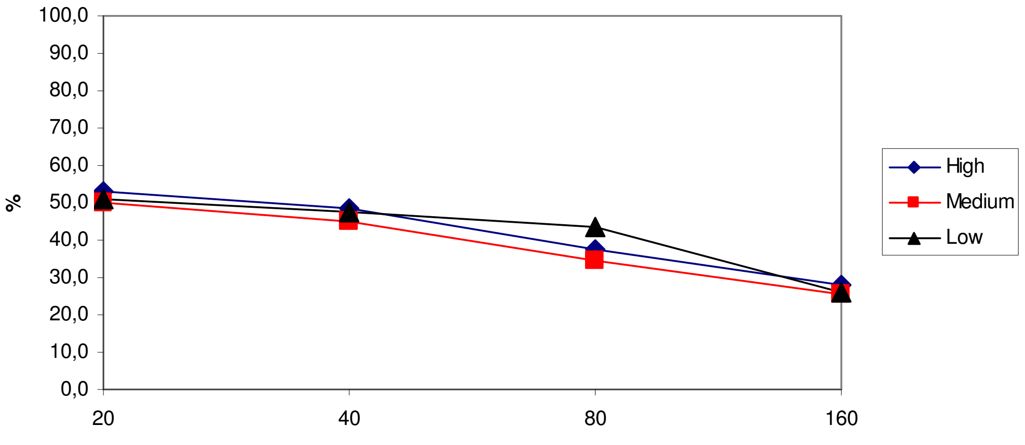


Figure 5. priority/FIFO mode

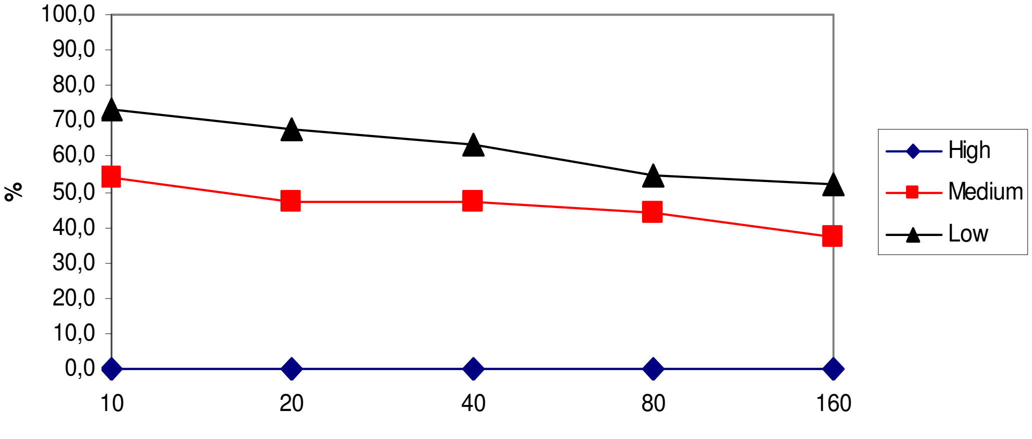
- TL=0.9s, TM=TL/2 and TH=TL/3,
- Now the low percentages of telegrams lost are due to a congestion in the KNXnet/IP Router.
- The adoption of the priority/FIFO mode allows to avoid loss of high and medium priority telegrams;
- the low priority telegrams are penalised as the percentage of telegram lost increases.

Results



Queue Length

Figure 6. normal FIFO mode



Queue Length

Figure 7. priority/FIFO mode

- TL=0.6s, TM=TL/2 and TH=TL/3,
- In this case the frequency of transmission of telegrams increases, stressing the congestion in the KNXnet/IP Router.
- it’s clear that now the congestion is higher and that the adoption of the priority/FIFO mode is able to limit only loss of high priority telegrams;
- the medium and low priority telegrams are still penalised.

Results

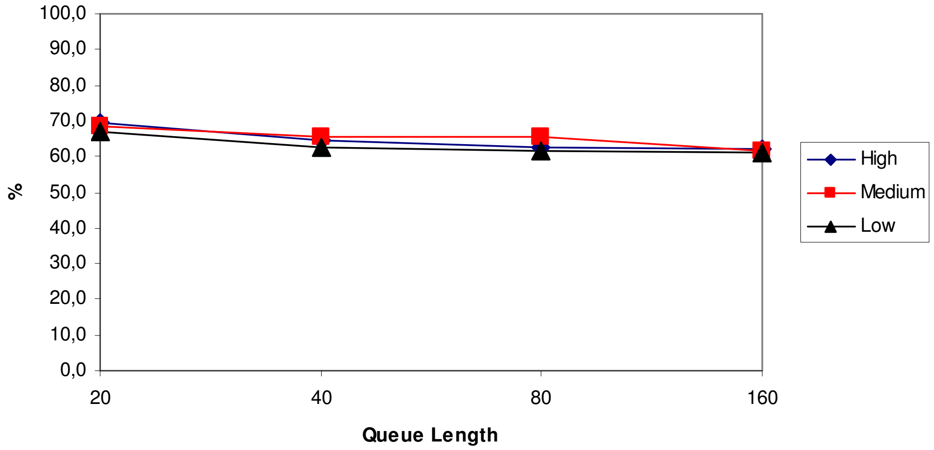


Figure 8. normal FIFO mode

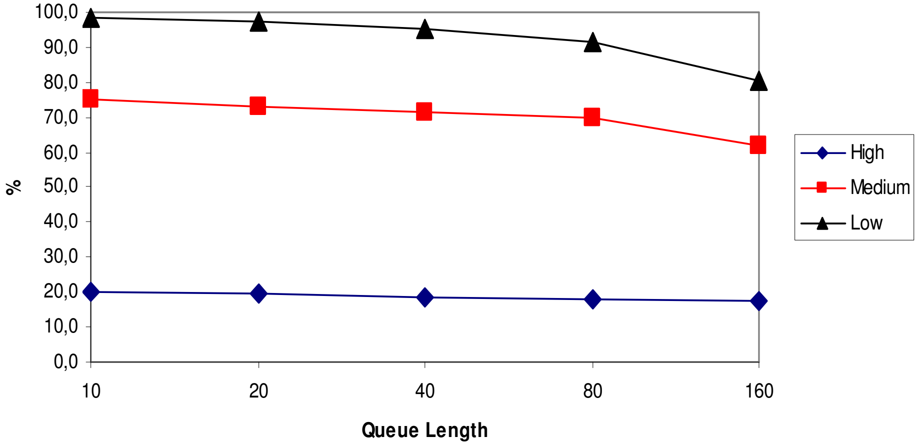


Figure 9. a priority/FIFO mode

- TL=0.6s, TM=TL/2 and TH=TL/3,
- The KNX additional traffic generator has been considered in each KNX subnetwork, in order to reduce the available bandwidth.
- Congestion of the KNXnet/IP Router depends **also** on the average queueing time in the KNXnet/IP Router which increases when the available bandwidth in the KNX is reduced;
- in other words, congestion in a KNXnet/IP Router may increase when the access to the KNX subnetwork is more difficult due to huge traffic.

Final remarks

- The results presented in the paper have confirmed a well known problem: congestion of the KNXnet/IP Router.
- Although the congestion problem is well known, important remarks may be done on the basis of the results achieved
- First: the forwarding rules of the KNXnet/IP Router have a strong impact on the congestion. The use of priority/FIFO mode allows a drastic reduction of high and medium priority telegrams loss. On the basis of the results achieved, it’s highly recommended to adopt an efficient forwarding rule in the implementation of KNXnet/IP Router.
- Second: it seems that congestion isn’t a sporadic event as the traffic conditions considered in the performance evaluation wasn’t particularly critical. This points out the real need of the definition of efficient congestion flow control procedures inside the KNXnet/IP standard.
  - A work is currently being performed by University of Catania (Prof.S.Cavalieri) and CNR (Dott.M.Aliberti), about definition of congestion control strategies inside KNXnet/IP

Congestion Analysis in KNXnet/IP

Salvatore Cavalieri and Giovanni Cutuli

University of Catania, Faculty of Engineering
Department of Computer Science and Telecommunications Engineering
Viale A.Doria, 6 – 95125 Catania (Italy)
E-mails: Salvatore.Cavalieri@diit.unict.it, Giovanni.Cutuli@diit.unict.it
Tel.: +39 095 738 2362, Fax:+39 095 738 2397

Abstract **– Very recently, KNXnet/IP system has been defined; it allows integration of different KNX subnetworks through an IP network, used as fast backbone. Integration is realised by a particular device called KNXnet/IP Router. Due to the very limited transmission speeds foreseen for the KNX networks, the KNXnet/IP router may represent a bottleneck in routing telegrams coming from the IP network whose destination is a KNX device located in the KNX subnetwork. The aim of the paper is to analyse the effect of congestion of KNXnet/IP Routers on the information flow between KNX devices attached to different KNX subnetworks through an IP backbone. Different communication scenarios will be considered, in order to clearly understand the behaviour of the KNX/IP integration through KNXnet/IP Routers.**

I. Introduction

Since June 2009, KNX communication system has been integrated into IP environment, having defined the KNXnet/IP specifications [1].

A KNXnet/IP system allows integration of different KNX subnetworks through an IP network, used as very fast backbone. Integration is realised by a particular device called KNXnet/IP Router, that provides network connection between a KNX network and an IP network. The router has two interfaces and two addresses compliant with the KNX and IP networks, respectively.

KNXnet/IP Router receiving telegrams from the IP network, must forward to the KNX subnetwork attached, only the telegrams relevant to this subnetwork; this is done on the basis of a particular Group Address contained in the incoming telegram. The KNXnet/IP Router shall provide two queues, one for the incoming telegrams from the IP network and the other for the telegrams sent from the KNX subnetwork towards other KNX subnetworks, through IP network. Two forwarding rules may be supported by a Router. The first is mandatory and is called normal FIFO mode; the telegrams enqueued will be transmitted in an order respecting the arrival time. The second mode, called priority/FIFO, foresees that telegrams with the highest KNX priority will be routed firstly; this second mode is optional and isn't generally supported by the current KNXnet/IP Routers available on the market.

Due to the very limited transmission speeds foreseen for the KNX networks, the KNXnet/IP Router may represent a bottleneck in routing telegrams coming from the IP network whose destination is a KNX device located in the KNX subnetwork. If the KNXnet/IP Router receives more telegrams from IP network than it is able to sell off, the internal queue can fill up to the maximum size and consequently all the other received telegrams will be lost. That condition is relevant to a congestion and should be avoided by a congestion flow control mechanism. The only solution foreseen by the KNX to realise congestion flow control is mainly based on a "stop/go" mechanism which may block the transmission on the IP medium, on telegrams loss conditions. In particular, if a KNXnet/IP Router realises that a congestion occurs, it communicates to all the sending devices that the queue is filling up, in order each sending device could reduce the amount of information flow to be transmitted, leading to a progressive reduction of the congestion condition. The use of this mechanism is foreseen only for occasional and temporary telegrams loss conditions; this because it's very drastic, as each sender cannot know the KNXnet/IP Router involved in the congestion (due to the multicast transmission from the congested KNXnet/IP Router), and so it has to reduce or suspend all its transmissions, also those addressed to other not congested KNXnet/IP Routers.

On the basis of what said, it seems very important to evaluate different things. First of all, it's important to understand if congestion is a sporadic event or it may happen frequently; secondly an investigation about the information flow scenario which may cause congestion seems useful. Then, evaluation of the impact of congestion on the information flow should be evaluated in terms of number of telegrams lost. Finally, the impact of the forwarding rules of the KNXnet/IP Router on the congestion should be evaluated. This analysis may allow to point out:

- The real need of efficient congestion flow control procedures in the KNXnet/IP standard, in the case the analysis points out a very huge effect of congestion on the number of telegrams lost and on the information flow between control devices;
- Recommendation to adopt an efficient forwarding rule in each KNXnet/IP Router, if the analysis points out that a particular forwarding rule may limit the impact of congestion in terms of loss of telegrams.

According to what said, the aim of the paper is to analyse the effect of congestion of KNXnet/IP Routers on the information flow between KNX devices attached to different KNX segments connected to an IP network; evaluation will be realised in terms of percentage of telegrams lost. Different communication scenarios will be considered, in order to clearly understand the behaviour of the KNX/IP integration through KNXnet/IP Routers.

The paper will be organised into a brief overview of the KNX and KNX/IP specifications, followed by the description of the model adopted for the congestion analysis; finally the main results achieved by simulation of the model will be pointed out. Final remarks will conclude the paper.

II. KNX AND KNXNET/IP OVERVIEW

The aim of this section is to give basic information about KNX specifications; this information will be useful to understand KNX modelling and simulation presented in the paper. The overview on KNX specifications will be split into two parts: the description of the original KNX communication system and the current KNXnet/IP extension.

A. KNX Communication System

KNX stack foresees the Physical, Data Link, Network, Transport and Application layers; session and presentation layer are empty and transparent to the neighbour layers.

Several physical media have been foreseen for the KNX Physical layer [2]: powerline [3], radio frequency [4] and twisted pair (TP0 and TP1) [5][6]. In this paper only the twisted pair 1, TP1, will be considered.

According to KNX TP1 specifications [6], devices are connected to a physical Segment, which may feature a linear, star, tree or a mixed topology. Two different TP1 specifications have been defined: TP1 64 and TP1 256; the main difference is that up to 64 (TP1-64) or 256 (TP1-256) devices can be connected to a physical Segment. In any case, the maximum distance between two devices in a physical Segment is 700 m and the maximum cable length of a physical Segment is 1000 m. According to the KNX TP1 coding, a logical ‘1’ represents the idle state of the bus, meaning that the transmitter of a MAU is disabled during transmission of ‘1’. Although each device must be sure that the bus was free before starting transmission, it can happen that two or more devices transmit simultaneously; in this case, transmission of both logical ‘0’ and logical ‘1’ at the same time results in a logical ‘0’. Simultaneous transmission of logical ‘0’ from several devices results in a signal that is nearly the same as that of a single transmitting device, because signal is coded in baseband. If a sending device detects that an own logical ‘1’ is overwritten by another logical ‘0’, transmission has to be disabled after this bit, however the receivers of both devices are still in progress.

KNX Medium Access Control (MAC) strictly depends on the KNX physical medium. Considering the TP1 KNX specification, the MAC is based on CSMA/CA, Carrier Sense Multiple Access with Collision Avoidance [6]. CSMA/CA means that collisions shall be resolved within a bit time. Before a device may start the transmission of a request Frame it has to wait for at least 50 bit times line idle since the last bit of the preceding Data Link layer message cycle (a Data Link layer message cycle shall always consist of a Data Link layer request Frame and the subsequent Data Link layer acknowledgment - or a subsequent Data Link layer response Frame [7]). If a device starts transmission and the Physical layer detects a collision (i.e. the transmission of a Frame with higher priority sent by another device is in progress), transmission is immediately stopped. All Frame parts that have already been sent shall be interpreted as being part of the higher priority Frame relevant to the transmission in progress. After line busy detection and after a collision, the device shall wait until the end of the message cycle in progress and shall make another attempt to transmit the data link request PDU after 50 bit times or more line idle time.

Four message priority levels are supported at the Data Link layer [7]. For the process communication, three priorities are available: urgent, normal and low. Configuration and/or management procedures are realised using a higher priority level, above process communication.

Network layer in KNX standard [8] is compliant with the definition of the ISO/OSI model network layer (ISO 7498). Transport layer in KNX standard [9] provides data transmission over different communication modes; in particular KNX transport layer provides four different communication modes: point-to-point connection-less (multicast), point-to-all-points connection-less (broadcast), point-to-point connection-less and point-to-point connection-oriented.

According to KNX Application layer [10], particular objects, called datapoints, allow communication between devices; they are abstract objects that gives access to internal object (Application Interface Object) in a device, which can store any type of information and can be writeable and readable by a generic device. KNX Application layer services fall into two distinct groups: device configuration/diagnosis and exchange of process values/data. Device configuration/diagnosis services use either the connection oriented (reliable) or the connectionless (unreliable) point-to-point communication service provided by the transport layer. Process communication follows a producer/consumer communication pattern. The communication endpoints of devices for this purpose are referred to as “group object datapoints” or “group objects”. Every group object represents a single input or output value (which may – but most often does not – consist of multiple fields that are transmitted simultaneously). Binding is done by assigning identical “group addresses” to the group objects whose values are to be kept in sync. However, multiple outputs can be bound to a single input, replacing this model of coupled states with generic message channels.

The Application layer services for process communication consist of an unconfirmed service to announce the update of a shared value (A GroupValue Write) and a confirmed service to ask for the current state of a shared value (A GroupValue Read). The former is the mainstay of KNX communication. It is not only used for reporting that the value of a point has changed, but also for all kinds of state change or action requests.

B. KNXnet/IP

The KNXnet/IP protocol has been developed by the KNX Task Force IP, with the following main objectives: maintaining the simplicity and scalability of the KNX system, allowing reuse of existing stack implementations as much as possible and adding capabilities of the KNX systems for advanced applications.

A KNXnet/IP system at least contains: an IP network, that is used as backbone to carry information exchanged by KNX devices, one KNX subnetwork, that may be one of the communication medium foreseen in KNX specifications and a KNXnet/IP Router, that provides network connection between the KNX subnetwork and IP network. Figure 1 shows a KNXnet/IP system, made up by four KNX subnetworks.

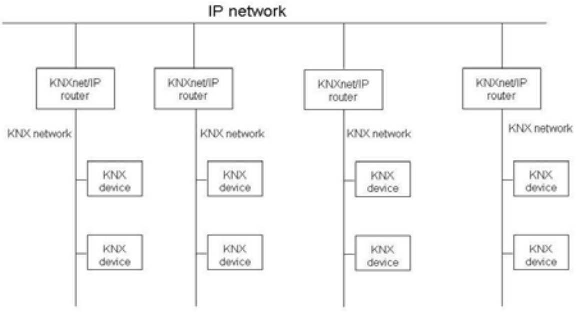


Figure 1. An example of KNXnet/IP system.

Both the IP communication modes, unicast and multicast, are used by a KNXnet/IP Router. Unicast communication mode is used only for KNXnet/IP Tunnelling and Management operations where a direct connection (and confirmation) is needed. In unicast communication mode each KNXnet/IP Router would have been configured to send data to a pre-defined list of devices.

The other IP addressing method used by the KNXnet/IP Router is multicast; a KNXnet/IP Router doesn’t establish a communication channels between each other. A KNXnet/IP Router always transmit data to all KNXnet/IP Routers that belong to the same IP multicast address on the same network. It is possible to route data abroad the network or other network through normal standard IP routers.

All KNX telegrams received by a KNXnet/IP Router must be processed and filtered on the basis of the Group Address contained in the telegram. On the basis of the results of filtering operation, the telegram will be or not forwarded into the KNX network attached to the KNXnet/IP Router.

A KNXnet/IP Router shall provide two telegram queues. Both of them allow queuing at least two telegrams for the KNX side and at least two telegrams for the IP side; in fact, any telegram cannot be instantly forwarded but it must be queued in the respective telegram queue if it is possible.

The KNXnet/IP Router may support two forwarding policy in case two or more telegram is waiting in the telegram queue. The first mode, called priority/FIFO, foreseen that the telegrams with the highest priority shall be routed first even if there are other queued telegrams with lower priority. The second mode, called normal FIFO, foreseen the transmission of the telegram that was queued first, regardless of the KNX priority. Priority/FIFO mode is not a compulsory functionality, so a KNXnet/IP Router must guarantee at least the FIFO mode functionality.

The KNXnet/IP Router discards all telegrams in the queue if bus connection failure has been detected. Telegrams loss is even possible in case of queue overflow; in that case, a congestion occurs and the KNXnet/IP Router will discard the telegrams received that couldn’t be enqueued.

III. KNXNET/IP MODEL

Evaluation of the impact of the congestion on the KNX communication system, has been realised through the definition of a model of the KNXnet/IP system and its simulation. Use of this approach allows both evaluation of the current KNX/IP system and exploration of different solutions conceived to improve the performances.

Current literature offers many frameworks able to offer a set of ready to use useful objects and tools for creating a custom model and to simulate it. Among them, OMNeT++ framework has been chosen [11].

A. Description of the Model

Different modules linked to the main KNXnet/IP features, have been defined inside OMNeT++.

All the five layers belonging to the KNX stack, have been incorporated into a compound module that represents a KNX device.

Another module, representing a KNX subnetwork, has been defined; its made up by a KNX bus object to which several KNX devices (up to the maximum number allowed) can be attached. The bus object represents KNX TP1 communication system and features several properties linked to the following KNX TP1 parameters:

- the maximum distance between two devices; the default value has been set to 700 meters
- the bus propagation delay; the default value has been set to 12 μs
- a single bit duration also known as “bit time”; the default value has been set to 104 μs
- transmission rate; the default value has been set to 9600 bits/sec.

The IP network used in the simulation model is a ready to use channel that is possible to find within the INET Framework used by OMNeT++ [11]. Transmission rate inside IP has been fixed at 100 Mbit/s.

KNX subnetworks are connected to the IP backbone through KNXnet/IP Routers, so a relevant model has been defined inside OMNeT++. In particular two different interchangeable objects have been defined for the KNXnet/IP Router. The first foresees the use of the normal FIFO mode for realising the forwarding rule of the router; a single queue has been modelled to collect telegrams coming from the IP backbone and routed to the KNX subnetwork according to the only arrival time. The other KNXnet/IP Router model foresees telegrams priority-based forwarding rule (according to the priority/FIFO mode); priority management of the telegrams incoming from the IP side, has been realised using three separate queues, one for each of the three Data Link layer priorities. Each queue collects telegrams coming from the IP backbone that need to be routed to the KNX subnetwork, according to the priority of the telegrams; the telegrams are dequeued and transmitted to the KNX subnetwork, according to the priority, starting from the higher priority queue, and, for each priority queue, according to the arrival time. The size of the queue/s is a configurable parameter and different simulation scenarios have been simulated with different KNX queue length values. The size of the queue used to collect telegrams coming from KNX subnetworks towards IP network, has been set to 2 that is the minimum requirement for a KNXnet/IP Router, as standard describes [1].

Recalling that all the telegrams coming from the KNX subnetwork, whose destination is abroad the source subnetwork, are routed by the KNXnet/IP Router in multicast telegram transmission mode, it wasn't necessary to have KNX Network Layer and KNX Transport Layer implemented in the KNXnet/IP Router (resulting in a smaller and lighter simulation model); in fact, these layer are useful only during configuration phase when point to point connection oriented services are used and not during normal operation (when multicast telegram transmission mode is used, as said before).

Particular hypotheses have been assumed to model the information flow between KNX devices connected to different KNX subnetworks and exchanging data through IP backbone. A one-to-many data exchange has been considered; a KNX device placed into a KNX subnetwork sends information to KNX devices placed in all the KNX subnetworks; in particular, for each KNX device sender belonging to a KNX subnetwork, one KNX device receiver for each of all the KNX subnetworks (including that of the sender) have been associated.

Transmissions are realised using the Application layer "group objects", considering a Group address for each KNX device sender and the associated KNX device receiver.

Another hypothesis done in the model is related to the kind of information transmitted by each single KNX device; it has been assumed that each information is a value of variables periodically updated. In particular, for each KNX device sender, a process producing periodic information has been considered; an example may be a sensor sampling an analogue value and producing samples according to the sampling interval. The length of each information produced may be configured in the model; Payload is the name of the parameter used in the model to set the length of each information produced, in terms of bit. So, the whole KNX telegram length is due to the length of the Payload plus the following other fields: 8 bits for various control bits (including priority bits selection), 16 bits for the Source Address, 16 bits for the Destination Address, 8 bits to contain the whole frame length and the Address Type and 8 bits to contain checksum. Priorities (urgent, normal and low) have been considered for the KNX sending devices. In particular, for each of them, it is possible to set the number of variables whose values must be transmitted using high priority telegrams (N_H), the number of variables whose values must use the normal priority (N_M) and that using the low priority (N_L). It has been assumed that all the variables featuring the same priority are updated (i.e. a new value is produced) in a periodic fashion with the same period; so three periods have been defined for each priority: T_H, T_M and T_L.

Other behaviours have been modelled. Particular KNX devices, called KNX traffic generators, have been considered and modelled; they are optionally located in each KNX subnetworks and send messages in the subnewtork they belong to, adding additional traffic in order to reduce the available bandwidth; it is possible to enable or disable the presence of KNX traffic generators in the simulation model. The same concept can be applied to the IP backbone where it is possible to enable a couple of sender/receiver that generates IP traffic in order to increase information flow in the IP network; in this case the default configuration foresees the presence of a IP traffic generator.

IV. Performance Evaluation

Performance evaluation has been finalised to analyse the impact of the congestion on the information flow exchanged by KNX devices through the IP network, and the impact of the forwarding policy used inside KNXnet/IP Router on the congestion itself. As said in the paper, the default forwarding mode of the KNXnet/IP Router doesn't take into account the priority of the telegram; the performance evaluation mainly aims to quantify the advantages of using a priority-based forwarding rule inside the KNXnet/IP Router.

On the basis of what said, the performance evaluation will compare two scenarios, featured by the use of normal FIFO and priority/FIFO modes in the forwarding of telegrams from IP to KNX network. Performance evaluation has been based on the measure of percentage of telegrams lost; in particular the number of telegrams lost have been evaluated taking into account:

- the number of telegram incoming from IP network to the KNXnet/IP Router, that cannot be enqueued, as the incoming queue is full;
- the number of telegram already enqueued and waiting to be transmitted into a KNX subnetwork, which cannot be transmitted (e.g. due to high traffic in the KNX subnetwork) and are deleted from the queue.

In the following subsection, the details about the choices of parameters present in the model will be given; then the other subsection will point out the main results achieved.

A. Performance Evaluation Hypotheses

As explained in the previous section, the KNXnet/IP model defined by the authors, features several parameters which can be configured before simulation; in the following, the main parameters and the relevant values chosen will be pointed out.

Different sizes of the incoming queue from IP to KNX network have been considered, starting from the minimum value foreseen by the KNX standard, equal to 2 KNX telegrams. As the priority/FIFO mode has been modelled using three different queues, one for each priority (urgent, normal and low), the size of each queue has been varied during the congestion analysis carried on. According to what said, and in order to compare the two scenarios, it's important to recall that for each queue size considered for the normal FIFO mode, the equivalent queue size in the priority/FIFO mode is given by the sum of the three queue sizes.

The queue length related to the information flow from KNX subnetwork to IP has been fixed to 2 telegrams, as stated in the standard [1], for both the scenarios.

Each KNX subnetwork features an optional KNX traffic generator, used to evaluate the congestion in the KNXnet/IP Router when available bandwidth in each KNX subnetwork is reduced. The traffic generator defined in the model has been configured in such a way to produce an additional amount of traffic equal to the 65% of the available throughput; this has been achieved by sending 6150 bits per second on the KNX subnetwork.

The KNXnet/IP model foresees the presence of a configurable number of KNX subnetworks; in the performance evaluation carried on, only three subnetworks have been considered.

The Payload, i.e. the number of bits used to code each variable value, has been fixed to 16 bits.

Each KNX sending device located in a KNX subnetwork, features N_H=3, N_M=5 and N_L=20.

The performance evaluation shown in this paper, refers to a scenario featured by higher transmission frequency for the telegrams with higher priority; in particular it was assumed that $T_H<T_M<T_L$, setting $T_M=T_L/2$ and $T_H=T_L/3$. Different values of T_L have been considered during the performance evaluations, as pointed out in the following.

B. Performance Evaluation Results

Figures 2 and 3 show the percentage of telegrams lost in a KNXnet/IP Router considering normal FIFO and a priority/FIFO mode, respectively; for both of them T_L has been set to 15s. It's very important to remember that the queue length in the abscissa of Figure 3 refers to the size of each of the three queues considered in the priority/FIFO mode scenario. As can be seen the low percentages of telegrams lost are mainly due to a wrong dimension of the incoming queue. The adoption of the priority/FIFO mode allows to avoid loss of high and medium priority telegrams, also when small queue sizes are considered.

Figures 4 and 5 show the percentage of telegrams lost in a KNXnet/IP Router considering T_L=0.9s and taking into account again normal FIFO and priority/FIFO modes, respectively. As can be seen, now the low percentages of telegrams lost are not linked to a wrong dimension of the queue/queues, but are due to a congestion in the KNXnet/IP Router. In this scenario, the percentage of telegrams enqueued and not transmitted in the KNX subnetwork is 0; this means that the percentage of telegram lost is only related to the telegrams incoming from the IP network, that cannot be enqueued as the queue of the Router is full. Comparing Figure 4 and 5, the adoption of the priority/FIFO mode allows to avoid loss of high and medium priority telegrams; the low priority telegrams are penalised as the percentage of telegram lost increases.

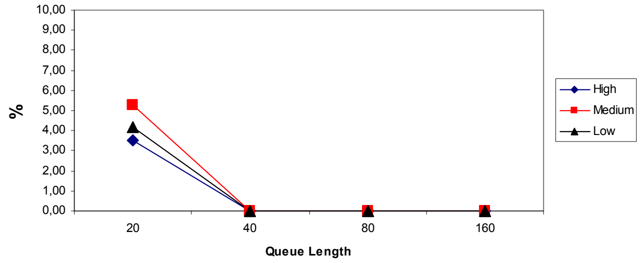


Figure 2. Percentage of telegrams lost, T_L=15s, $T_M=T_L/2$ and $T_H=T_L/3$, considering a normal FIFO mode

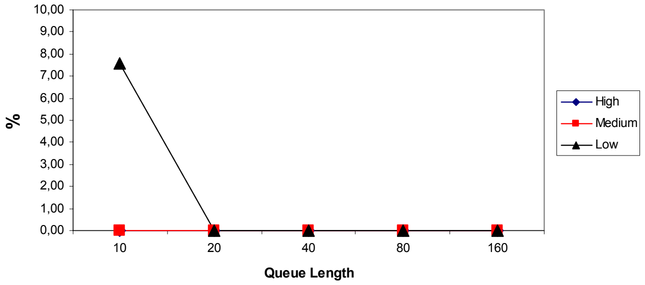


Figure 3. Percentage of telegrams lost, T_L=15s, $T_M=T_L/2$ and $T_H=T_L/3$, considering a priority/FIFO mode

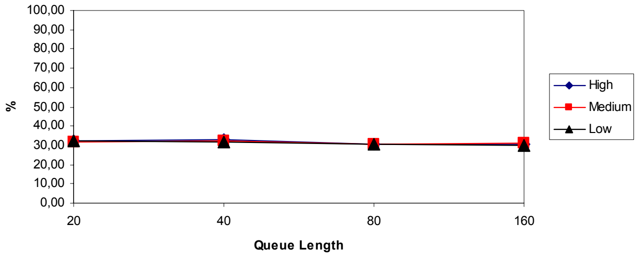


Figure 4. Percentage of telegrams lost, T_L=0.9s, $T_M=T_L/2$ and $T_H=T_L/3$, considering a normal FIFO mode

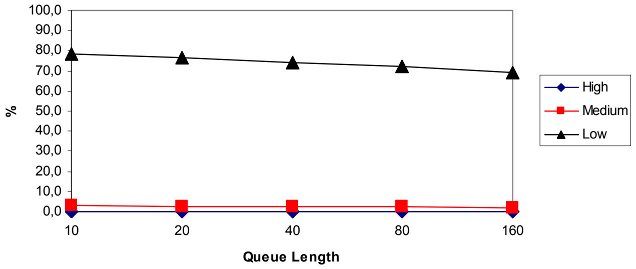


Figure 5. Percentage of telegrams lost, T_L=0.9s, $T_M=T_L/2$ and $T_H=T_L/3$, considering a priority/FIFO mode

Figures 6 and 7 show the percentage of telegrams lost in a KNXnet/IP Router considering T_L=0.6s; again, the figures refer to the normal FIFO and priority/FIFO modes, respectively. In this case the frequency of transmission of telegrams increases, stressing the congestion in the KNXnet/IP Router. Like the previous scenario, the percentage of telegrams enqueued and not transmitted in the KNX subnetwork is 0; this means that the percentage of telegrams lost is only related to the incoming telegrams from the IP network. Comparing Figure 6 and 7, it's clear that now the congestion is higher and that the adoption of the priority/FIFO mode is able to limit only loss of high priority telegrams; the medium and low priority telegrams are still penalised.

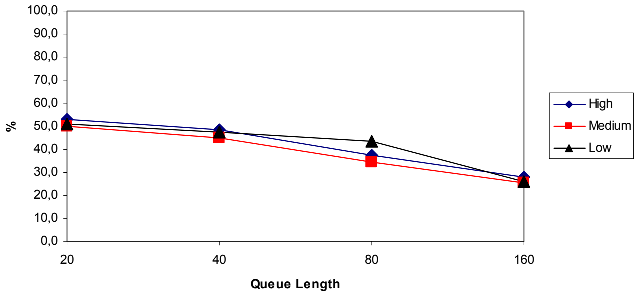


Figure 6. Percentage of telegrams lost, T_L=0.6s, T_M=T_L/2 and T_H=T_L/3, considering a normal FIFO mode

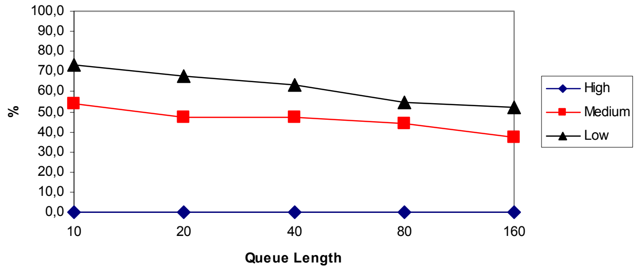


Figure 7. Percentage of telegrams lost, T_L=0.6s, T_M=T_L/2 and T_H=T_L/3, considering a priority/FIFO mode

The results presented until now pointed out that congestion of KNXnet/IP Router increases when the frequency of incoming telegrams from IP network increases; the telegrams are lost as the incoming queue is full. But there is another cause which could determine an increase in the number of telegrams lost incoming from IP network. Congestion of the KNXnet/IP Router may also depend on the time needed by the Router to dequeue each telegram stored in the queue and to transmit it to the KNX subnetwork; this time is linked to the KNX medium, and in particular to the available bandwidth. In fact when the number of telegrams sent inside each KNX subnetwork increases, the number of retry to transmit each single telegram is higher, and the time spent by each telegram inside the queue of a KNX device (including the KNXnet/IP Router) increases too. The average queueing time in the KNXnet/IP Router may increase when the available bandwidth in the KNX is reduced; in other words, congestion in a KNXnet/IP Router may increase when the access to the KNX subnetwork is more difficult due to huge traffic. On the basis of what said, evaluation of the congestion of KNXnet/IP Router has been done considering a scenario where a generic KNX subnetwork features a huge number of accesses. The KNX additional traffic generator has been considered in each KNX subnetwork, in order to simulate the increased number of accesses to the medium, as explained before.

Figures 8 and 9 show the percentage of telegrams lost in a KNXnet/IP Router considering T_L=0.6s and the KNX additional traffic generator. Comparing Figure 8 and 9 with Figures 6 and 7, respectively, it's clear that now the congestion led to higher values of telegrams lost, both with the normal FIFO and the priority/FIFO mode.

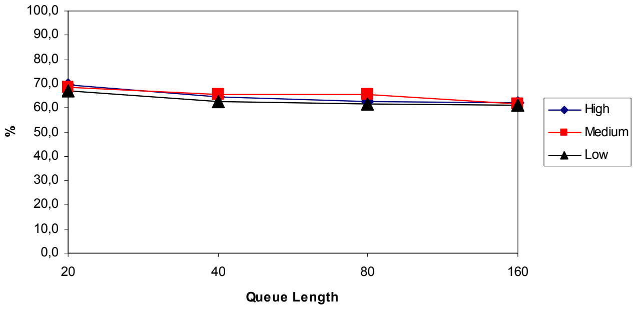


Figure 8. Percentage of telegrams lost, T_L=0.6s, T_M=T_L/2 and T_H=T_L/3, considering a normal FIFO mode & KNX additional traffic generator

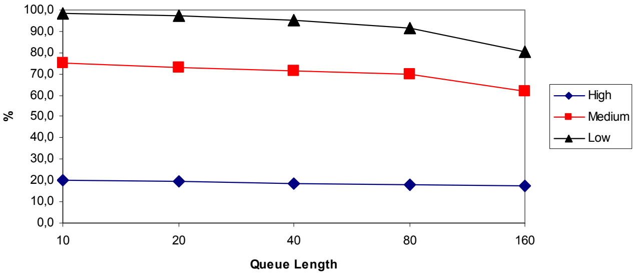


Figure 9. Percentage of telegrams lost, T_L=0.6s, T_M=T_L/2 and T_H=T_L/3, considering a priority/FIFO mode & KNX additional traffic generator

Tables 1 and 2 point out, for each priority and for each of the normal FIFO and the priority/FIFO modes, the percentage of telegrams incoming from IP network that have been enqueued but couldn't be transmitted in the KNX subnetwork, due to the higher access medium contention; these percentages are indicated with % H not Tx, %M not Tx, %L not TX, for the three priorities. The tables compare these percentages with those related to the total number of telegrams lost (%H Tot, %M Tot, %L Tot). From the values shown in the tables, it's clear that the percentage of telegrams enqueued and not transmitted in the KNX subnetworks is low for all the priorities; in other words the KNXnet/IP Routers is able to transmit the most part of telegrams incoming from IP and already enqueued, whith a very low percentage of failure.

This means that the KNX subnetworks aren't congested, i.e. the bandwidth is still available. According to what said, it's clear that the higher percentage of telegrams lost depicted by Figures 8 and 9, as compared with the results shown by Figures 6 and 7, mainly depends on the high rate of the incoming telegrams from IP network, and are influenced by the higher waiting times in the queue of the KNXnet/IP Router. In any case, the worst behaviour depicted by Figures 8 and 9 is not due to a congestion in a KNX subnetwork, but is still due to a congestion in the KNXnet/IP Router.

Table 1. Percentages of Telegrams Lost, normal FIFO mode.

Queue Size	% H not Tx	% H Tot	% M not Tx	% M Tot	% L not Tx	% L Tot
20	2,1	69,5	1,9	68,5	1,9	67,2
40	2,1	64,6	1,8	65,3	2,0	62,5
80	2,0	62,8	1,9	65,3	2,1	61,6
160	2,1	62,3	1,7	61,8	2,0	61,1

Table 2. Percentages of Telegrams Lost, priority/FIFO mode.

Queue Size	% H not Tx	% H Tot	% M not Tx	% M Tot	% L not Tx	% L Tot
10	5,0	20,0	0,4	75,0	0,0	98,3
20	5,0	19,8	0,3	73,0	0,0	97,1
40	5,0	18,5	0,3	71,6	0,0	95,0
80	5,3	18,1	0,2	69,7	0,0	91,4
160	5,2	17,7	0,1	61,9	0,0	80,3

V. Final Remarks

The results presented in the paper have confirmed a well known problem; due to the very limited transmission speeds foreseen for the KNX networks, the KNXnet/IP Router may represent a bottleneck in routing telegrams coming from the IP network whose destination is a KNX device located in the KNX subnetwork. If the KNXnet/IP Router receives more telegrams from IP network than it is able to sell off, the internal queue can fill up to the maximum size and consequently all the other received telegrams will be lost.

Although the congestion problem is well known, important remarks may be done on the basis of the results achieved.

First of all, it seems that congestion isn't a sporadic event as the traffic conditions considered in the performance evaluation wasn't particularly critical. This points out the real need of the definition of efficient congestion flow control procedures inside the KNXnet/IP standard.

Secondly, the performance evaluation carried on pointed out the strong impact of the forwarding rules of the KNXnet/IP Router on the congestion. The use of priority/FIFO mode allows a drastic reduction of high and medium priority telegrams loss. On the basis of the results achieved, it's highly recommended to adopt an efficient forwarding rule in the implementation of KNXnet/IP Router.

References

[1] KNX Association, KNX Standard, System specifications: *KNXnet/IP Routing*, Chapter 3/8/5, 2009.

[2] KNX Association, KNX Standard, System specifications: *Physical Layer General*, Chapter 3/3/1, 2001.

[3] KNX Association, KNX Standard, System specifications: *Communication media Powerline*, Chapter 3/2/4, 2001.

[4] KNX Association, KNX Standard, System specifications: *KNX Radio Frequency,* Volume 3, Supplement 22, 2003.

[5] KNX Association, KNX Standard, System specifications: *Communication media Twisted Pair 0*, Chapter 3/2/1, 2001.

[6] KNX Association, KNX Standard, System specifications: *Communication media Twisted Pair 1*, Chapter 3/2/2, 2001.

[7] KNX Association, KNX Standard, System specifications: *Data Link Layer General*, Chapter 3/3/2, 2001.

[8] KNX Association, KNX Standard, System specifications: *Network Layer*, Chapter 3/3/3, 2001.

[9] KNX Association, KNX Standard, System specifications: *Transport Layer*, Chapter 3/3/4, 2001.

[10] KNX Association, KNX Standard, System specifications: *Application Layer*, Chapter 3/3/7, 2001.

[11] http://www.omnetpp.org/

Mapping KNX to BACnet/WS

Stefan Szucsich and Wolfgang Kastner

Vienna University of Technology
Institute of Computer-Aided Automation, Automation Systems Group
Treitlstr. 1-3, A-1040 Vienna
{stszucsich, k} @ auto.tuwien.ac.at
https://www.auto.tuwien.ac.at

In the last years, the desire to access building automation (BA) data from within existing enterprise applications has significantly arisen. For this kind of machine-to-machine interaction, the evolution of Web technologies has created new opportunities and challenges. On this way, Web Services (WS) play a pioneering role, since they follow a modular approach supporting transmission, eventing, discovery, and security. Building Automation and Control Network – Web Services (BACnet/WS) is a WS based standard that allows a simple integration of BA systems and IT networks at the management level. Its services can be used to access and manipulate data from any source. This paper discusses BACnet/WS and shows how a BA system that follows the KNX standard can be represented by means of the BACnet/WS data model. As a proof of concept, a prototype implementation of a BACnet/WS server for KNX networks, is presented.

1 Introduction

Building Automation Systems (BASs) have typically been separated from IT networks so far [1]. Nevertheless, for purposes of management-level system integration, a common, preferably open format for access to BAS datapoints (and possibly management functions as well) that is aligned with IT standards is a key asset. HTTP and HTML certainly are popular in BAS, but are limited to user interfaces since HTML is ill-suited for machine-to-machine communication.

The evolution of the web and web technologies has created new opportunities for solving the integration problem. Web Services (WS) are self-contained, modular software applications that can be published, located, and called across the web [2]. Central to WS are the use of XML for data representation, an XML-based language that provides a model for describing Web Services (WSDL), and a standardized resource access protocol (typically SOAP, sometimes also plain HTTP).

WS technology has several advantages including platform independency and a loose coupling of distributed system components. However, one drawback is the additional overhead introduced by the XML encoding (especially if small amounts of data are transmitted frequently) and the fact that servers cannot push information to clients (unless the latter implement some kind of server functionality themselves). While the first disadvantage can be countered by compressed XML formats (e.g., Efficient XML Interchange (EXI) [3]), the latter one should remain a minor issue given the hardware resources available at the management level. For typical field devices the need for processor power increases as a result of parsing the XML-formatted messages, tough.

WS can be used to implement system architectures based on the Service Oriented Architecture (SOA) paradigm. There are various definitions for the notion of a SOA. This paper follows the definition given in [4]:

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

This basic definition helps to understand the most important concepts of a SOA. Its essential keywords are *automomous* and *interoperable*. *Autonomous* means that the systems are created independently of each other and that they operate independently of their environment. The term *interoperable* implies that the interfaces of services, that are exposed to their environment, are clearly abstracted from the implementation of these services.

2 State of the art

Currently, there exist a few web services based technologies ready for the use in building automation. The most popular ones are OPC Unified Architecture (OPC UA), Open Building Information Exchange (oBIX), and Building Automation and Control Network – Web Services (BACnet/WS).

OPC UA [5] has been specified by the OPC Foundation[1] to overcome the disadvantages of "classical" OLE for Process Control (OPC) specifications. The main problems with classical OPC variants like OPC Data Access (OPC DA) are their platform dependency to Windows-based operating systems and the limited data model where modelling of information is only possible in a very restricted manner. OPC UA has been standardized as series of standards IEC 62541. The structure of the series can be found in [5].

The oBIX specification [6] has been published by the Organization for the Advancement of Structured Information Standards (OASIS)[2] in December 2006. This platform independent technology is designed to provide machine-to-machine communication between embedded software systems over existing networks using standard technologies such as Extensible Markup Language (XML) and Hypertext Transfer Protocol (HTTP). The oBIX specification defines a flexible object model for describing the data and operations available on a server. In oBIX, everything is an object. Objects are also used to describe data types (classes) and operations (method signatures). Together with the possibility to custom define any kind of object by subtyping and composition, this makes the oBIX data model highly extensible.

BACnet/WS [7] has been specified by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE)[3] as addendum to ANSI / ASHRAE standard 135-2004 in October 2006. It extends the Building Automation and Control Network (BACnet) standard [8] by defining a web service interface and data model for the integration of data from field devices into existing enterprise applications. BACnet/WS has been incorporated into ANSI / ASHRAE standard 135-2008.

[1] http://www.opcfoundation.org

[2] http://www.oasis-open.org

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

OPC UA has the most complex data model of all standards mentioned before. The data model of OPC UA is almost arbitrarily extensible by subtyping and composition. Also multiple inheritance is not forbidden in OPC UA [9]. However, only inheritance rules for single inheritance are specified. Furthermore, references with different semantics are provided (symmetric or asymmetric, hierarchical or non-hierarchical). On the other hand, the BACnet/WS data model is quite simple and not extensible. There is no possibility of subtyping. Somewhere in between lies the data model of oBIX. It is concise, but very flexible. Multiple inheritance is explicitly allowed and rules for single and multiple inheritance are specified. The flexibility of the data model is achieved by so called contracts.

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

The integration of the Building Automation System (BAS) into existing enterprise networks leads to a higher need for security. OPC UA defines its own security model including communication channel security, authentication and authorization, and certificates [9]. It also specifies services to satisfy the security requirements. In oBIX and BACnet/WS no security mechanisms are specified. However, they both recommend the use of the Transport Layer Security (TLS) protocol for secure communication. This includes the usage of HTTP over TLS (HTTPS) instead of HTTP. More information about TLS and HTTPS can be found in [10] and [11]. Additional security measures can be placed in the client and server implementation but are not specified by these standards.

Localization can be used to provide multilingual displays or to adapt the format of dates, times, and numbers according to a given locale. Each of the previously mentioned standards allows some kind of localization. In OPC UA, the built-in data type `LocalizedText` allows to add a language tag to a localized string. The format of the language tags is defined in RFC 3066 [12]. In oBIX, localization of some facets is recommended. Auto-conversion of units may also be possible. However, the mechanisms for localization are not specified by oBIX. Localization in BACnet/WS is similar to oBIX. Several attributes are localizable but the mechanisms for localization are left as an implementation issue. Table 1 depicts a short comparison of OPC UA, oBIX, and BACnet/WS.

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

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

3 BACnet/WS in a nutshell

BACnet/WS is based on a service-oriented client/server architecture. It basically defines services for accessing and manipulating data in the server. Implementations of the service requests and responses shall conform to the Web Services Interoperability Organization (WS-I) Basic Profile 1.0 which specifies the transport of XML-formatted data over network using SOAP over HTTP [13]. A BACnet/WS server may act as a gateway to other protocols or own the data locally. Hence, BACnet/WS is independent from the underlying BAS.

3.1 Data model

The data model defined by the BACnet/WS specification is rather simple and not extensible. Its basic element is the *node*. Nodes are hierarchically arranged and described by attributes. Except the root node, i.e., the topmost node in the hierarchy, each node has exactly one parent node and may have an arbitrary number of children.

Attributes are used to describe nodes and may themselves have attributes. BACnet/WS servers may define proprietary attributes at any level of the hierarchy. Some attributes are required for all nodes, and some are optional. However, it can be assumed that the set of available attributes will remain unchanged in normal operation and changes only occur after reconfiguration of the server.

There are three types of attributes. Primitive attributes are attributes of primitive datatypes defined by the XML Schema [14], such as `boolean`, `string` or `double`. Enumerated attributes are enumerations of XML Schema datatype `string` [14]. The set of allowed values is defined by the BACnet/WS standard. The last type, array attributes, are attributes that consist of an array of primitive datatypes. All elements of the array have the same primitive datatype. Arrays can be accessed either as an array of separate elements or as concatenation of all the elements.

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

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

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

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

Structuring the node hierarchy and the naming of nodes is generally of local concern. However, there exist some standardized nodes that allow clients to retrieve some basic information about the server.

As a special feature, the BACnet/WS data model allows nodes to logically appear in different places of the hierarchy. Almost all attributes of a *reference node* reflect the corresponding attribute of the node it refers to (i.e., the so called *referent node*). To avoid ambiguous parent nodes, the reference node's subhierarchy is independent from the referent node's subhierarchy. However, to reflect the whole subhierarchy, the reference node's children may refer to the referent node's children, and so on.

The BACnet/WS data model allows multiple reference nodes to refer to the same referent node. It is also possible to point to other reference nodes. Though, it is not allowed to create loops, nor is a node allowed to refer to itself.

The BACnet/WS standard defines web services that are used to access and manipulate data in a server. Some services allow to specify service options, such as `precision` and `locale`, that manipulate their behavior or their return values. They are specified as character strings of form:

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

again `'[]'` indicates an optional element and `'...'` stands for the iteration of the previous element.

The defined services can mainly be divided into two groups – reading and writing attributes. Basic services can be used to access single values. However, to reduce network traffic, services that operate on a collection of arbitrary multiple nodes are defined. Additionally, services that allow accessing entire arrays or historical data are provided. BACnet/WS allows only the *Value* attribute of a node to be written.

4 Representing KNX in BACnet/WS

In order to allow a BACnet/WS client accessing a KNX installation, a suitable mapping of the KNX system to the BACnet/WS data model has to be found. The representation has to provide the methods for monitoring and affecting the process remotely. Hence, a mapping for datapoint and communication types – i.e., process and management communication – is required. Additionally, a reasonable way for browsing through the mapped datapoints has to be supported.

4.1 KNX interworking

KNX defines rules for interworking to ensure interoperability between products from different vendors without any additional equipment, such as gateways. Applications are split up into functional blocks (FBs) which may be spread over a number of devices. An FB represents a certain functionality of a device, like "binary push button" as part of a switch. Each FB is assigned to a single device and contains a set of datapoints (DPs) and parameters.

Depending on the configuration mode, DPs can be realized as group objects (GOs), interface object properties (IOPs), memory-mapped datapoints (MMs) or polling value datapoints (PVs). GOs are the endpoints of the KNX group communication which is based on a producer/consumer communication model. In KNX, process data is exclusively exchanged via group communication using group addresses. Depending on whether the semantics of a GO are stateful or stateless, data can either be transmitted in a sponaneous push-style or in a request-based pull-style. IOPs are merely used for transmission of management data, i.e., parameters, configuration and diagnostic data. They are accessed via unicasts following a simple client/server communication model. The addressing information for IOPs consists of the physical address of the device, object index, and property key. Typically, MMs are application parameters written by a management client directly into the memory of the device. PVs are accessed by using the fast polling mechanism of KNX not supported on all media.

The syntax and partial semantics of a DP are defined in the datapoint type (DPT). A DPT determines the data type of a value (format and encoding) and the dimension (range and engineering unit).

4.2 DPT mapping

Interaction with a KNX installation requires a suitable representation of DPs within the BACnet/WS server. Hence, each DP is mapped to a certain two-tier node hierarchy. The DP's data type determines the structure of the node hierarchy whereas its dimension is represented by standard attributes. Mapping DPs to node hierarchies is necessary due to a major limitation of the BACnet/WS data model: Only the *Value* attribute of a node can be written by clients. Hence, complex data types – e.g., DPT_B1U3 for dimming and controlling blinds – have to be mapped to multiple nodes, each containing a single value.

The dimension of a value can easily be represented by BACnet/WS standard attributes, such as the attributes *Minimum*, *Maximum*, and *Units*.

Figure 1 shows the mapping of datapoint type DPT_Switch. The datapoint and DPT name are the only information contained by the node representing the datapoint. Its value is represented by a child node of type *Property*, indicating that its information logically belongs to its parent node. The *Writable* attribute states whether or not the node's value can be written by a client. Since BACnet/WS treats boolean values as enumeration of two arbitrary labels, the encoding of the value, i.e., its possible labels, can simply be mapped to the *PossibleValues* and *WritableValues* attributes[4]. Hence, the mapping of other DPT_B1 types only requires to alter the possible and writable labels.

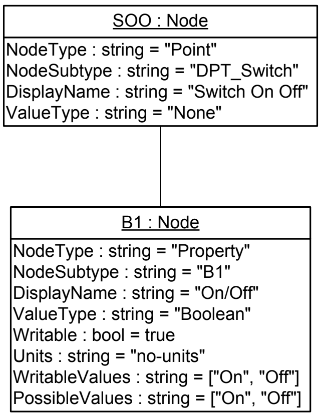

SOO : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Switch"
DisplayName : string = "Switch On Off"
ValueType : string = "None"
B1 : Node
NodeType : string = "Property"
NodeSubtype : string = "B1"
DisplayName : string = "On/Off"
ValueType : string = "Boolean"
Writable : bool = true
Units : string = "no-units"
WritableValues : string = ["On", "Off"]
PossibleValues : string = ["On", "Off"]

Figure 1: Mapping of DPT_Switch

[4] Since all possible values can be written by clients too, the *WritableValues* attribute contains the same labels as the *PossibleValues* attribute.

As mentioned before, the mapping of complex datatypes requires to represent each subvalue by single nodes. In Figure 2 the mapping of the complex DPT_Control_Dimming is shown. The mapping of the boolean value is similar to the mapping of a simple DPT_B1 datapoint type. The stepcode of the DPT_Control_Dimming datapoint type is a 3 bit unsigned integer value and thus is limited to values from 0 to 7. This limitation is achieved by setting the *Minimum* and *Maximum* attributes of the node to the proper values.

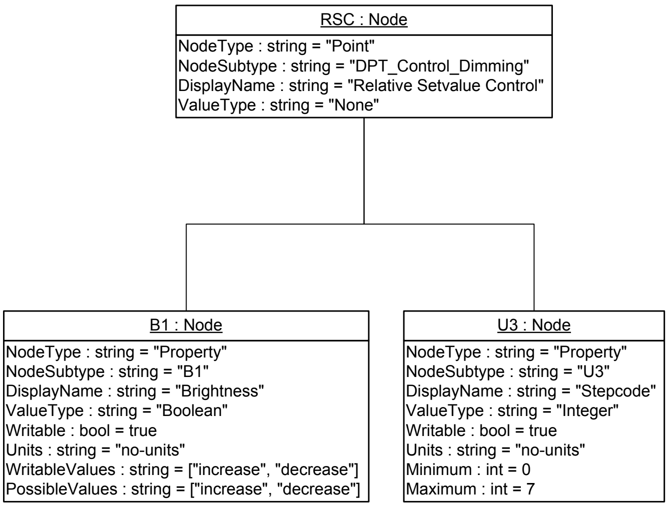

RSC : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Control_Dimming"
DisplayName : string = "Relative Setvalue Control"
ValueType : string = "None"
B1 : Node
NodeType : string = "Property"
NodeSubtype : string = "B1"
DisplayName : string = "Brightness"
ValueType : string = "Boolean"
Writable : bool = true
Units : string = "no-units"
WritableValues : string = ["increase", "decrease"]
PossibleValues : string = ["increase", "decrease"]
U3 : Node
NodeType : string = "Property"
NodeSubtype : string = "U3"
DisplayName : string = "Stepcode"
ValueType : string = "Integer"
Writable : bool = true
Units : string = "no-units"
Minimum : int = 0
Maximum : int = 7

Figure 2: Mapping of DPT_Control_Dimming

As the interested reader can imagine, simple datapoint types like DPT_B1 and DPT_U8 could be mapped to a single node containing the value. However, for the sake of regularity, simple datapoint types are also mapped to a two-tier node hierarchy as shown in Figure 1.

4.3 Communication services mapping

After having found a suitable mapping of DPTs, the various communication modes remain to be mapped. As mentioned before, KNX process communication solely relies on group communication. However, depending on the communication mode associated with a particular group address, different types of interaction with the KNX installation occur.

The first type of interaction to be considered is request-based transmission. Since the BACnet/WS server acts as a management station, this case only makes sense with the server as data sink. In this case, associated GOs have state semantics, i.e., the server can always obtain the current values from the KNX network. When the BACnet/WS client reads one of the values of a complex DP, the server has to retrieve the entire complex DP value from the KNX network.

The second type of interaction to look at is spontaneous transmission. In case of spontaneous transmission, the associated GOs have event semantics. With the server as data sink, incoming events trigger an update of the server's internal process image, i.e., its node hierarchy. When acting as data source, a transmit operation for sending commands to the KNX network is provided.

In KNX, management communication happens via IOPs and point-to-point communication using physical addresses. Mapping this mode of communication is simple as long as access is restricted to single properties. The access types of IOPs are read-only and read-write.

4.4 Browsing the server

With the datapoint type and communication services mapping defined, the methods for interacting with the KNX system exist. However, a suitable way for browsing the BACnet/WS server remains to be found.

The structure of KNX systems allows two different views of such systems – the physical and the logical view. The physical – or device centric – view is based on the physical addresses of KNX devices whereas the logical – or group address centric – view is based on the logical group addresses. Since the data model of BACnet/WS consists of hierarchically arranged nodes, both views can easily be reflected by the BACnet/WS server. Hence, browsing the server can be done iteratively beginning with the root node, without any knowledge about the actual KNX installation.

As mentioned above, one possibility of mapping the KNX system structure to the BACnet/WS data model is the physical view. By reflecting the device centric view, the system functionality is perceived as a set of KNX devices and their associated functional blocks. Each FB itself consists of group objects and interface object properties. The physical view is shown in the left part of Figure 3. Since the BACnet/WS server might be connected to more than one KNX network, domain nodes are introduced to separate them from each other. A domain node can contain up to 16 area nodes, each of them containing up to 16 line nodes. Each line node itself can hold up to 256 device nodes. Device nodes themselves may have an arbitrary number of functional block nodes associated with them. A functional block node in turn consists of IOP and GO nodes. Due to the use of reference nodes, a GO node contains a "logical" datapoint for each group address associated with it.

While the device centric view is suitable for accessing parameters and diagnostic data, it is not useful for controlling the process. In the physical view, there is no information about the function that will be effected by changing a GO's value. However, due to the use of reference nodes, it is possible to switch between the physical and logical view (note the dashed arrows in Figure 3).

The logical view allows a more suitable way for controlling the process (see the right part of Figure 3). It focusses on system functions – i.e., group addresses – rather than individual devices. The functions are structured by functional entities (FEs), such as "HVAC" or "Lighting". Each FE can be composed of system functions – called actions – or other FEs. This view allows to browse the server in an intuitive way and simplifies the means of controlling the process.

As a matter of principle, node identifiers can be chosen arbitrarily. However, for functional entity and action nodes descriptive identifiers should be chosen since paths are used to access nodes via web services. Area, line, and device nodes may be named using their respective part of individual device addresses, yielding to paths containing the device addresses.

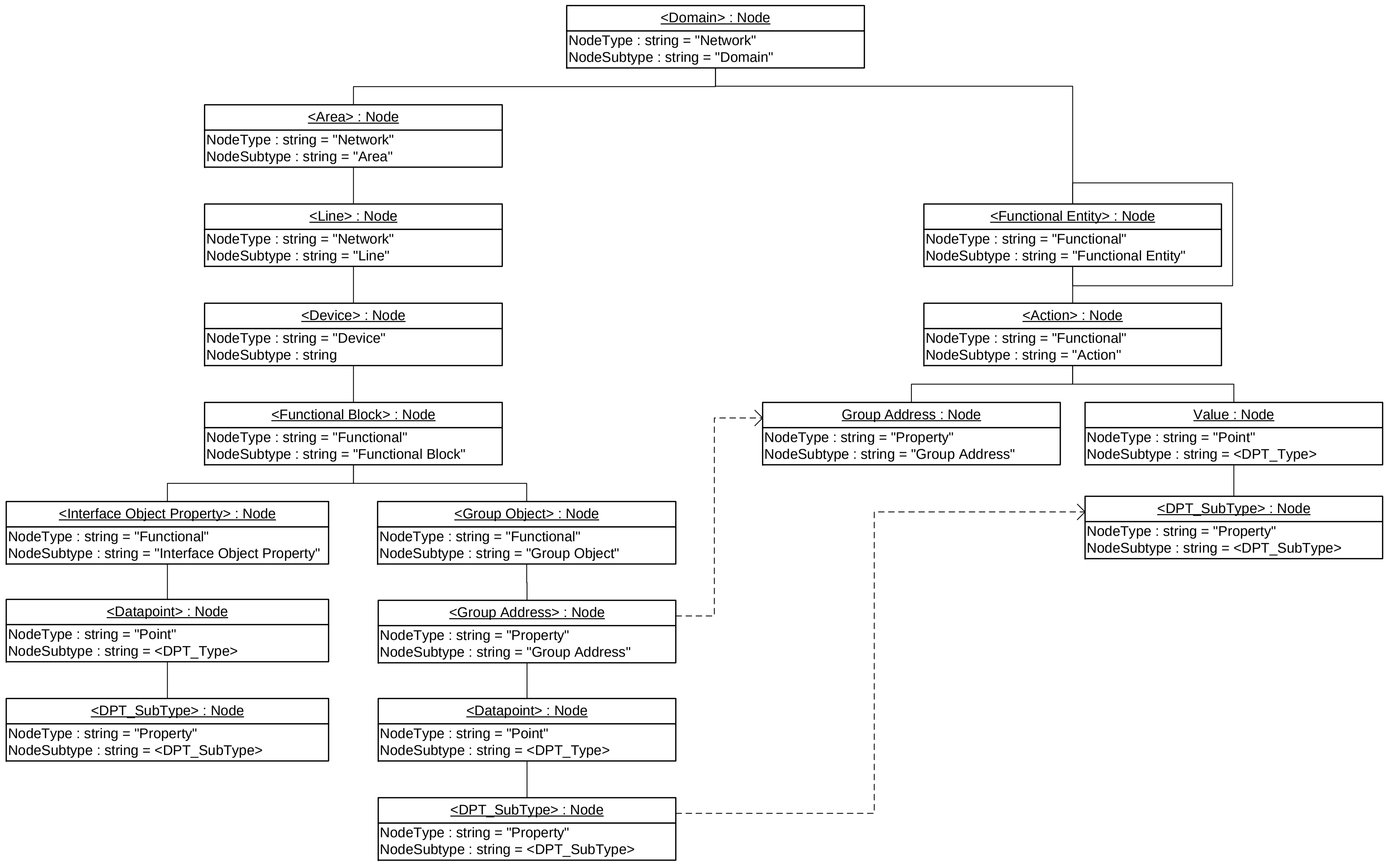

<Domain> : Node
NodeType : string = "Network"
NodeSubtype : string = "Domain"
<Area> : Node
NodeType : string = "Network"
NodeSubtype : string = "Area"
<Line> : Node
NodeType : string = "Network"
NodeSubtype : string = "Line"
<Device> : Node
NodeType : string = "Device"
NodeSubtype : string
<Functional Block> : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Block"
<Interface Object Property> : Node
NodeType : string = "Functional"
NodeSubtype : string = "Interface Object Property"
<Datapoint> : Node
NodeType : string = "Point"
NodeSubtype : string = <DPT_Type>
<DPT_SubType> : Node
NodeType : string = "Property"
NodeSubtype : string = <DPT_SubType>
<Group Object> : Node
NodeType : string = "Functional"
NodeSubtype : string = "Group Object"
<Group Address> : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
<Datapoint> : Node
NodeType : string = "Point"
NodeSubtype : string = <DPT_Type>
<DPT_SubType> : Node
NodeType : string = "Property"
NodeSubtype : string = <DPT_SubType>
<Functional Entity> : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Entity"
<Action> : Node
NodeType : string = "Functional"
NodeSubtype : string = "Action"
Group Address : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
Value : Node
NodeType : string = "Point"
NodeSubtype : string = <DPT_Type>
<DPT_SubType> : Node
NodeType : string = "Property"
NodeSubtype : string = <DPT_SubType>

Figure 3: Physical and logical view

4.5 Case study

After having discussed the fundamentals of the KNX-to-BACnet/WS mapping, a simple KNX installation shall further exemplify the mapping. The KNX network of interest consists of two lamps, two switches, and a dimmer switch and provides the following functions: turn on/off the lights individually, dimming control for “Lamp 2”, and turn on/off all lights, i.e., a central function for both lamps. The KNX network and its devices with their individual addresses and associated group addresses are depicted in Figure 4.

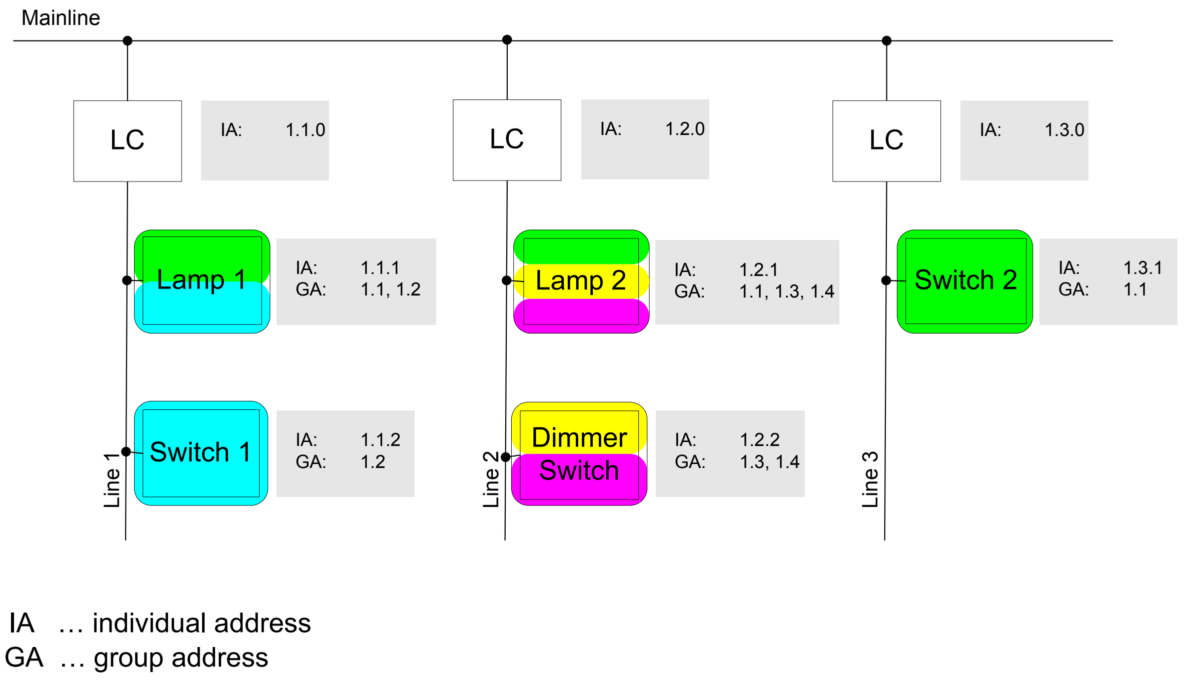


Figure 4: Example KNX installation

The resulting physical view of this installation is shown in Figure 5. Note that only the group objects "Switch On Off" (SOO) and "Relative Setvalue Control" (RSC) of “Lamp 2” are fully depicted. Missing nodes are indicated by “empty” connections, such as for the group object "Info On Off" (IOO) of “Lamp 2”. Assuming that the server is not connected to any other KNX installation, the domain node, as shown in Figure 3, can be omitted.

Figure 6 depicts the corresponding logical view. Each system function is associated with an action. Again, missing nodes are indicated by empty connections.

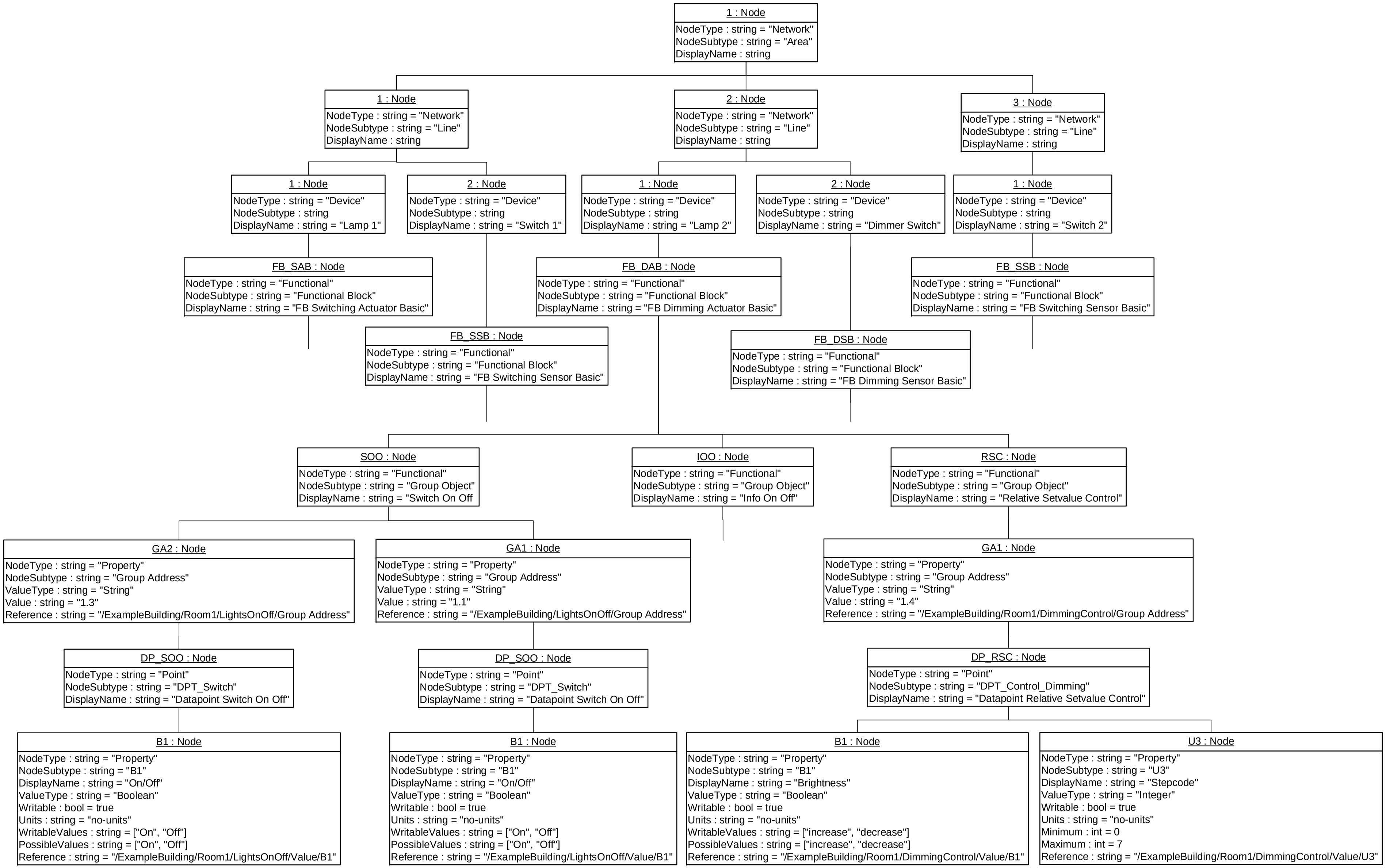

1 : Node
NodeType : string = "Network"
NodeSubtype : string = "Area"
DisplayName : string
1 : Node
NodeType : string = "Network"
NodeSubtype : string = "Line"
DisplayName : string
2 : Node
NodeType : string = "Network"
NodeSubtype : string = "Line"
DisplayName : string
3 : Node
NodeType : string = "Network"
NodeSubtype : string = "Line"
DisplayName : string
1 : Node
NodeType : string = "Device"
NodeSubtype : string
DisplayName : string = "Lamp 1"
2 : Node
NodeType : string = "Device"
NodeSubtype : string
DisplayName : string = "Switch 1"
1 : Node
NodeType : string = "Device"
NodeSubtype : string
DisplayName : string = "Lamp 2"
2 : Node
NodeType : string = "Device"
NodeSubtype : string
DisplayName : string = "Dimmer Switch"
1 : Node
NodeType : string = "Device"
NodeSubtype : string
DisplayName : string = "Switch 2"
FB_SAB : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Block"
DisplayName : string = "FB Switching Actuator Basic"
FB_DAB : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Block"
DisplayName : string = "FB Dimming Actuator Basic"
FB_SSB : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Block"
DisplayName : string = "FB Switching Sensor Basic"
FB_SSB : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Block"
DisplayName : string = "FB Switching Sensor Basic"
FB_DSB : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Block"
DisplayName : string = "FB Dimming Sensor Basic"
SOO : Node
NodeType : string = "Functional"
NodeSubtype : string = "Group Object"
DisplayName : string = "Switch On Off
IOO : Node
NodeType : string = "Functional"
NodeSubtype : string = "Group Object"
DisplayName : string = "Info On Off"
RSC : Node
NodeType : string = "Functional"
NodeSubtype : string = "Group Object"
DisplayName : string = "Relative Setvalue Control"
GA2 : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
ValueType : string = "String"
Value : string = "1.3"
Reference : string = "/ExampleBuilding/Room1/LightsOnOff/Group Address"
GA1 : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
ValueType : string = "String"
Value : string = "1.1"
Reference : string = "/ExampleBuilding/LightsOnOff/Group Address"
GA1 : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
ValueType : string = "String"
Value : string = "1.4"
Reference : string = "/ExampleBuilding/Room1/DimmingControl/Group Address"
DP_SOO : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Switch"
DisplayName : string = "Datapoint Switch On Off"
DP_SOO : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Switch"
DisplayName : string = "Datapoint Switch On Off"
DP_RSC : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Control_Dimming"
DisplayName : string = "Datapoint Relative Setvalue Control"
B1 : Node
NodeType : string = "Property"
NodeSubtype : string = "B1"
DisplayName : string = "On/Off"
ValueType : string = "Boolean"
Writable : bool = true
Units : string = "no-units"
WritableValues : string = ["On", "Off"]
PossibleValues : string = ["On", "Off"]
Reference : string = "/ExampleBuilding/Room1/LightsOnOff/Value/B1"
B1 : Node
NodeType : string = "Property"
NodeSubtype : string = "B1"
DisplayName : string = "On/Off"
ValueType : string = "Boolean"
Writable : bool = true
Units : string = "no-units"
WritableValues : string = ["On", "Off"]
PossibleValues : string = ["On", "Off"]
Reference : string = "/ExampleBuilding/LightsOnOff/Value/B1"
B1 : Node
NodeType : string = "Property"
NodeSubtype : string = "B1"
DisplayName : string = "Brightness"
ValueType : string = "Boolean"
Writable : bool = true
Units : string = "no-units"
WritableValues : string = ["increase", "decrease"]
PossibleValues : string = ["increase", "decrease"]
Reference : string = "/ExampleBuilding/Room1/DimmingControl/Value/B1"
U3 : Node
NodeType : string = "Property"
NodeSubtype : string = "U3"
DisplayName : string = "Stepcode"
ValueType : string = "Integer"
Writable : bool = true
Units : string = "no-units"
Minimum : int = 0
Maximum : int = 7
Reference : string = "/ExampleBuilding/Room1/DimmingControl/Value/U3"

Figure 5: Physical view of the example installation shown in Figure 4

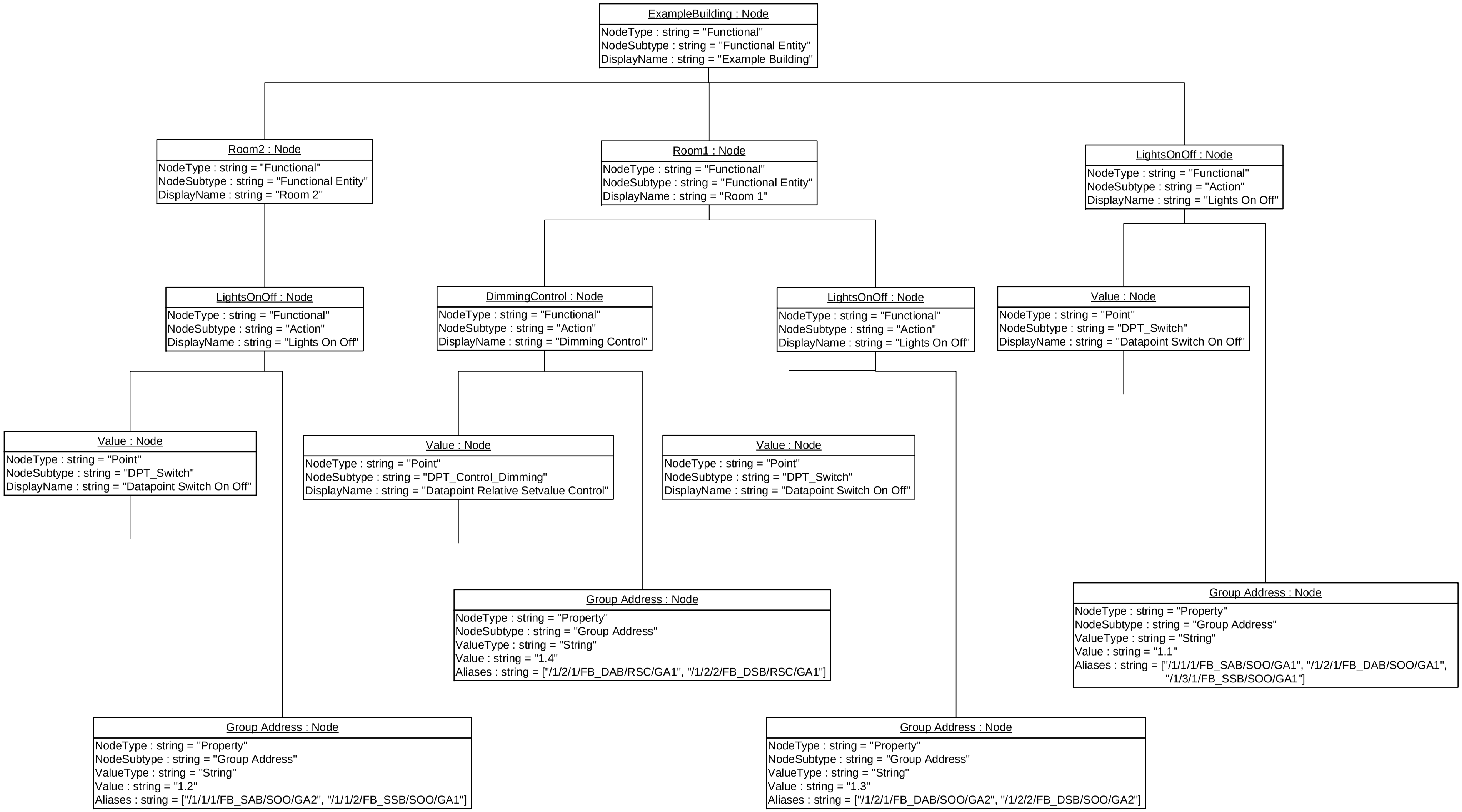

ExampleBuilding : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Entity"
DisplayName : string = "Example Building"
Room2 : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Entity"
DisplayName : string = "Room 2"
Room1 : Node
NodeType : string = "Functional"
NodeSubtype : string = "Functional Entity"
DisplayName : string = "Room 1"
LightsOnOff : Node
NodeType : string = "Functional"
NodeSubtype : string = "Action"
DisplayName : string = "Lights On Off"
LightsOnOff : Node
NodeType : string = "Functional"
NodeSubtype : string = "Action"
DisplayName : string = "Lights On Off"
DimmingControl : Node
NodeType : string = "Functional"
NodeSubtype : string = "Action"
DisplayName : string = "Dimming Control"
LightsOnOff : Node
NodeType : string = "Functional"
NodeSubtype : string = "Action"
DisplayName : string = "Lights On Off"
Value : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Switch"
DisplayName : string = "Datapoint Switch On Off"
Value : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Switch"
DisplayName : string = "Datapoint Switch On Off"
Value : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Control_Dimming"
DisplayName : string = "Datapoint Relative Setvalue Control"
Value : Node
NodeType : string = "Point"
NodeSubtype : string = "DPT_Switch"
DisplayName : string = "Datapoint Switch On Off"
Group Address : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
ValueType : string = "String"
Value : string = "1.1"
Aliases : string = ["/1/1/1/FB_SAB/SOO/GA1", "/1/2/1/FB_DAB/SOO/GA1", "/1/3/1/FB_SSB/SOO/GA1"]
Group Address : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
ValueType : string = "String"
Value : string = "1.4"
Aliases : string = ["/1/2/1/FB_DAB/RSC/GA1", "/1/2/2/FB_DSB/RSC/GA1"]
Group Address : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
ValueType : string = "String"
Value : string = "1.2"
Aliases : string = ["/1/1/1/FB_SAB/SOO/GA2", "/1/1/2/FB_SSB/SOO/GA1"]
Group Address : Node
NodeType : string = "Property"
NodeSubtype : string = "Group Address"
ValueType : string = "String"
Value : string = "1.3"
Aliases : string = ["/1/2/1/FB_DAB/SOO/GA2", "/1/2/2/FB_DSB/SOO/GA2"]

Figure 6: Logical view of the example installation shown in Figure 4

5 Server implementation

5.1 Design

The prototype implementation of the server is intended as a proof of concept. It uses Java and is based on the Calimero library, version 2.0 [15]. Calimero is used for DPT transcoding and network access via KNXnet/IP Tunnelling. Figure 7 shows the design of the BACnet/WS server.

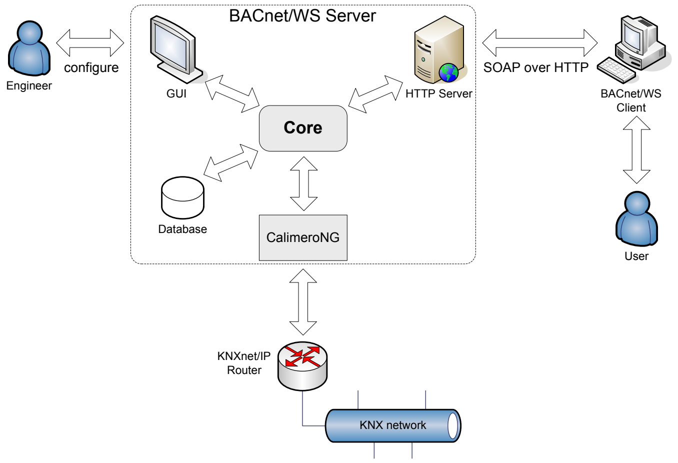


Figure 7: Server design

The HTTP server implements the SOAP over HTTP protocol binding necessary for dealing with BACnet/WS service requests and responses. It handles incoming service requests from the BACnet/WS client and invokes the appropriate methods on the core. It is also responsible for setting up the service response and returning it to the client. In the case study given above, the following service request could be used to turn on “Lamp 1”:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:setValue xmlns:m="http://example.server.org/">
      <Options>readback=false</Options>
      <Path>/ExampleBuilding/Room2/LightsOnOff/Value/B1:Value</Path>
      <Value>On</Value>
    </m:setValue>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The GUI module allows engineers to configure the server. However, it is also possible to control the process via this integrated configuration tool.

The BACnet/WS server core provides methods for actually executing service requests. It accesses the KNX network to retrieve the current values requested and, before returning them to the HTTP server, stores them into the internal process image. The database holds configuration data and static information about the KNX network, i.e., datapoints, communication modes, and addressing information.

5.2 Functionality

Since the server implementation is intended as a proof of concept, it has several limitations. DTP mapping is limited to the most popular types. Also BACnet/WS services are restricted to basic access services. Historical data access is not available.

Table 2 depicts the parameters of the supported `getValue` service. This service can be used to retrieve any single attribute of a single node. If more than one attribute has to be read, the `getValues` service can be used instead. Since "batching" read operations reduces network traffic, the `getValues` service should be preferred. Table 3 summarizes the parameters of the `getValues` service.

Table 2: Parameters and result of the `getValue` service

Parameter Name	**Datatype**	**Description**
Options	string	The option string
Path	string	The path to the requested attribute
Result	string	The requested value as single string

Table 3: Parameters and result of the `getValues` service

Parameter Name	**Datatype**	**Description**
Options	string	The option string
Paths	array of string	An array containing the paths to the requested attributes
Results	array of string	An array containing the requested values as single strings

When reading array attributes using the `getValue` service, array elements are concatenated separated by semicolons (';') and returned as single strings. If an array of strings should be returned rather than a single string, the `getArray` service can be used instead. The parameters of the `getArray` service are shown in Table 4.

Table 4: Parameters and result of the `getArray` service

Parameter Name	**Datatype**	**Description**
Options	string	The option string
Path	string	The path to the requested array attribute
Result	array of string	The requested array attribute

Clients can write the *Value* attribute of any single node by using the `setValue` service. "Batching" write operations can be achieved through the use of `setValues` service. Tables 5 and 6 depict the parameters of the `setValue` and `setValues` services.

Table 5: Parameters and result of the `setValue` service

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

Table 6: Parameters and result of the `setValues` service

Parameter Name	**Datatype**	**Description**
Options	string	The option string
Paths	array of string	An array containing the paths to the *Value* attributes to be written
Values	array of string	An array containing the new values to be set
Results	array of string	An array containing single return strings

The integrated configuration tool allows engineers to manually create and modify the internal process image. Also a possibility to import XML data from the KNX Engineering Tool Software is desirable.

For reducing the KNX network load, retrieved values can be cached in the internal process image. After a configurable timeout, the cache gets refreshed by retrieving the current value from the KNX network.

6 Outlook

Although published as BACnet Addendum, BACnet/WS is not tied to BACnet as an underlying BAS. It can equally be used to expose data from non-BACnet systems as has been shown for KNX in this paper. The fundamental primitive data element in the BACnet/WS data model are nodes which are arranged in a tree structure. A node is mainly a static collection of attributes that represent a single aspect or quality of a node, such as its value, unit or resolution. BACnet/WS services operate upon node attributes. Services are provided for retrieving and modifying single primitive values as well as “plural” versions of these services operating on a collection of arbitrary multiple nodes.

As seen in the case study given above, the proposed mapping seems to require high effort for manual server configuration. However, due to some major limitations of the BACnet/WS data model, it is not possible to simplify the mapping such that the effort for manual server configuration can effectively be reduced. Thus, importing XML data from the KNX Engineering Tool Software is desirable to reduce manual configuration effort to a minimum.

Providing access to historical data would also be preferable. As defined in the BACnet/WS specification, various resampling methods would have to be supported to satisfy requests for historical values independent from the actual sample rate of the server.

Further steps would also include access to management data via IOPs and a performance evaluation regarding access to the KNX network.

Acknowledgment

This work was funded by FFG (Austrian Research Promotion Agency) under the EraSME/COIN project “WebCom" P824675.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[15] *Calimero — EIBnet/IP Tunnelling (and more) for Java*. http://calimero.sourceforge.net/ (accessed October 23, 2010).

KNX and OPC UA

Wolfgang Granzer [*], Wolfgang Kastner [*], Paul Furtak [‡]
[*] Vienna University of Technology
Institute of Computer Aided Automation
Automation Systems Group
Treitlstraße 1-3, A-1040 Vienna, Austria
{w, k}@auto.tuwien.ac.at

[‡] NETxAutomation Software GmbH
Maria Theresia Strasse 41, A-4600 Wels, Austria
Paul.Furtak@NETxAutomation.com

A key requirement at the management level of modern building automation systems is to provide interoperability between devices and systems that use different technologies from various vendors. A possible solution for this challenge is the use of Web services. Web services provide a generic, platform- and technological-independent way to access and manipulate information. Due to the use of IP networks in today's building automation systems, Web services are gaining increasely importance within the building automation domain. This is especially true for KNX systems, since the upcoming opportunity to use IP as native KNX medium provides new possibilities for integrating Web services into KNX. One of the most important technologies within this field is OPC UA. Therefore, this paper is dedicated to OPC UA in KNX. After an introduction to this technology, an approach to use OPC UA for KNX systems is presented. The paper is concluded with an outlook on OPC UA server and client solutions for KNX systems.

1 Introduction

The control functionality of today's Building Automation Systems (BAS) is organized in a three-level hierarchy [1, 2] (cf. Figure 1). Immediate interaction with the environment is associated with the field level: collecting data (measurement, counting, metering) and physically acting upon the process (switching, setting, positioning). The automation level encompasses the various aspects of automatic control – that is, the execution of control loops and sequences, building upon the functionality of the field level. Global configuration and management tasks (such as visualization and accessing trend logs) are considered management level functionality.

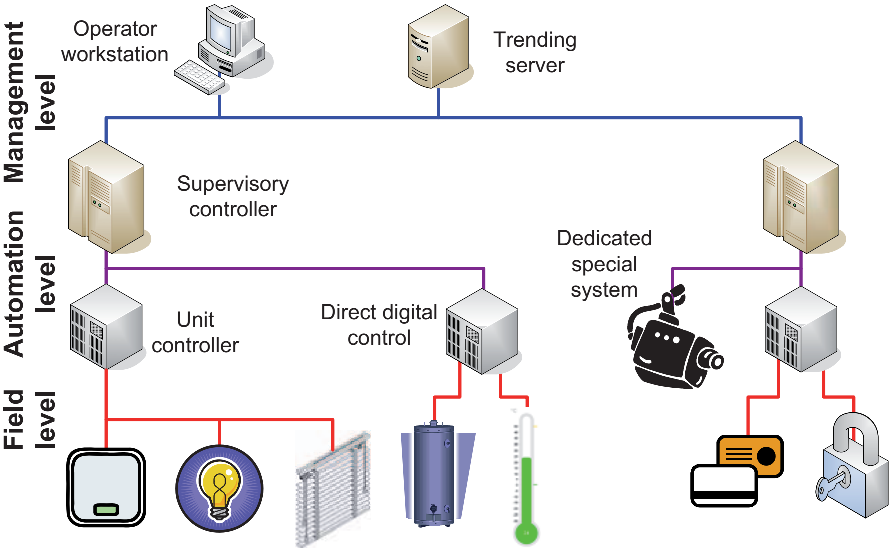


Figure 1: BAS three-level functional hierarchy

To provide the necessary functionality of these different levels, various standards and protocols exist. At the field level and automation level, robustness and cost-efficiency are the most important requirements. Therefore, field bus technologies and embedded devices are commonly used there. The task of the management level is to provide a global view of the entire BAS in order to be able to provide the opportunity for global configuration and maintenance. Furthermore, supporting connections to other, possible heterogeneous systems is the responsibility of the management level. Since network technologies from the IT world are commonly used today at the management level (e.g., IP networks), existing mechanisms from the IT world are also gaining importance within the building automation domain. This is particularly true for KNX based systems. With the introduction of KNXnet/IP as well as the upcoming opportunity to use IP networks as native KNX medium, KNX installations can be easily enhanced with modern IT services. This provides the possibility to integrate new applications that in turn will extend the application domain of KNX.

Key technologies within this area are Web Services (WS). Modern WS implement a Service-oriented Architecture (SOA) where communication is based on messages (message-oriented communication). This SOA paradigm is the key for providing interoperability between devices of different vendors or even different application domains. Consider, for example, Web clients that are connected through heterogeneous networks to access process data within the building automation system. Following a message-oriented communication concept has another benefit. WS are totally platform-independent – they can be implemented using any programming language and run on any hardware platform or operating system. This allows not only an integration of heterogeneous networks but also the usage of miscellaneous hardware/software platforms.

The most important building automation standards that are based on WS are Open Building Information eXchange (oBIX) [3], BACnet Web Services (BACnet/WS) [4], and OPC Unified Architecture (OPC UA) [5]. While nearly no final products exist for oBIX and BACnet/WS, OPC UA is considered as one of the most promising WS technology for building automation. The main reason is that OPC UA is the successor of the so-called classical OPC specifications which are well-established in industrial and building automation, too. Due to its power, OPC UA is a promising candidate for KNX systems. Therefore, this paper is dedicated to OPC UA in KNX. In Section 2, a general introduction to OPC UA and its main features are given. Section 3 demonstrates how OPC UA can be integrated into KNX networks. A key challenge is information modelling. On one hand, this concerns the representation of process data like data points and functional blocks. To achieve this, the interworking model of KNX must be represented using OPC UA's modeling mechanism. On the other hand, meta-information like the building structure and the physical network topology have to be represented, too. This is done by enriching the information model with additional meta-data. Section 4 concludes the paper with an outlook on a prototype of an OPC UA/KNX server.

2 OPC Unified Architecture

In the nineties of the last century, the use of Windows-based PC software for automation systems was getting more and more popular. As a result, many different vendor-specific protocols and interfaces that were not compatible to each other were developed and used. To overcome this incompatibility, a task force of different vendors of Human Machine Interface (HMI) and Supervisory Control and Data Acquisition (SCADA) software was formed in 1995. The aim of this task force was to define a standard for Windows-based device drivers that provide a uniform way to access automation systems. After only one year of development, the first standard was released. This standard provides services for reading, writing, and monitoring of control data. Since it uses Microsoft's technology Windows OLE to provide a unified way to access the network, the name *OLE for Process Control (OPC)* was chosen. During this first release, the nonprofit organization called *OPC foundation* was formed. The OPC foundation is a nonprofit organization that provides and maintains all OPC related specifications. Today, nearly all distinguished vendors that deal with industrial automation are member of the OPC foundation.

In addition to this first standard release, many other OPC specifications have been defined. Therefore, the initial OPC standard was later renamed to *OPC Data Access (OPC DA)*. Other important OPC standards are OPC Alarm&Event (OPC A&E), OPC Historical Data Access (OPC HDA), OPC Batch, OPC Security, OPC Data Exchange (OPC DX), OPC Compliance Tests, and OPC and XML. All these standards have been continuously revised by the OPC foundation. In its current version 3, OPC DA is the most prominent OPC specification [6].

The main reasons of the success of OPC are the focus-oriented design and the existence of well-defined interfaces. The main goal was to provide an easy-to-use Application Programming Interface (API) that hides the complexity of the underlying communication system. Thanks to these unified OPC interfaces, developers of OPC client software are able to concentrate on the functionality of their clients – they do not have to worry about communication details of the underlying network technologies. To provide such a unified communication interface, Microsoft's technologies Component Object Model (COM) and Distributed Component Object Model (DCOM) were used in OPC. Thus, OPC was dedicated to Windows-based systems exclusively. Choosing Microsoft Windows as standard platform for OPC had another advantage. At the time of the first OPC standard release, most HMI and SCADA systems were Windows-based systems and so, a quick adoption to the new OPC standards was possible.

However, while the usage of COM/DCOM was the key driver for OPC's success, relying on COM/DCOM introduces several disadvantages that are getting a major problem in modern automation systems. This is for various reasons:

- OPC is limited to Windows-based systems – implementing OPC on Linux-based platforms or even on embedded devices is not possible.
- COM/DCOM is only practicable for Local Area Networks (LANs). Using it within Wide Area Networks (WANs) like for remote access via the Internet is not possible due to limited support for routing and security of COM/DCOM.
- Configuration and management of COM/DCOM is not an easy task. Especially, configuring the security is a major problem.
- Due to the use of COM/DCOM, the compatibility of OPC software also depends on the used Windows versions. For example, there are problems when Windows XP based systems are used with systems that are based on Windows Vista or Windows 7.

In addition to the lack of COM/DCOM, the OPC specifications have further drawbacks and open issues. The functionality needed to implement a comprehensive OPC software is spread over several OPC specifications. This may introduce a barrier for software developers since they must have detailed knowledge about the different OPC specifications. The situation may be even worse since different versions of the same type of OPC specification are only partly compatible. Another drawback of classical OPC is its rudimentary data model where modelling of information is only possible in a very restricted way. Modelling complex data structures is not feasible due to missing features.

To overcome the drawbacks of these so called *classical OPC specifications*, the OPC foundation decided to define a new specification that is referred to as *OPC Unified Architecture (OPC UA)*. The main goal of OPC UA was to eliminate the dependency to Windows COM/DCOM by using a Service Oriented Architecture (SOA). To achieve this, OPC UA uses Web services or a TCP based protocol for data exchange that both enable full platform-independence. In addition to this new communication concept, another key concept within OPC UA is the support of a powerful and extensible information model which can be used to model any kind of information and data. The current version of OPC UA [5] will soon be published as IEC 62541 [7].

The OPC UA specification consists of 13 parts [8]. OPC UA covers the whole area of all classical OPC specifications and so OPC UA can be seen as a full replacement to classical OPC – if OPC UA is applied, other specifications from classical OPC are not needed anymore. However, to be able to further use classical OPC devices and software, so called OPC UA wrappers can be used. Figure 2 shows an overview of the different parts of the OPC UA specification.

The OPC UA specification has two main goals. First, it defines mechanisms to model information about the process under control (i.e., process data, meta-information about the process and its environment). To achieve this, OPC UA specifies a so called *address space model* which can be used to model any kind of information (cf. Section 2.1). In addition, OPC UA also specifies mechanisms to access the modelled data. Therefore, various communication services and ways to implement them are also included (cf. Section 2.2).

2.1 Address space model and information modelling in OPC UA

A key requirement of modern automation systems is interoperability. To support interoperability between devices of different vendors or between devices that may even use different technologies, it is essential to provide a standardized, well-defined way to model the data that shall be

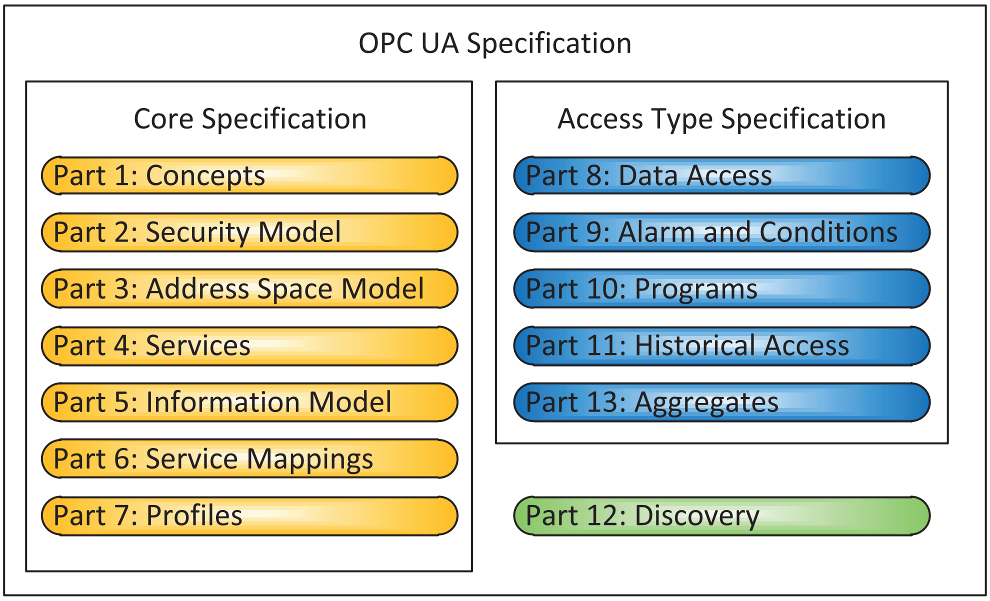


Figure 2: OPC UA specification [8]

accessible by the different devices. In classical OPC DA, modelling data is only possible in a very limited way since important mechanisms like specifying a type hierarchy or using other object-oriented concepts are missing. Therefore, complex data structures can not be defined in OPC DA. To overcome this limitation, OPC UA introduces a so called *address space model* which provides the following features [8]:

- The address space model supports object-oriented mechanisms allowing the definition of type hierarchies and inheritance.
- Type information and type hierarchies can be accessed by clients since they are exposed like normal data.
- The modelling concept allows a definition of models that consists of full-meshed nodes.
- Vendors and developers can extend OPC UA's information model by defining their own models on top of the built-in model.
- Information models are defined server-side and so, clients do not need to be aware of the models since they are able to retrieve them from the server.

A key concept of OPC UA information modelling is that the OPC-specific information model can be extended by defining own models on top of the existing one. Figure 3 shows the basic concept of information modelling in OPC UA.

Information modelling in OPC UA is based on defining *nodes* and *references* between them. Nodes are used to model any kind of information. This includes the representation of data instances or the definition of data types. Each node belongs to exactly one *node class*. In OPC UA, the following node classes are distinguished:

- `Base NodeClass`: This node class represents the base class from which all other node classes are derived.
- `ReferenceType NodeClass`: This kind of nodes is used to define different reference

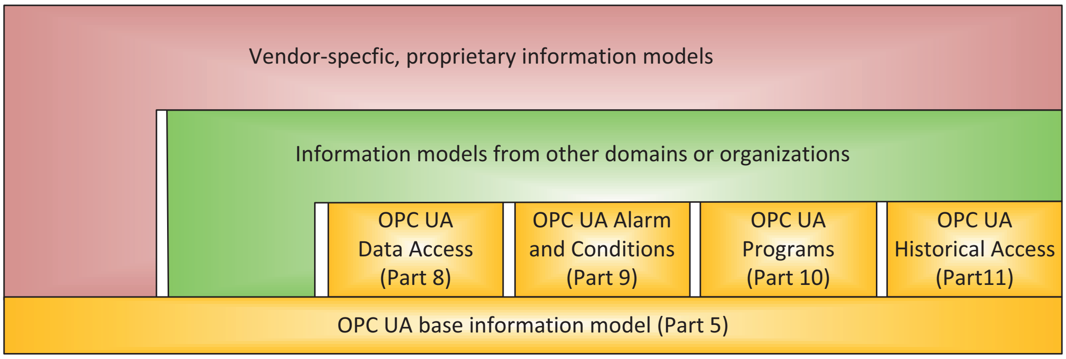

Vendor-specfic, proprietary information models
Information models from other domains or organizations
OPC UA Data Access (Part 8)
OPC UA Alarm and Conditions (Part 9)
OPC UA Programs (Part 10)
OPC UA Historical Access (Part11)
OPC UA base information model (Part 5)

Figure 3: Information models in OPC UA

types.

- `View NodeClass`: A view summarizes a subset of the nodes of the address space. Views are used to make only part of the address space visible to the clients.
- `Object NodeClass`: Objects are used to represent real-world entities like system components, hardware and software components, or even whole systems.
- `ObjectType NodeClass`: This node class is used to provide type definitions for objects.
- `Variable NodeClass`: Variables are used to model values of the system. OPC UA distinguishes between data variables (e.g., data points that represent physical values of the process under control) and properties (e.g., meta-data that further describes nodes).
- `VariableType NodeClass`: This node class is used to provide type definitions for variables.
- `DataType NodeClass`: The `DataType NodeClass` is used to provide type definitions for the values of variables.
- `Method NodeClass`: Methods are used to model callable functions.

Depending on the node class, a node has several attributes that describe it. Some attributes are common to all nodes like the `NodeId` (uniquely identifies a node within the address space), the `NodeClass` (defines the node class), the `DisplayName` (localized text that can be used by a client for displaying the name), and the `BrowseName` (identifies a node when browsing the address space). Other attributes are only available for certain node classes. For example, variables have the attribute `Value` that represents the value of the variable and the attribute `DataType` that indicates the data type of the `Value` attribute. For a complete list of all attributes, see [5].

The list of attributes can not be extended by the user. Adding additional semantic to nodes has to be done by introducing a type hierarchy and by adding references to other nodes. A key concept within OPC UA is the modelling of *complex type definitions* for objects and variables. A complex type is an object or variable that embeds other nodes as sub components. The assignment of these sub nodes to their parent node is done by using references.

Another powerful modelling mechanism is the use of *references*. References are always defined from one to another – references are therefore always one-to-one relations. References can be defined symmetric or asymmetric. Symmetric references are bidirectional and valid in both directions. If a reference is asymmetric, it is possible to specify a name for the inverse direction of the reference. References can further be hierarchical or non-hierarchical. Hierarchical references can be used to introduce topologies and hierarchies within the model. Typical examples of OPC-specific hierarchical references are `Organizes` (used to introduce a general hierarchy of nodes) and `HasComponent` (use to reference a complex type to its sub nodes). An example of an OPC-specific non-hierarchical reference is `HasTypeDefinition` which is used to reference a object or variable to its type definition.

There are many other modelling mechanisms in OPC UA which can not be described here. For more details about these mechanisms, see [5].

2.2 Services and communication in OPC UA

OPC UA is completely based on the client/server model – the communication is performed in *sessions*. To gain access to data, an OPC client establishes a connection to one or more OPC servers. In OPC UA, devices can choose one of two different transport protocols for communication. The first one is based on WS-I Basic Profile 1.1 [9] where SOAP and HTTP are used. The second one is a binary TCP based protocol which has been developed by the OPC foundation.

Since OPC devices are mainly located at the management level where the network may be shared with other application domains, a security model has been introduced in OPC UA. To secure the communication, a *secured channel* is set up during session establishment. This secured channel is provided by the OPC communication layer which is located between the transport layer and the application layer. The secured channel uses cryptographic techniques to guarantee data confidentiality, freshness, and data origin authentication. Entity authentication is provided during session establishment with the help of certificates. Authorization i.e., access control to the control data and services, is left up to the application.

Based on these secure transport protocols, OPC UA defines different services that are used by OPC clients to communicate with the servers and vice versa. These services are grouped into so called *services sets*. Typical service sets are the `Attribute Service Set` which contains services to read and write attributes of nodes as well as the `Session Service Set` that consists of services to open and close a session. For a complete list of all services specified by OPC UA, see [5].

3 OPC UA for KNX

While initially designed for the industrial automation domain, OPC specifications are becoming more and more important for BAS. In the building automation domain, OPC servers are commonly used at the management level where a global view of the entire system is achievable. The main goal of such an OPC server is to collect a live view of the process under control. Furthermore, interacting with the process by, for instance, changing process data is also possible. The resulting process image can be accessed by an OPC client using the well-defined OPC interfaces. Using this process image, OPC clients are able to perform control functionality or management tasks. Typical examples are visualization clients, clients for event and alarm handling, or operator workstations that are used for configuration and maintenance purposes. A possible OPC solution for KNX systems is provided by the company NETxAutomation. NETxAutomation offers KNX enabled OPC DA servers and clients. To communicate with the KNX

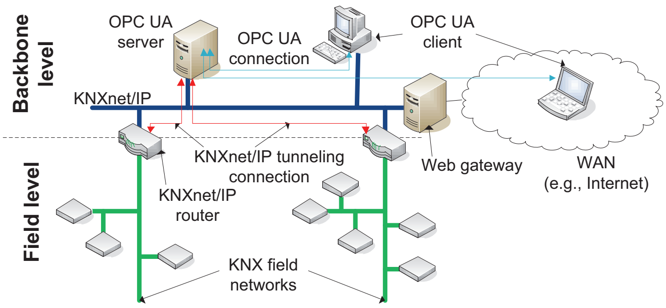


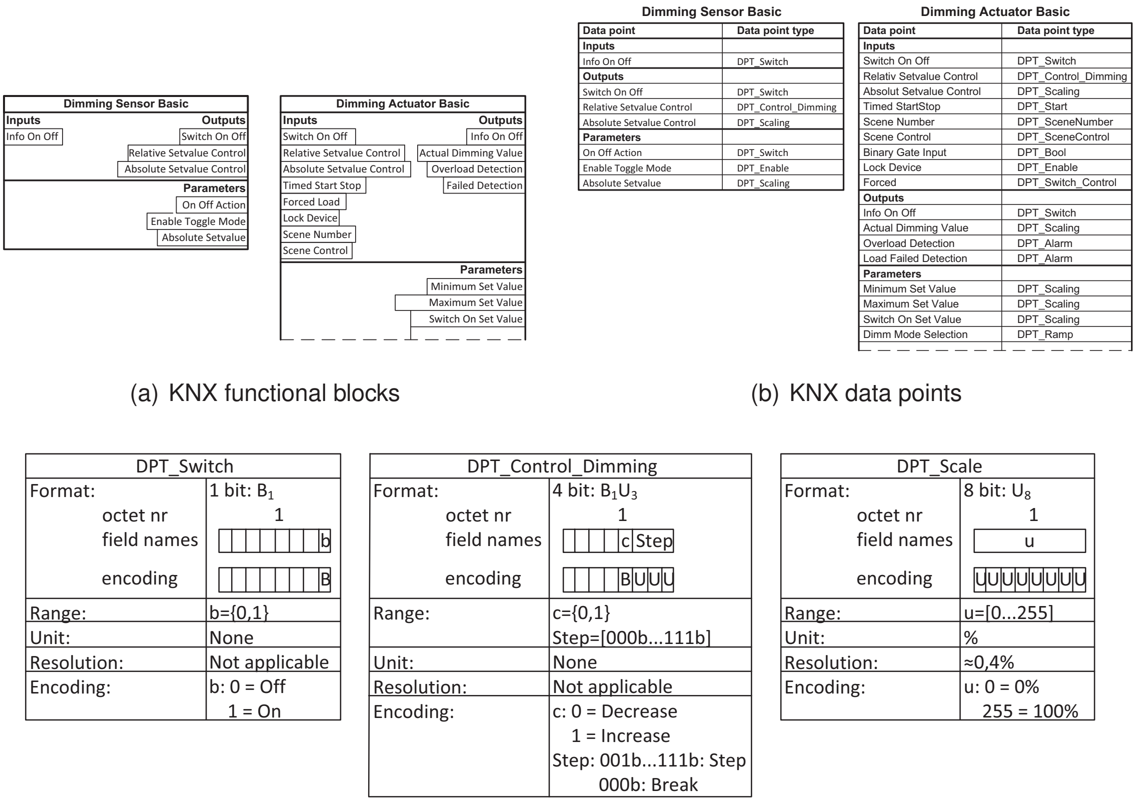
Figure 4: Using OPC in KNX systems

networks, OPC DA servers use KNXnet/IP tunneling to connect to one or more KNXnet/IP routers. Furthermore, it is possible to directly connect to KNX TP 1 field networks via, for example, a USB interface. Figure 4 shows a possible setup. Using the OPC DA interface that is provided by the OPC DA server, OPC DA clients can be deployed that perform visualization or management tasks. For more information about this implementation, see Section 4.

Due to the lack of classical OPC, OPC applications for KNX systems that are based on classical OPC specifications can only provide limited functionality. Because of the limited modelling capabilities, it was only possible to represent KNX data points in a very restricted way – a complete representation of the KNX interworking model (e.g., modelling KNX functional blocks and their data points) as well as a modelling of meta-information about the system (e.g., the building structure) are not possible. However, with the introduction of OPC UA, new opportunities of information modelling come into play.

To represent KNX systems using OPC UA, a key requirement is to map the interworking model of KNX to the address space of OPC UA. To achieve this, the supported modelling capabilities of OPC UA have to be used. KNX systems are typically distributed systems where the control functionality is spread across various so called *functional blocks*. These functional blocks are the key components within the KNX interworking model. KNX functional blocks are distributed to different devices. To be able to exchange data between functional blocks, KNX functional blocks can be bound to each other via the KNX network. Each functional block is dedicated to a single device, and each device can host multiple functional blocks. Each functional block is described by a well-known behavior and consists of one or more so called *data points*. A single data point represents a single data of the application. Depending on the type, it is distinguished between *input data points*, *output data points*, and *parameters*. In addition, functional blocks may have additional I/Os. Figure 5(a) shows the functional blocks `Dimming Sensor Basic` and `Dimming Actuator Basic` that are used as examples throughout the remaining part of the section.

Data points are accessible via the KNX network. To be able to guarantee interoperability, each data point has a defined *data point type (DPT)*. Figure 5(b) shows the DPTs of the data points that are part of the functional blocks `Dimming Sensor Basic` and `Dimming Actuator Basic`.



(a) KNX functional blocks

Dimming Sensor Basic

Data point	Data point type
Inputs	
Info On Off	DPT_Switch
Outputs	
Switch On Off	DPT_Switch
Relative Setvalue Control	DPT_Control_Dimming
Absolute Setvalue Control	DPT_Scaling
Parameters	
On Off Action	DPT_Switch
Enable Toggle Mode	DPT_Enable
Absolute Setvalue	DPT_Scaling

Dimming Actuator Basic

Data point	Data point type
Inputs	
Switch On Off	DPT_Switch
Relativ Setvalue Control	DPT_Control_Dimming
Absolut Setvalue Control	DPT_Scaling
Timed StartStop	DPT_Start
Scene Number	DPT_SceneNumber
Scene Control	DPT_SceneControl
Binary Gate Input	DPT_Bool
Lock Device	DPT_Enable
Forced	DPT_Switch_Control
Outputs	
Info On Off	DPT_Switch
Actual Dimming Value	DPT_Scaling
Overload Detection	DPT_Alarm
Load Failed Detection	DPT_Alarm
Parameters	
Minimum Set Value	DPT_Scaling
Maximum Set Value	DPT_Scaling
Switch On Set Value	DPT_Scaling
Dimm Mode Selection	DPT_Ramp

(b) KNX data points

DPT_Switch	
Format:	1 bit: B_1
octet nr	1
field names	b
encoding	B
Range:	b={0,1}
Unit:	None
Resolution:	Not applicable
Encoding:	b: 0 = Off 1 = On

DPT_Control_Dimming	
Format:	4 bit: B_1U_3
octet nr	1
field names	c Step
encoding	B U U U

Range:	c={0,1}
	Step=[000b...111b]

Unit:	None
Resolution:	Not applicable
Encoding:	c: 0 = Decrease 1 = Increase Step: 001b...111b: Step 000b: Break

DPT_Scale	
Format:	8 bit: U_8
octet nr	1
field names	u
encoding	U U U U U U U U
Range:	u=[0...255]
Unit:	%
Resolution:	≈0,4%
Encoding:	u: 0 = 0% 255 = 100%

(c) KNX data point types

Figure 5: KNX interworking model

Each DPT states the format (i.e., the bit length), the encoding (e.g., unsigned encoding), the range (e.g., the upper and lower bound of the value), and the unit (e.g., percent) of the data point. This detailed description of a DPT guarantees that the sender i.e., the producer and the receiver i.e., the consumer use the same rules to interpret the value of a data point. Figure 5(c) shows the description of the data point types `DPT_Switch`, `DPT_Control_Dimming`, and `DPT_Scale`.

To use OPC UA to access KNX systems, it has to be defined how this interworking model is represented within the address space of OPC UA. To model KNX functional blocks, two different opportunities exist. One possibility is to use complex OPC UA variables. The corresponding data points that are related to a functional block can be modelled as variables that are part of the complex variable. However, representing functional blocks as complex OPC UA variables has a major drawback. Complex variables may only contain variables – including objects or methods is not allowed. While this restriction is not of relevance for modelling KNX functional blocks, representing other interworking models may not be possible. For example, in ZigBee, it is also possible to define commands. Therefore, it is more convenient to use a more generic way by modelling functional blocks as complex OPC UA objects.

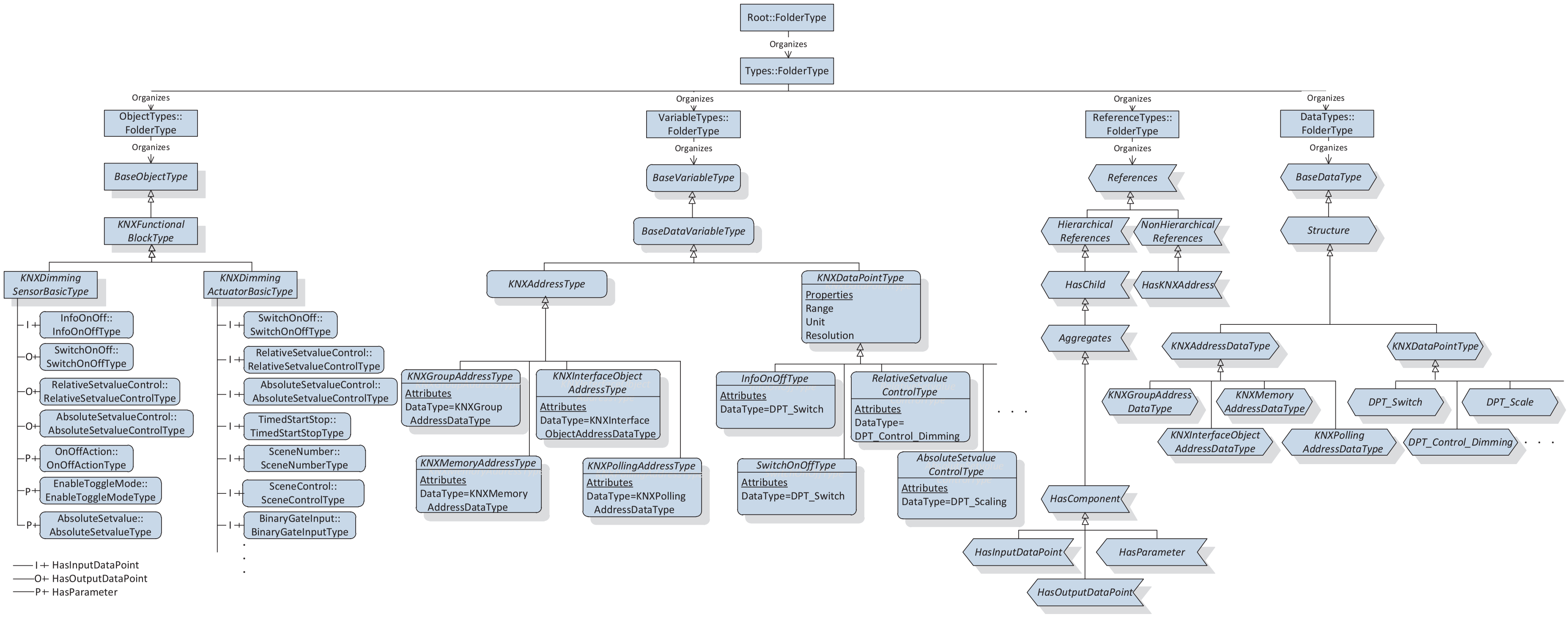


Figure 6: KNX information model for OPC UA

Figure 6 shows how the functional blocks Dimming Sensor Basic and Dimming Actuator Basic are modelled in OPC UA. For each KNX functional block, a complex OPC UA object type is defined. In the provided example, the functional block Dimming Sensor Basic is modelled as KNXDimmingSensorBasicType and Dimming Actuator Basic is represented as KNXDimmingActuatorBasicType. These objects types are all subtypes of the abstract object type KNXFunctionalBlockType which is in turn a subtype of the OPC-specific object type BaseObjectType. Figure 6 also shows the position of the newly defined object types within the standard OPC UA information model.

The corresponding data points of functional blocks are modelled as simple OPC UA variables that are embedded within the object type that represent the functional block. Each data point has a certain, user-defined variable type that are all subtypes of the user-defined variable type KNXDataPointType which is in turn a subtype of the OPC-specific type BaseDataVariableType. The user-defined variable type KNXDataPointType also defines the properties Range (indicates the range of the data point), Unit (defines the engineering unit), and Resolution (specifies the resolution of the data point) that are common to all KNX DPTs. However, since these properties are not applicable to all DPTs, they are optional. To assign a data point to a functional block, OPC UA references are used between the complex object (that represents the functional block) and the simple variables (that represent the data points). To differ between input and output data points as well as parameters, different user-defined reference types are used – input data points are referenced by HasInputDataPoint, output data points by HasOutputDataPoint, and parameters by HasParameter. All three reference types are user-defined ones that are subtypes of the OPC-specific reference type HasComponent. As an example, Figure 6 shows the definition of the user-defined variable types InfoOnOffType, SwitchOnOffType, RelativeSetvalueControlType, and AbsoluteSetvalueControlType. Furthermore, the newly defined reference types HasInputDataPoint, HasOutputDataPoint, and HasParameter are also presented within this figure.

What is still remaining is the representation of the present value of a data point. In OPC UA, process data is modelled using the attribute Value of variables where each value is of certain type. The type is specified by the attribute DataType of the variable. In OPC UA, several built-in data types (e.g., int32, Boolean, . . .) are specified. KNX defines more than 300 different DPTs – each one with its own semantic that differs from all other DPTs. Therefore, representing KNX DPTs by OPC built-in data types is not convenient since the semantic of the DPTs would get lost. Therefore, it is more practical to use user-defined data types in OPC. To define new data types, OPC UA provides three different possibilities. First, it is possible to use so called *simple data types*. Simple data types are derived from the built-in data types. Using simple data types, additional semantic can only be provided by the given name (i.e., via the BrowseName and/or DisplayName attributes) and/or by the Description attribute – changing, for example, the size of the data type or defining structures of concatenated built-in types is not possible. Therefore, simple data types are not sufficient for representing KNX DPTs. The second kind of user-defined data types are called *enumerated data types*. Enumerated data types are similar to the well-known enumerations known from modern programming languages (e.g., C enumerations). An enumerated data type defines a list of allowed strings that are internally mapped to an integer value. In OPC UA, these allowed strings are always mapped to a 32 bit integer – restricting or extending to other integers is not possible. Enumerated data types are useful to model dedicated KNX DPTs. For example, the KNX DPT DPT_Switch can

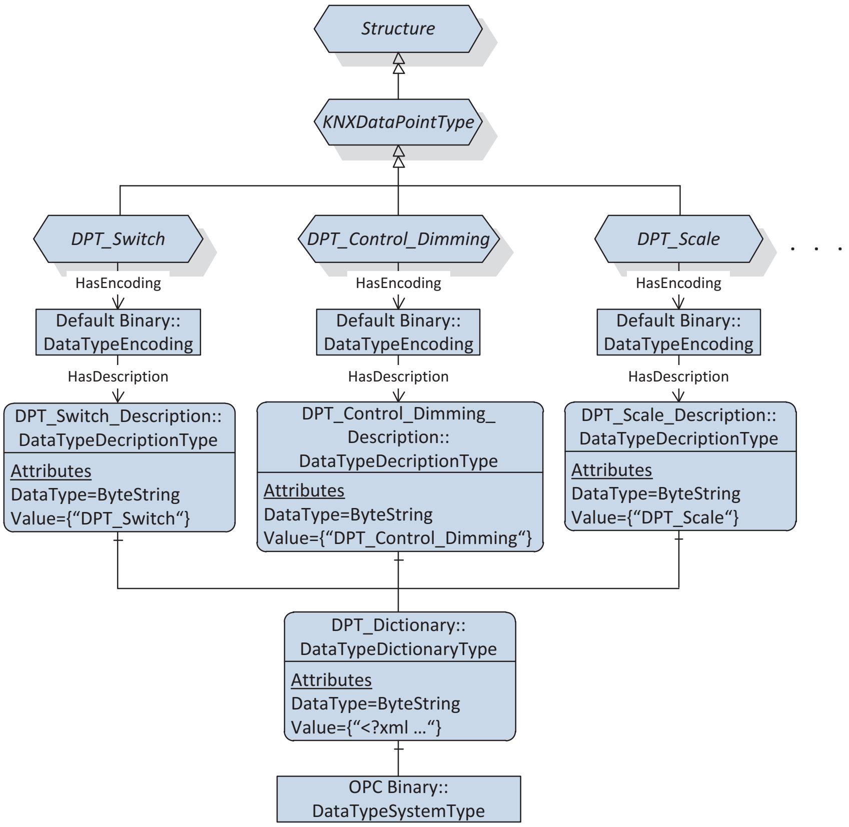

Structure
KNXDataPointType
DPT_Switch
DPT_Control_Dimming
DPT_Scale
. . .
HasEncoding
HasEncoding
HasEncoding
Default Binary:: DataTypeEncoding
Default Binary:: DataTypeEncoding
Default Binary:: DataTypeEncoding
HasDescription
HasDescription
HasDescription
DPT_Switch_Description:: DataTypeDecriptionType
Attributes
DataType=ByteString
Value={“DPT_Switch“}
DPT_Control_Dimming_ Description:: DataTypeDecriptionType
Attributes
DataType=ByteString
Value={“DPT_Control_Dimming“}
DPT_Scale_Description:: DataTypeDecriptionType
Attributes
DataType=ByteString
Value={“DPT_Scale“}
DPT_Dictionary:: DataTypeDictionaryType
Attributes
DataType=ByteString
Value={“<?xml ...“}
OPC Binary:: DataTypeSystemType

Figure 7: KNX DPTs represented in OPC UA

be represented as an OPC enumerated data type where the value `On` is mapped to 1 and `Off` is mapped to 0. However, this mapping introduces an overhead since `DPT_Switch` is defined as 1 bit long but in OPC it is mapped to a 32 bit integer. Furthermore, concatenated DPTs (e.g., `DPT_Control_Dimming` that consists of 1 bit for `Increase` or `Decrease` and 3 bits for the step size) can not be represented. Therefore, the most appropriate solution is to model KNX DPTs as so called *structured data types*. Structured data types are the most powerful but also most complicated way to model data types in OPC UA. Using structured data types, the structure of the data types can be freely defined – it is possible to concatenate several primitives to a single, structured data type where the size of each part can be defined bit-wise. Furthermore, specifying a user-defined encoding is also possible. Another important feature is that the clients do not to be aware of the structure and the encoding of the data types. Since the encoding rules are also represented within the address space of the server, clients can retrieve these encoding rules during runtime. Due to their power, structured data types have been chosen to model KNX DPTs.

Figure 7 shows how the KNX DPTs `DPT_Switch`, `DPT_Control_Dimming`, and `DPT_Scale` are modelled as structured data types. Each user-defined structured data type has to be a subtype of the user-defined data type `KNXDataPointType` which is in turn a subtype of OPC-specific data type `Structure`. Furthermore, each user-defined structured data type must have at least one encoding. An encoding is specified by defining a `HasEncoding` reference to

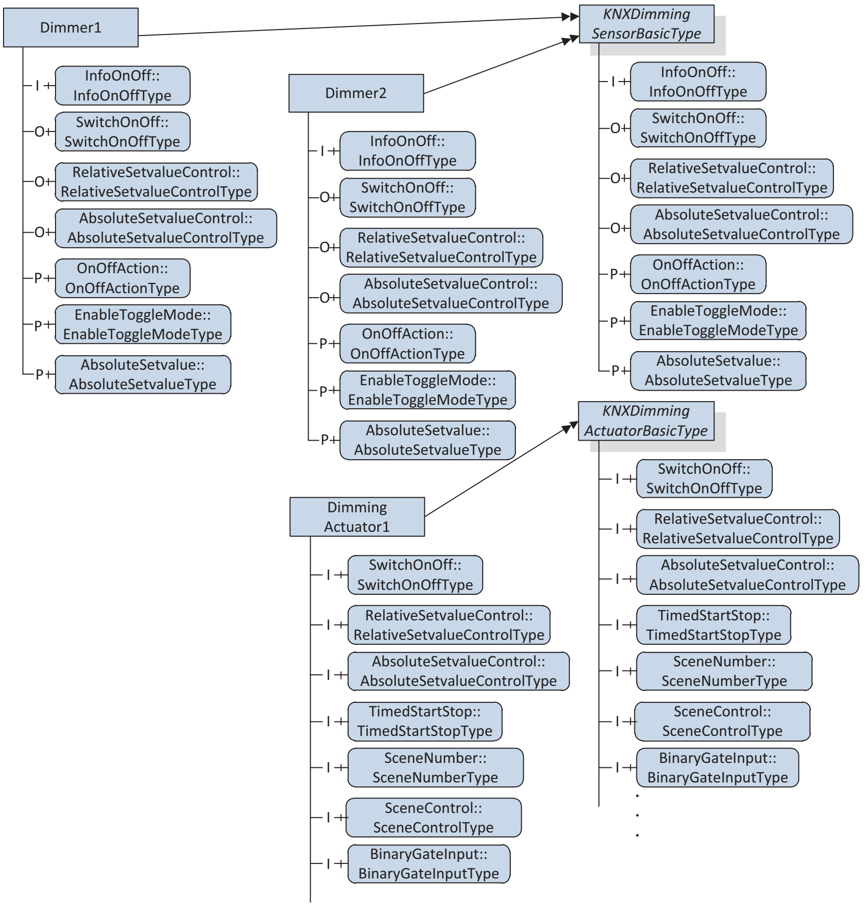
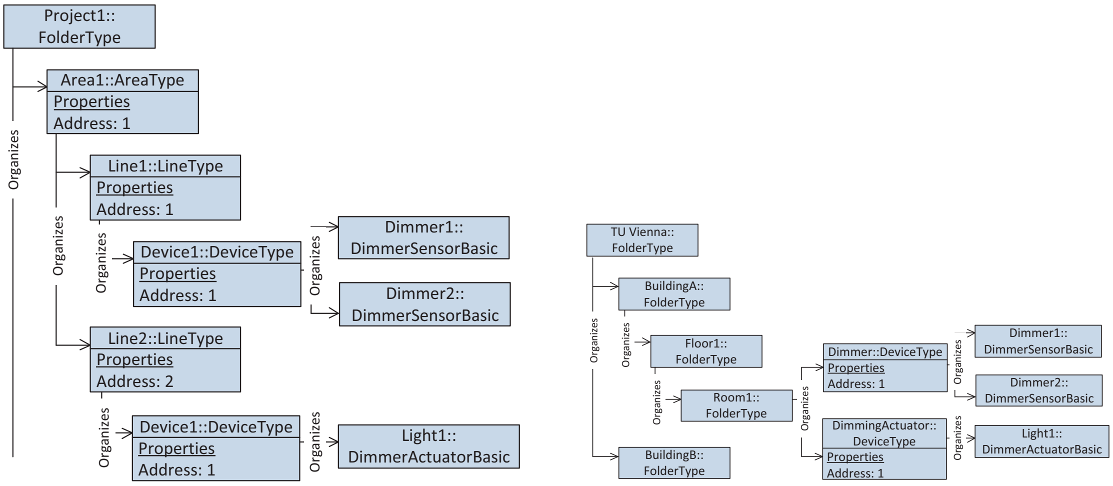


Figure 8: KNX example in OPC UA

a variable instance that is of type `DataTypeEncoding`. OPC clients can choose one of these encoding schemes. If an encoding is not specified by the client, the default encoding is chosen by the server – `Default XML` for XML transport and `Default Binary` for OPC UA binary transport. However, for each user-defined structured data type, only one default encoding has to be specified which is chosen if no other encoding is available. Each encoding variable points to exactly one variable instance of type `DataTypeDescriptionType`. This data type description variable specifies the proper encoding within the so called data type dictionary which is in turn represented by a variable of type `DataTypeDictionaryType`. The value of the data type dictionary variable contains an XML representation of all user-defined encoding schemes. The XML format that is used to specify the encoding is defined within Part 3 of the OPC specification. For the KNX DPTs `DPT_Switch`, `DPT_Control_Dimming`, and `DPT_Scale`, this XML looks as follows:



(a) Network topology

(b) Building view

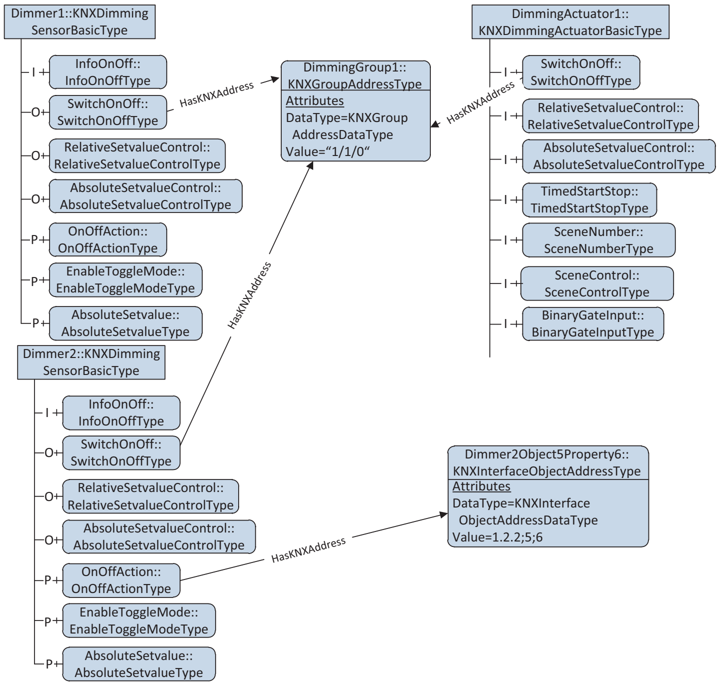
Figure 9: Data views in KNX

```
<EnumeratedType Name="DPT_Switch" LengthInBits="1">
        <EnumeratedValue Name="Off" Value="0"></EnumeratedValue>
        <EnumeratedValue Name="On" Value="1"></EnumeratedValue>
</EnumeratedType>
<StructuredType Name="DPT_Control_Dimming">
        <Field Name="c" TypeName="Bit" Length="1">
                <Documentation>0...Decrease;1...Increase</Documentation>
        </Field>
        <Field Name="Step" TypeName="Bit" Length="3">
                <Documentation>000b...Break;[001b...111b] step size</Documentation>
        </Field>
</StructuredType>
<OpaqueType Name="DPT_Scale" LengthInBits="8">
                <Documentation>0...0%;255...100%</Documentation>
</OpaqueType>
```

After having defined how functional blocks, data points, and DPTs are represented in OPC UA, it has to be specified how dedicated instances of KNX functional blocks are modelled in OPC UA. In other words, it has to be defined how concrete KNX installations are represented using the KNX information model mentioned above. Representing a dedicated KNX installation is done by modelling the concrete functional blocks with their corresponding data points. This is done by instantiating the user-defined object and variable types. Figure 8 shows an example.

In order to get a general view of the KNX installation, it is common to use different system views. Within such a view, the devices that contain the functional blocks and/or data points are assigned to a hierarchical representation of the system. Within ETS 3, two different views are possible. The first one reflects the network topology where the devices are arranged according to their location within the network i.e., to which area and line the device is associated. The second view represents the location of the devices within the physical building structure (e.g., building/floor/room). In addition to these views provided by ETS 3, other views are also possible. For example, it may also be desirable to provide a functional view where the devices are classified according to their functionality (e.g., Lighting/Dimming/DimmingSensor1).

To represent such views in OPC UA, the modelling capabilities of OPC UA can be used. Figure 9 shows examples how such views can be modelled in OPC UA. In Figure 9(a), the network topology as it is used within ETS 3[1] is shown. To model such a network topology, user-defined objects types for areas, lines, and devices are used. These objects types contain the property `Address` which represents the address information of the corresponding object. Figure 9(b) presents a view that reflects the building structure. Within this example, OPC-specific object type `FolderType` together with the hierarchical reference type `Organizes` are used to model the building structure. However, it is also possible to add more semantic by defining specific object types for the different building elements (e.g., object types for rooms, floors, and buildings).



Dimmer1::KNXDimming SensorBasicType
I
InfoOnOff:: InfoOnOffType
O
SwitchOnOff:: SwitchOnOffType
O
RelativeSetvalueControl:: RelativeSetvalueControlType
O
AbsoluteSetvalueControl:: AbsoluteSetvalueControlType
P
OnOffAction:: OnOffActionType
P
EnableToggleMode:: EnableToggleModeType
P
AbsoluteSetvalue:: AbsoluteSetvalueType
HasKNXAddress
DimmingGroup1:: KNXGroupAddressType
Attributes
DataType=KNXGroup AddressDataType
Value="1/1/0"
HasKNXAddress
DimmingActuator1:: KNXDimmingActuatorBasicType
I
SwitchOnOff:: SwitchOnOffType
I
RelativeSetvalueControl:: RelativeSetvalueControlType
I
AbsoluteSetvalueControl:: AbsoluteSetvalueControlType
I
TimedStartStop:: TimedStartStopType
I
SceneNumber:: SceneNumberType
I
SceneControl:: SceneControlType
I
BinaryGateInput:: BinaryGateInputType
HasKNXAddress
Dimmer2::KNXDimming SensorBasicType
I
InfoOnOff:: InfoOnOffType
O
SwitchOnOff:: SwitchOnOffType
O
RelativeSetvalueControl:: RelativeSetvalueControlType
O
AbsoluteSetvalueControl:: AbsoluteSetvalueControlType
P
OnOffAction:: OnOffActionType
P
EnableToggleMode:: EnableToggleModeType
P
AbsoluteSetvalue:: AbsoluteSetvalueType
HasKNXAddress
Dimmer2Object5Property6:: KNXInterfaceObjectAddressType
Attributes
DataType=KNXInterface ObjectAddressDataType
Value=1.2.2;5;6

Figure 10: Address binding of KNX data points in OPC UA

What is still missing is the representation of the address information. To be able to read and write data points, address information that can be used to access the data point over the network has to be assigned to the data points and parameters. From a communication point of view, KNX distinguishes between four different kinds of data points:

- *Group object data points* are addressed using KNX group addresses. To access them, the services `A_GroupValue_Read` and `A_GroupValue_Write` are used.
- *Interface object property data points* are addressed by using the individual address of the device, the object ID, and the property ID. Interface object property data points are accessed by using the services `A_PropertyValue_Read` and `A_PropertyValue_Write`.
- *Memory mapped data points* are addressed by sending `A_Memory_Write` and `A_Memory_Read` requests. The address information consists of the individual address of the devices and the address of the memory region where the data point is mapped to.

[1] While this example is based on functional blocks that are distributed to the different devices, ETS 3 only uses data points that are assigned to the device. However, the presented information model also allows to use only data points instead of functional blocks.

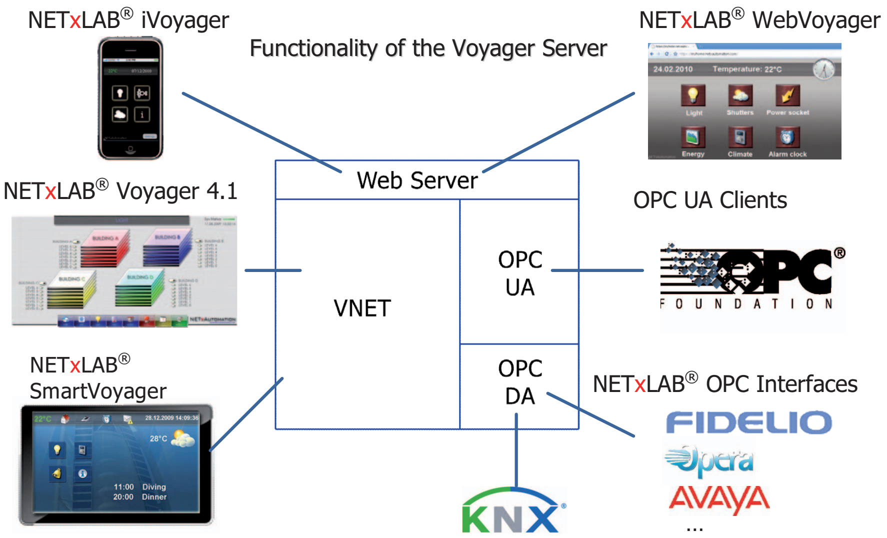


Figure 11: New NETxLABⓇVoyager server with OPC UA interface

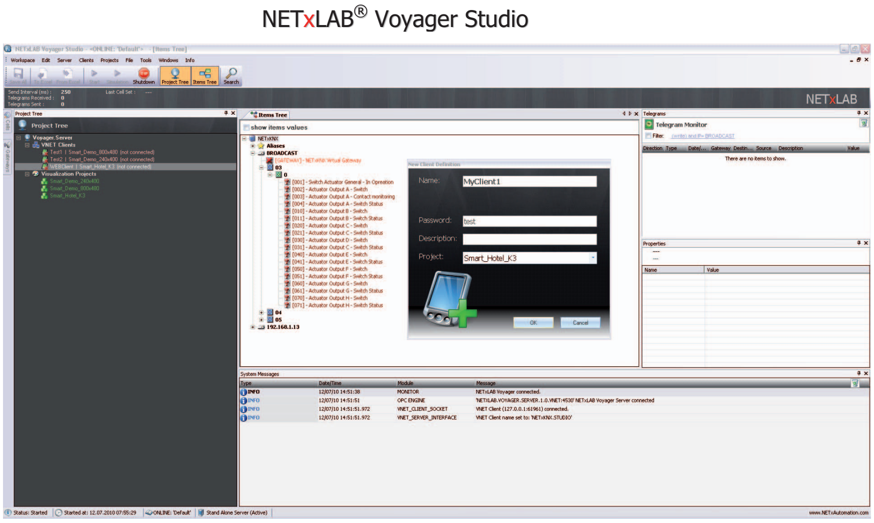
- *Polling value data points* are accessed by using the polling mechanism of KNX.

To model KNX address information, user-defined variable types are specified – `KNXGroupAddressType` for group object data points, `KNXPropertyAddressType` for interface object property data points, `KNXMemoryAddress` for memory mapped data points, and `KNXPollingAddress` for polling value data points. Each variable type has a dedicated data type that is used to represent the corresponding address. Again, structured data types are used. Figure 6 shows the different variable types for modelling address information.

To bind address information to a certain data point, an instance of the variable type that represents the required address has to be defined. To assign this address to a data point, the user-defined reference type `HasKNXAddress` which is a subtype of the OPC-defined reference `NonHierarchicalReferences` is used. Note that the binding between data points and addresses is $n : m$ – an address can be referenced by multiple data points, a data point can reference multiple addresses. Figure 10 shows an example how data points of KNX functional blocks are bound to an address information.

4 OPC UA server for KNX

The NETxLABⓇVoyager Global Visualization enables the control and visualization of KNX and other building automation projects. The system consists of a central server system called NETxLABⓇVoyager server and different NETxLABⓇVoyager clients. Due to the flexible design, the NETxLABⓇVoyager Visualization system is applicable to projects of all sizes and types – from small homes to the large functional buildings. The individual clients are connected to the server which is responsible for providing access to the process data that is required by the clients. The server includes an integrated Web server that enables a global, platform-independent access to the clients. A key technology within the NETxLABⓇVoyager is OPC. Via the OPC DA interface, the visualization NETxLABⓇVoyager clients are able to connect to



New Client Definition

Name: MyClient1

Password: test

Description:

Project: Smart_Hotel_K3

OK Cancel

System Messages

Type	Date/Time	Module	Message
INFO	12/07/10 14:51:38	MONITOR	NETxLAB Voyager connected.
INFO	12/07/10 14:51:51	OPC ENGINE	'NETXLAB.VOYAGER.SERVER.1.0.VNET:4530' NETxLAB Voyager Server connected
INFO	12/07/10 14:51:51.972	VNET_CLIENT_SOCKET	VNET Client (127.0.0.1:61961) connected.
INFO	12/07/10 14:51:51.972	VNET_SERVER_INTERFACE	VNET Client name set to: 'NETxKNX.STUDIO'

Stand Alone Server (Active)

www.NETxAutomation.com

Figure 12: NETxLAB®Voyager Studio

the service as OPC client. Furthermore, other systems (e.g., BACnet) can be integrated by using the NETxLAB®OPC Bridge and the embedded NETxLAB®OPC interface of the server. To increase the data redundancy and reliability, the NETxLAB®Voyager server can be realized as a main/backup solution. Even a cluster configuration of individual, networked servers is possible. Figure 11 shows the structure the NETxLAB®Voyager Global Visualization system.

The server and each client are communicating through OPC DA 2.0. However, due to the drawbacks of COM/DCOM (cf. Section 2), a new protocol called VNET has been developed. VNET provides encrypted and efficient connections on top of TCP/IP. The drawback of VNET is that it is not open for other products from different vendors. Therefore, an OPC UA interface will also be integrated into the NETxLAB®Voyager Global Visualization system. This OPC UA interface shall overcome the limitations of OPC DA and VNET.

In Figure 11, an overview of the new NETxLAB®Voyager server that features an OPC UA interface is shown. The basis for the new system is the OPC DA server with connection to KNX and other systems like Micros Fidelio/Opera and Avaya. In addition, the new VNET server interface for remote connection of NETxLAB®clients like Voyager 4.1 and SmartVoyager client and a Web interface for browser clients like iVoyager or other browsers are also included. A key component will be the new OPC UA interface which can be used by OPC UA clients to access the new NETxLAB®Voyager server. OPC UA clients will be served by Voyager server like other OPC DA or VNET clients. This new OPC UA interface will offer other vendors to use Voyager servers.

The NETxLAB®Voyager Studio (cf. Figure 12) is used to analyze, configure, and update the visualization projects. The entire system is managed centrally. It is also used to monitor and maintain all parts of the Voyager server, the VNET clients, and will be also used to configure the OPC UA clients.

It is planned to integrate the OPC UA information model for KNX that has been presented in Section 3. It will reside beside the OPC DA model to allow customers to adapt to this new model. In conjunction with the built-in Lua scripting language, the OPC UA nodes will be accessible to implement user-defined control logic. So, it will even be possible to create new virtual devices. For OPC UA clients there will be no difference between real devices and devices created within the NETxLAB®Voyager server. This will open a greater flexibility to meet various projects requirements in KNX building automation.

5 Conclusion and future work

The resulting increased complexity of OPC UA is an entry barrier for Small and Medium Enterprises (SMEs). Therefore, the EraSME project “Web-based Communication in Automation"’ (WebCom) has been initiated [10]. The main goals of WebCom are to thoroughly analyze the OPC UA specifications and to provide concentrated information for SMEs.

As a first activity within the project WebCom, an OPC UA information model for KNX has been presented within this paper. Within this information model, the KNX interworking model was mapped to the address space of OPC UA. While a first proof of concept for dimming sensors and actuators has been proposed, the developed model must be further refined and other functional blocks and data points have to be modelled in a similar way. Furthermore, as described in Section 4, it is planned to integrate this information model into the NETxLAB®Voyager Visualization system provided by the WebCom project partner NETxAutomation [11].

In addition to information modelling, research activities within the project WebCom are also concerned with the development of OPC UA Software Development Kits (SDKs). Therefore, the development of an OPC UA server and client SDK is currently underway by the WebCom project partner HB-Softsolution [12]. This SDK provides easy-to-use high-level APIs that can be used to develop OPC UA applications of any sizes and types. Using the SDK, the OPA UA application developers do not need to deal with communication details – instead the developers are able to concentrate on the development of the OPC UA applications.

Acknowledgment

This work was funded by FFG (Austrian Research Promotion Agency) under the EraSME/COIN project “WebCom" P824675.

References

[1] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, “Communication Systems for Building Automation and Control”, *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, 2005.
[2] International Organization for Standardization, “Building Automation and Control Systems (BACS) – Part 2: Hardware”, ISO 16484-2, 2004.
[3] “oBIX 1.0 Committe Specification”, OASIS, 2006.
[4] “BACnet – A Data Communication Protocol for Building Automation and Control Networks”, ANSI/ASHRAE 135, 2008.
[5] “OPC Unified Architecture Specification”, OPC Foundation, 2009.
[6] “OPC Data Access Specification”, OPC Foundation, 2003.
[7] “OPC Unified Architecture”, IEC 62541, 2010, Current status: Approved for FDIS circulation.
[8] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*, Springer, 2009.
[9] “WS-I Basic Profile 1.1”, Web Services Interoperability Organization, 2004.
[10] (Oct. 2010), WebCom – project homepage [Online]. Available: http://www.webcom-eu.org
[11] (Oct. 2010), NETxAutomation [Online]. Available: http://www.netxautomation.org
[12] (Oct. 2010), HB-Softsolution [Online]. Available: http://www.hb-softsolution.com