FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Hardware-in-the-Loop Simulation in der Gebäudeautomation

unter Verwendung von BACnet und PowerRPDEVS

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Stefan Adelmann
Matrikelnummer 01633044

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Franz Josef Preyser

Wien, 7. August 2019

Stefan Adelmann	Wolfgang Kastner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Hardware-in-the-Loop Simulation in the Domain of Building Automation

using BACnet and PowerRPDEVS

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Stefan Adelmann
Registration Number 01633044

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof.Dr. Wolfgang Kastner
Assistance: Dipl.-Ing. Franz Josef Preyser

Vienna, 7th August, 2019

Stefan Adelmann | Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Stefan Adelmann

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

Wien, 7. August 2019

Stefan Adelmann

Kurzfassung

Das Ziel dieser Arbeit ist die Implementierung eines Hardware-in-the-Loop (HIL) Frameworks, welches die Fähigkeiten der hybriden Modellsimulation des PowerRPDEVS Simulators mit der Kontrollfunktionalität einer Testhardware (BACnet Controller) verbindet. Das Projekt umfasst ein Simulations-Backend, das Daten in das Simulationsmodell einbringen und ausgeben kann, und ein Frontend, das diese Daten entsprechenden BACnet Geräten und Objekten zuweist, welche im Gegenzug von der BACnet kompatiblen Hardware gelesen und beschrieben werden können.

Abstract

The goal of this thesis is to implement a Hardware-in-the-Loop (HIL) framework, combining hybrid model simulations via the PowerRPDEVS hybrid simulator with the control functionality of a BACnet controller acting as the hardware under test. The thesis describes the steps taken to create a Simulation Backend, enabling simulation data to flow in and out of the simulation model, as well as a Simulation Frontend that links the simulation data to simulated BACnet devices and objects to interact with the hardware.

Contents

CHAPTER 1

Introduction

1.1 Motivation

In the course of the ongoing digital transformation, also BA (Building Automation) plays a role. Especially commercial buildings can benefit from digitization and usage of BAS (Building Automation Systems) that connect management, automation and field layers altogether. Due to the very definition of a BAS, the underlying network spreads over large distances, across all levels of a building combining different data and control interfaces of HVAC (Heating, Ventilation and Air Conditioning), lighting and shading. Due to this fact, testing and verifying implemented solutions is difficult to accomplish. The solution proclaimed in this thesis is to exploit the underlying network structure of BAS by utilizing a simulation engine and providing input and output interfaces from this simulation to the control hardware that is later deployed in a building. This method of simulating hardware devices as opposed to building a physical testbed is called HIL (Hardware-in-the-Loop) and it is already in widespread use, especially in the field of embedded systems. Thereby, data is fed to the hardware under test from a simulation and the controller is able to interact with the simulation in response.
As an example of a BAS, BACnet (Building Automation and Control network) was chosen in this thesis while the simulation side was developed for the PowerRPDEVS hybrid model simulator. The goal of this thesis is to create a framework (Simulation Backend/Frontend) that allows BACnet controllers to be tested by connecting them to a simulation framework.

1.2 State of the Art

HIL is a process that is more commonly associated with embedded systems. In [ISS99] for example, HIL simulation is presented as a viable option to develop systems that are very dependent on their environment. The authors describe that it is a common problem,

that building a real environment for development or testing is often costly, would take too much time or is not possible. As a practical example, an engine control system is listed in the paper.
On the simulation site, Augenbroe and Godfried describe in [Aug04] that building simulation is an ever growing trend that becomes more and more important especially in the region of HVAC. In [PWBH12], the authors describe that building simulation could be used to identify flaws of the control environment by comparing the real world values with the simulation data.
For the combination of the two concepts, i.e. HIL with BAS simulation, the literature review concluded that there aren't many solutions in this area. [Wet11] presents a type of co-simulation in combination with HVAC systems. The goal of this paper is to connect multiple simulations via middleware and the HVAC model is only used as possible use case.
[SPK17] proposes a BACnet HIL system like described in this thesis. It offers a proof of concept by an example implementation using PowerDEVS as the simulation framework. The model used in the paper though, only shows a single room. Testing was done by monitoring and controlling the BACnet devices with a software called YABE (Yet Another BACnet Explorer). The paper describes such a solution as a viable option for BAS testing. With that it provides a foundation for the goals of this thesis.

1.3 Methodology and Structure of the Thesis

Chapter 2 introduces HIL and provides a design for the system described in this thesis that is not implementation specific. It will describe the ideas and functions of the building blocks whose particular implementation for this thesis is described in chapter 4. The technology overview in chapter 3 presents the two endpoints of the HIL approach described in this thesis. BACnet as the connection to a hardware under test and PowerRPDEVS as the simulation engine. Chapter 4 consists of a section describing the software that is used and two sections on the main building blocks, the *Simulation Backend* and *Simulation Frontend*. The solution proclaimed in this thesis is deployed on the hardware presented in chapter 5. The chapter will also describe the hardware setup of both devices in order to follow the deployment process shown in chapter 6. Deployment covers the model generation, specific hardware setup processes that are necessary to execute a certain simulation as well as the simulation process itself. chapter 7 presents a use case of the implementation as well as the results that are produced by this example application. A conclusion is given in chapter 8 combined with an outlook on possible future work in the area of this thesis. The entire code of the implementation is provided in the appendix chapter 9 as well as in a *SOURCEFORGE* repository ([SRC]).

CHAPTER 2

Design

The goal of the design presented in this thesis is to combine BA with HIL. The next chapters will introduce the HIL process and the translation that is necessary for BA to function in this context.

2.1 Hardware In the Loop

HIL primarily consists of 3 distinct parts, simulation, emulation and control as presented in Figure 2.1.

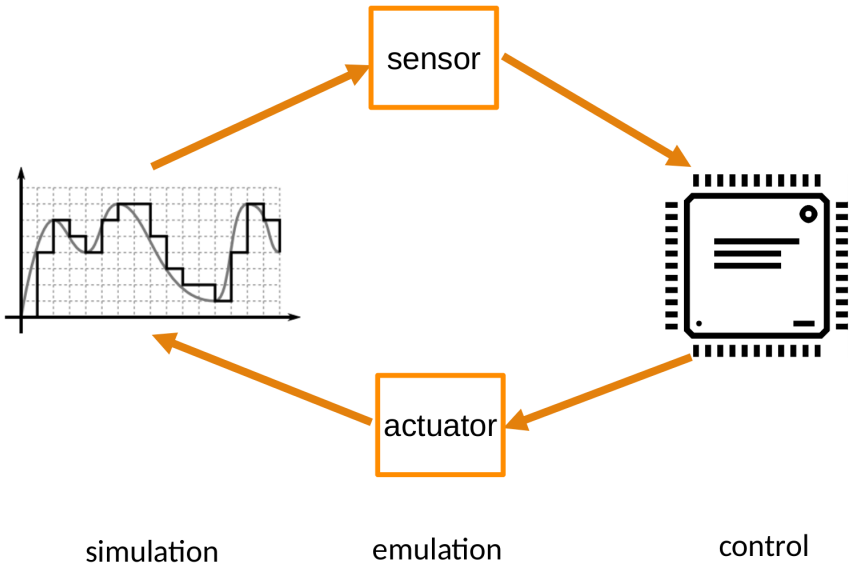


Figure 2.1: Overview of the HIL process

2.1.1 Simulation

The simulation framework is used to simulate real world, physical processes. Since simulations are abstractions of the reality, it is the restriction on how close the HIL testing is to real world behavior of the system. In BAS this simulation will be hybrid, meaning that there is a combination of continuous and discrete processes that interact with each other. As the user is in control of the simulation model, it is very easy to apply different test cases to the system. Worst-case scenarios can be run through by the simulation without damaging components.

2.1.2 Emulation

The simulation and HIL framework are used to test a control hard and software without actually connecting actuators and sensors. In order to deploy the test entity in the real system without changes, the actuators and sensors have to be emulated. An actuator will write its control value to the simulation while a sensor receives data from the simulation.

2.1.3 Control

Control represents the hardware under test. The HIL framework is connected to a controller running the software that needs to be evaluated. The goal of the HIL simulation is to be able to swap the hardware under test directly from the HIL framework to the real world application without needing to change anything about the hard/software.

2.2 HIL Connections

Since the hardware under test is generally given by the nature of the application and there are already proven simulation engines available, the main parts of the actual HIL process are the connections between the three components. In Figure 2.1 two main connection interfaces can be identified.

2.2.1 Simulation to Emulation

Simulation data needs to be able to flow from the simulation to the sensors and control values have to be received from the actuators. In this thesis, this connection will be handled by the so called *Simulation Backend* that is part of the simulation framework. Data is exchanged with the Emulation by the means of a communication protocol.

2.2.2 Emulation to Control

Emulation is handled by providing bus entities to the network that connects the HIL host with the controller itself. This is handled by the *Simulation Frontend*. On the one hand this component exchanges data with the *Simulation Backend*, on the other it establishes a bus communication and acts as the hardware that is emulated.

2.3 HIL in BAS

The BAS process has to be translated from the real hardware approach to the refined HIL process shown in Figure 2.2.

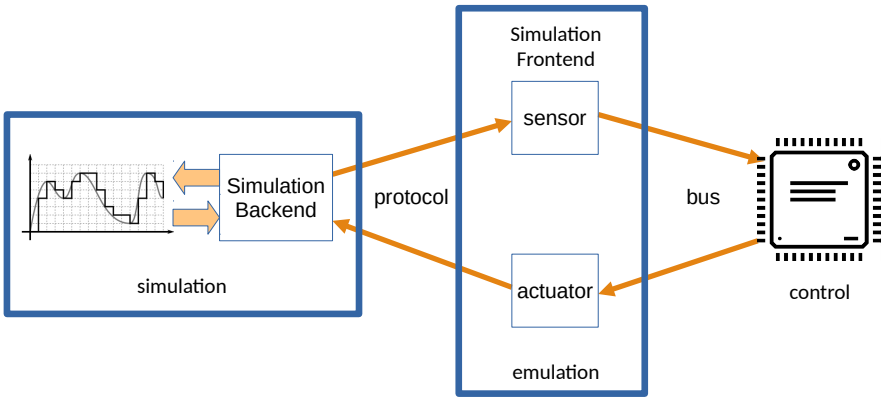


Figure 2.2: Overview of the HIL process

The classical approach (on the left in Figure 2.3) to test the controller software is to implement a physical testbed or the real hardware and connect it to the controller. The physical processes in the rooms have to be changed or influenced externally in order to apply certain test cases to the system. In case of BAS, heaters and temperature sensors are connected and placed in the rooms. So in order to introduce custom testcases apart from the standard room behavior, it is necessary to install additional heaters and coolers to simulate changes. A very costly and difficult process.
HIL in BAS (on the right in Figure 2.3) does not need physical components other than the processor running the software and a device running a simulation. Instead of the heaters and temperature sensors, a computer is connected and a separate software on the PC emulates the physical devices and simulates their behavior. The loop runs from the simulation (temperature values), to the controller which in turn reacts and controls certain actuators (heater on/off or power level) in the simulation. The simulation covers the physical properties of the rooms (e.g. the temperature behavior) as well as the influence of the actuators (e.g. heaters) according to their technical specification.

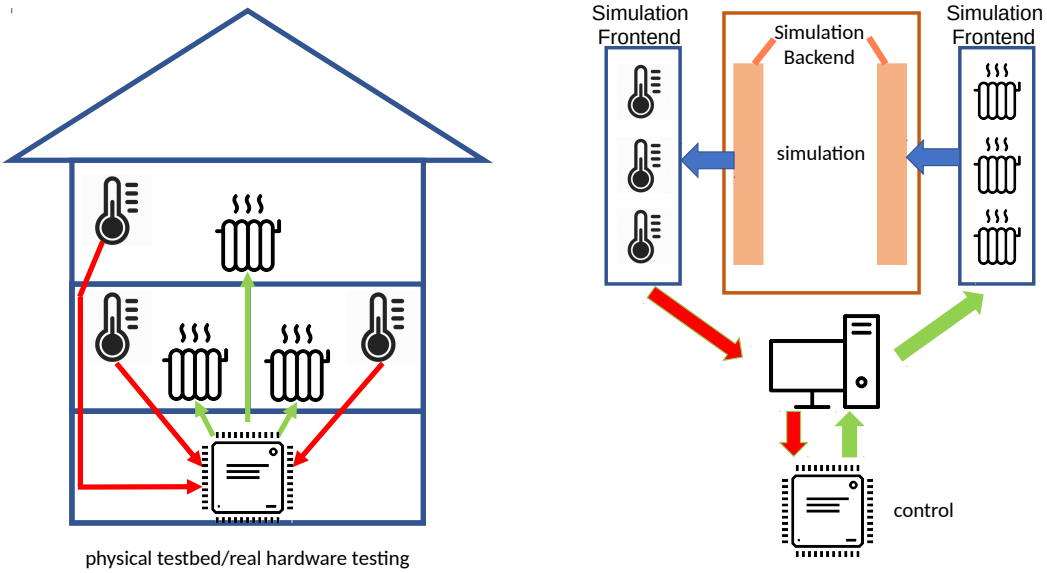


Figure 2.3: Comparison physical testbed vs. HIL in BAS

CHAPTER 3

Technology Overview

3.1 BACnet

BACnet was published in 1995 by the ASHRAE (American Society of Heating, Refrigeration and Air Conditioning Engineers) in cooperation with ANSI. It was especially designed to be used in BAS. In 2003, it was elevated to the ISO-Standard 16484-5 that can, like the previous version, be implemented without a license. Its goal is to enable an open and inter-operable communication between devices in the BAS framework connecting the field, automation and management level while staying network neutral. In order to be recognized as a standard, specialized network layers were added.

3.1.1 Objects

BACnet is an object oriented protocol. Physical objects are represented as BACnet objects of different types. In this thesis, BACnet/IP is used which means that individual objects communicate across the network with different IP addresses but on a shared port within the range 47808-47823, or more commonly written 0xBAC0-0xBACF.

Due to the object oriented nature of BACnet, real world physical devices like temperature sensors are reduced to BACnet objects that provide certain values like an analog value representing the measured temperature. Most physical devices provide multiple sensory data inputs or actuator outputs. Therefore, BACnet provides a special type of object, the BACnet device, which is able to function as a container that encapsulates 1-n different BACnet objects of other types. Every object consists of properties that describe the object and its functionality, with the most important being the *Object_Identifier*, consisting of the *Object_Type* and a number. This combination must be unique in the context of the object. Devices as container classes need a unique number across the whole network

Object Type	Description
Analog Input	Sensor inputs (e.g. Temperature sensors)
Analog Output	Control
Binary Input	Digital input (e.g. switch)
Binary Output	Digital output (e.g. relay)

Table 3.1: Available types of BACnet objects

Property	Type	Example
Object Identifier	Required	[Object Type] 1
Object Name	Required	"BI 1"
Object Type	Required	Binary Input
Present Value	Required	Active
Status Flags	Required	In_Alarm, Fault, Overridden, Out_Of_Service
Event State	Required	Normal
Out Of Service	Required	False
Units	Required	Degrees-Celsius
Description	Optional	"Heater On/Off"
Device Type	Optional	"Floor Heating Type 1"
Reliability	Optional	No_Fault_Detection
Update Interval	Optional	1s
COV Increment	Optional	0.5

Table 3.2: Available BACnet properties

since they are all of the same type of object. The objects they encapsulate only need to be unique inside the device. Unique for objects means that there is no other object with the same *Object_Type* and *Object_name*. Different objects with different types can share the same name, for example AI_1, BI_1.
BACnet consists of 18 different types of objects including the device type. The object types used in this thesis are shown in Table 3.1.

An object in turn consists of up to 25 properties that provide values to be read and to be written as well as to describe the capabilities and preferences of the object. Certain properties are required to be set. Including the identifier foremost, as well as flags and states of the object. A listing of possible properties is shown in Table 3.2.

3.1.2 Services

To access objects as well as their properties, BACnet uses different services that range from discovery, to read and write operations of different types. Generally speaking, confirmed and unconfirmed services can be distinguished. Unconfirmed services are often implemented as broadcasts, where different objects answer to a global request but no acknowledgment is expected. Confirmed services address a certain device and that device in turn answers with an acknowledgment that can contain the result of the service call, for example, the result of a read operation.
In sum, 32 services can be divided into 5 categories. The following sections describe the 3 service categories that are used in this thesis. File-access and virtual-terminal services are not part of this thesis.

Remote Device Management

An important part in a distributed environment is the discovery of the nodes that are active in the network. This is handled by the remote device management services in BACnet. Additionally, these services provide setup methods like time synchronization as well as the possibility for objects of the same vendor to initiate a private transfer of data that is not seen by other objects.
The 4 services that are used in this thesis are shown in Table 3.3. Type U means that the service is unconfirmed.

Service	Type	Parameters	Description
Who-Is	U	Device ID range	ask if a device is present
I-Am	U	Device identification	answer to an Who-Is
Who-Has	U	Dev. ID range, Object-ID	ask which device contains an object
I-Have	U	Device/Object-ID	answer to an Who-Has

Table 3.3: Remote Device Management Services

Object Access

Services of this class are responsible for the direct interaction with properties of an object. They include different kinds of read and write operations like simple reads/writes as well as complex ones that combine properties of different objects. There is also the possibility of removing and adding objects to a device. The 4 services used in this thesis are shown in Table 3.4.

Due to the nature of the object access services all of them are confirmed services, meaning that the caller sends a request to the called object which answers with an acknowledgment.

Service	Parameters	Description
ReadProperty	ID, Property	read an object property
ReadPropertyMultiple	List of ReadProperty	read multiple properties across objects
WriteProperty	ID, Property, Value	write a value to a property
WritePropertyMultiple	List of WriteProperty	write a value to multiple properties

Table 3.4: Object Access Services

Alarm and Events

In application cases with a high number of BACnet-devices, requesting property values every time the controller requires a value would be very inefficient. For less communication delays and streamlining the communication as a whole, certain events can be used in the BACnet-network. A list of the available Event-Services is presented in Table 3.5. One of the most important, especially for the use in this thesis is a service called subscription.

Service	Type	Description
SubscribeCOV	C	controller subscribes to an object
UnconfirmedCOVNotification	U	value change at the object
ConfirmedCOVNotification	C	confirmed value change at object
AcknowledgeAlarm	C	alarm acknowledgment by a human operator
ConfirmedEventNotification	C	inform subscriber of possible errors
GetAlarmSummary	C	request a list of active alarms
GenEnrollmentSummary	C	request a list of possible errors

Table 3.5: Alarm and Event Services

Subscription

Instead of the controller addressing a certain device and reading its value, the controller can subscribe to the object. For this, a subscription request is sent from the controller to the object which then saves the address and ID of the controller if subscriptions are allowed. The advantage of this is that the device itself sends a service call to all its subscribers if internal properties such as the present value change. When this event is triggered is controlled by the optional property *COV_increment*. If, as an example, the value is set to 0.5, the controller is notified if the *Present_value* changes more than 0.5 since the last notification or read operation.
The subscription itself is a confirmed service, so the controller is certain that it is notified by the device. In the most cases, the event itself is an unconfirmed service i.e. the device does not check if the value change was delivered to the controller.
For use cases where the loss of values could lead to wrong decisions by the controller, confirmed COV notifications can be implemented.

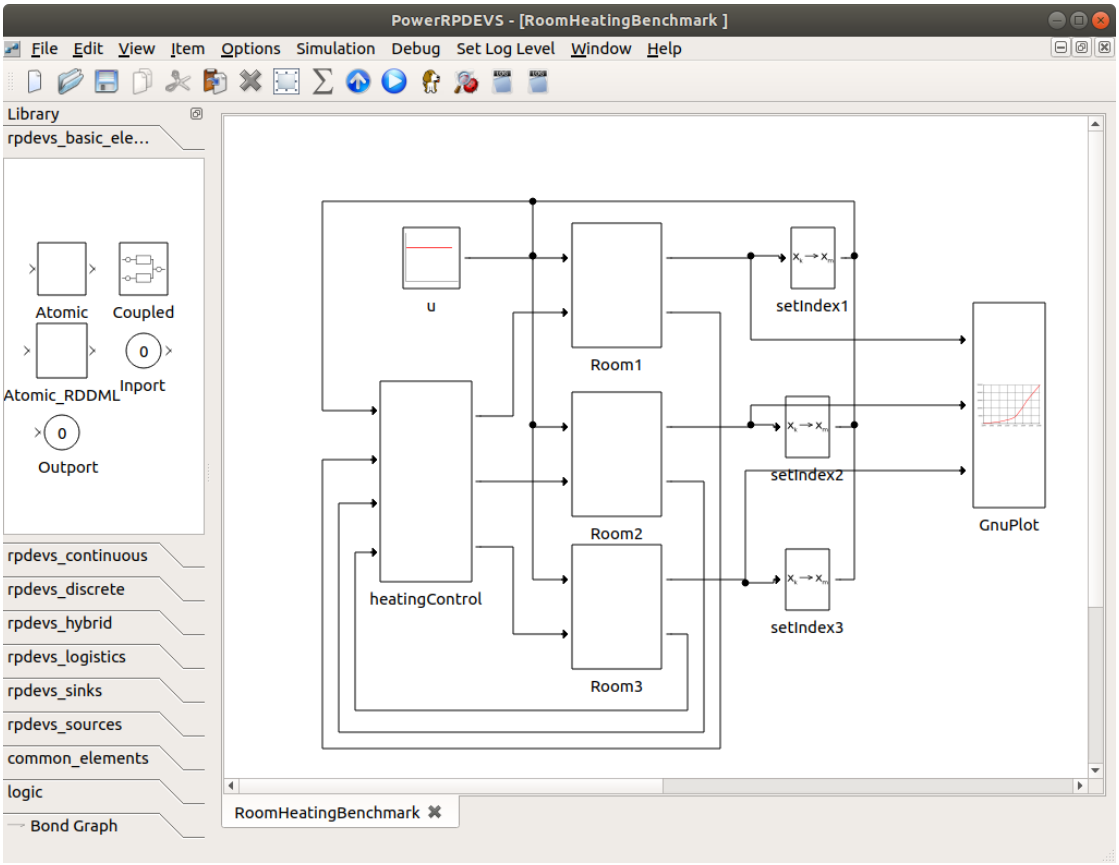


Figure 3.1: PowerRPDEVS editor, pdme

3.2 PowerRPDEVS

Building automation is a use case were by definition continuous processes collide with the digital world. Temperature development in a room, heat flow and changes in air quality for example must be detected and converted into digital values that can in turn be processed. Responses to these values have to be applied to actuator elements as digital or semianalog values (PWM) but then influence the continuous process.
Due to this behavior, PowerRPDEVS described in [PHK] was chosen as the simulation framework in this thesis. PowerRPDEVS, much like its predecessor PowerPDEVS is a hybrid system simulator, meaning that continuous processes can be simulated in combination with discrete ones. This capability makes it a good choice for the domain of BAS.

3.2.1 Operation

To simulate a process, a simulation model has to be created. For this, the graphical model editor called *pdme*, depicted in Figure 3.1 can be used.

The simulation model is constructed by combining blocks that implement a certain functionality. They can provide inputs, outputs or both. The action that is carried out by the block implementation, generates output due to the input and the current state. If only outputs are provided by the block it is called a source. Sinks only provide input ports. Between the input and output ports connection lines can be drawn. Although only a single line is displayed when connecting two ports of a block, they actually describes a potentially vectorial connection. Meaning that a single connection can transmit multiple datasets on different indices (channels) to the destination block. The model library offers *filter blocks*, which can be used to filter certain indices from a connection line. When the model development is finished, it can be compiled, resulting in a single executable model file. Additionally, a dialogue, shown in Figure 3.2, opens that provides certain options regarding the execution of the file.

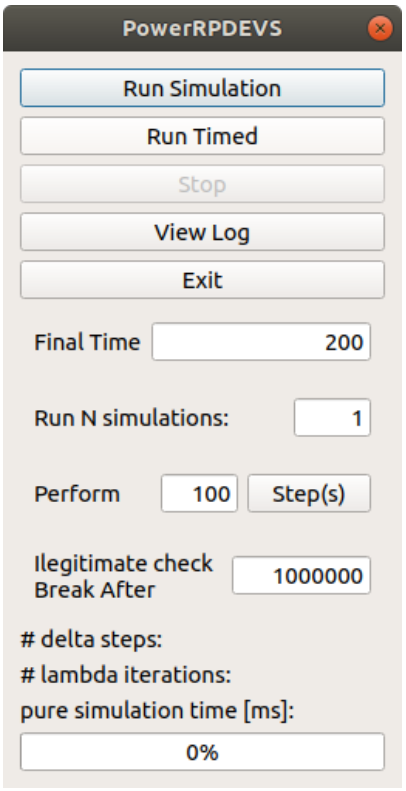


Figure 3.2: PowerRPDEVS simulation execution window

- Run Simulation
  Starts the model and runs the simulation until the final time. The final time can be set via the dialogue.
- Run Timed
  Starts the model and runs the simulation in real time. The final time can be set

via the dialogue. This mode is the mode used in this thesis, since temperature simulation and transmitting the data via UDP is only possible if the execution is not speed up.

3.2.2 Simulation Algorithm

The mathematical characteristics of the simulation algorithm are described in [PHK]. Important for this thesis is the operation cycle during the simulation.

When looking into a single block of the simulation, the operation can be divided into operation steps which are triggered by the entry of certain events. Figure 3.3 depicts the operation steps of a single block.

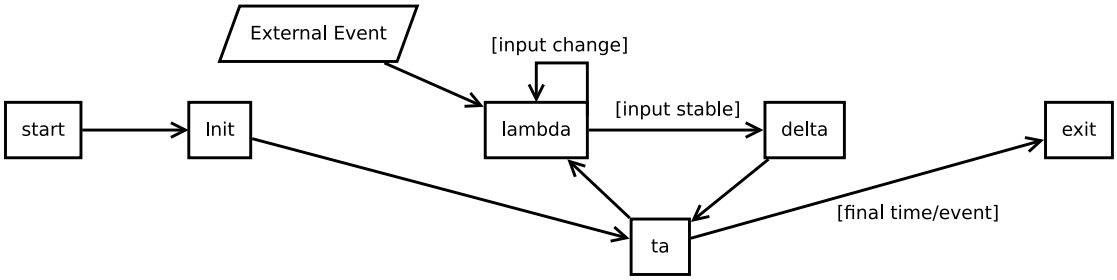


Figure 3.3: Schematic view of the PowerRPDEVS simulation operation steps

3.2.3 Init

Init initializes the block which means that state variables can be initialized. After the simulation start, the init operation is executed for every block.
Subsequently the ta (time advance) step is executed which calculates the time of the next internal event at the block.

3.2.4 Events

In PowerRPDEVS three types of events are defined:

- internal
  The elapsed time since the last event reaches the value of the time advance function of the block, which is the maximum time to live of the block's current state.
- external
  An input for the block occurred, because another block has written something on its output which is connected to the input of the first block.
- confluent
  Confluent is a combination of both previously mentioned events, in the sense that an internal event happens at the same time as an input.

3.2.5 Lambda

The lambda function is executed every time an event is triggered. It generates, in turn, the output of the block dependent on the current state and current input. With every change of the input due to lambda calculations of other blocks, lambda is re-calculated until the input is stable. Then the delta function is executed.

3.2.6 Delta

The delta function is executed if the inputs are stable and it calculates the new state of the current block. After that the ta step calculates the time to live of the new state giving the time of the next internal event of the block. If for all blocks, the time of the next internal event exceeds the final simulation time, the simulation quits.

3.2.7 Exit

If the final time of the simulation is reached, the simulation exits, causing the execution of the exit function in every block.

CHAPTER 4

Implementation

The implementation in this thesis will use PowerRPDEVS as the simulation framework. The *Simulation Backend* is part of the simulation engine. The communication with the controller is done via the BACnet bus. The *Simulation Frontend* will provide emulated devices to this network.

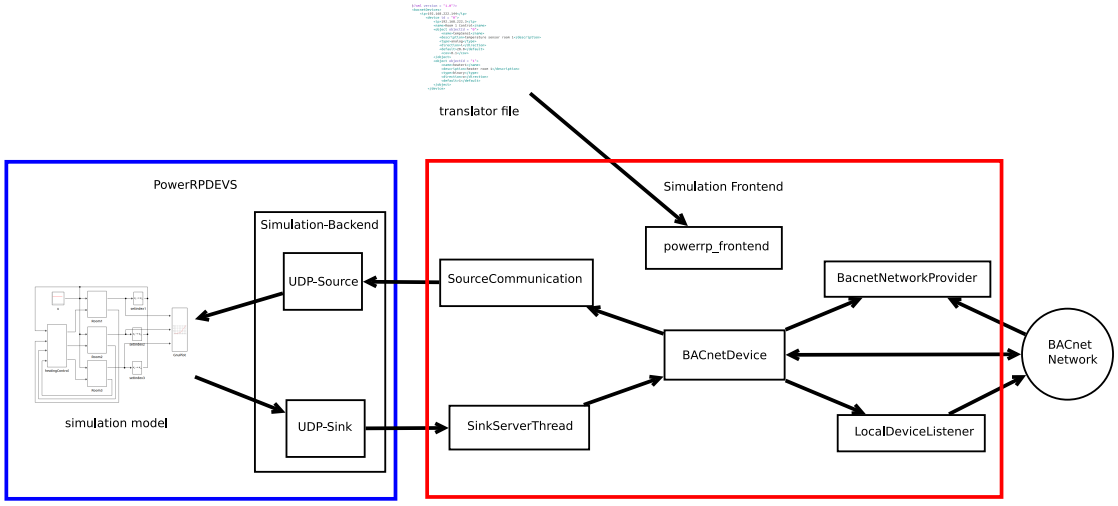


Figure 4.1: BACnet HIL implementation

The blue square encapsulates the components of the implementation that are part of PowerRPDEVS. The red square represents the *Simulation Frontend* and all its components that are implemented in JAVA.

4.1 Used Software

The following applications were used in this thesis.

4.1.1 PowerRPDEVS

So far, PowerRPDEVS can solely be installed on Linux and was tested only on Ubuntu. The download is available at [pow].
To be able to start the editor and use the software facilities included in the software, it has to be built first. This operation requires certain dependencies to be fulfilled. The required packages can be installed in Ubuntu with the following terminal command:

```
1 sudo apt-get install libprotobuf-dev libprotobuf-java protobuf-compiler pkg-config
    qt5-qmake qt5-default libqt5svg5-dev libboost-chrono-dev libboost-system-dev
    libboost-filesystem-dev libboost-libboost-thread-dev libhdf5-dev gnuplot
    gnuplot-x11 gtkwave kdbg
```

After the installation, the following commands install and setup the software:

```
1 make
2 cp ./bin/original.ini ./bin/powerdevs.ini
```

PowerRPDEVS can now be used by executing the *pdme* executable located in the bin folder of the installation.

4.1.2 IntelliJ IDEA Ultimate

IDEA Ultimate from IntelliJ is used to implement the parts of the software, that are written in JAVA. The software is available for free under an educational use license. To download the software, an account using a student mail address has to be created. After that every development software made by IntelliJ can be downloaded via the profile of the logged in user. An account can be registered using the following link [INT].
The IDE is available for Windows and Linux and offers features like internal git synchronization and the possibility to execute and debug the implementation using the included console.

4.1.3 TwinCAT2

TwinCAT 2 can be downloaded for free from the Beckhoff website ([TC2]), either a guest account has to be created or a permanent account registration is necessary. The user is offered two possibilities at the time of this thesis:

- TwinCAT 2.11 R3 (Build 2304)
  This package includes the engineering tool as well as the runtime. Note, it can only be installed on 32 bit architectures.

- TwinCAT 2.11 x64 Engineering (Build 2304)
  This package is optimized for 64 bit architectures but only includes the engineering tool. It is not possible for TC2 to execute the runtime on x64 systems. This version was used in the development process of this thesis.

The installation of TC2 adds the necessary drivers to the OS, by default it is registered as a startup service meaning that every time the OS is booted, a TC2 startup screen appears. The software is accessible via the quick menu in the bottom right of the Windows OS. A right click on the symbol offers the execution of the System Manager/PLC-Control, as well as RUN/STOP control over a connected Beckhoff controller.
The software can be used for free and in full extend during a 30 day test period. After that a registration is necessary. The process is described on the Beckhoff website ([BEC]).

4.2 Simulation Backend

PowerRPDEVS is implemented in C++. Preliminary to this work, the PowerRPDEVS model library already came with a UDP-Source and a UDP-Sink block. The UDP-Source block listens on a given IP address and on a given UDP port for arriving UDP packages. Arriving UDP packages are translated to either analog or digital values which are emitted into the simulation model via the output ports of the UDP-Source block. Similarly, the UDP-Sink block forwards messages which are received on its input ports on a configured UDP port to a configured destination IP address.

One goal of this thesis is to implement a Simulation Backend to exchange simulation data with the Frontend described in section 4.3. To reach this goal, the two blocks UDP-Source and UDP-Sink were modified and extended. The entire code can be found in the appendix, section 9.2. Since the software should be as modular and open for extensions as possible the library *protobuf* ([PRO]) is used to define and identify different message types. In the future, PowerRPDEVS could also be able to run on Windows systems. Socket operations are very dependent on the underlying OS. Thus, instead of C++ default libraries for socket communication, the boost.asio library ([BOO]) is used to abstract the OS layer from the implementation.

4.2.1 Libraries

protobuf

Protobuf was designed by Google with the goal to create a platform independent and language independent lightweight alternative to XML for serialized data. With protobuf, message formats can be defined in a file with the extension ".proto". Listing 4.1 shows a

part of the message format (see appendix, section 9.2.3 for the entire code) used for the UDP communication between Simulation Frontend and Simulation Backend.

```
// analog message format
message Analog
{
  optional int32 id = 1;
  optional double value = 2;
}

message CtrlCmd {
  enum CmdType {
    START = 0;
    STOP = 1;
  }

  optional CmdType cmd = 1;
}

message UDPMSG {
  oneof HIL {
    CtrlCmd ctrl   = 1;
    Digital digMsg = 2;
    Analog anMsg   = 3;
  }
}
```

Listing 4.1: Snippet of protobuf message formats

`Analog` describes the message format used to transmit analog input and output values. A distinction between in and out is not part of the implementation since this information is determined by the entity receiving the message.

BACnet output objects (actuators) are represented in the simulation by the UDP-Source. So every protobuf message that is sent from the Simulation Frontend to the Simulation Backend is interpreted as type output, since they act upon the simulated environment. The same applies for the UDP-Sink in the opposite direction, messages from the Simulation Backend to the Simulation Frontend are interpreted as input objects (sensory values from the simulated environment).
`CtrlCmd` represents an enumeration type with two possible values, `START` and `STOP`.

Message or variable properties are defined in protobuf with certain keywords:

- required
  A well formed message contains exactly one of this field.

- optional
  A well formed message contains none or maximum one of this field.
- repeated
  In a well formed message the field can repeat a number of times.

When the desired message types have been implemented in the .proto file, a protobuf makefile has to be executed. The result of the build process is a .cc file and a corresponding .h file. These source files provide functions to read, write, and manipulate the defined message types from a stream.

The implementation of the BACnet-HIL-Simulation requires the transmission of different kinds of messages across the same communication channels. Since UDP sockets are serialized, this can lead to problems due to different sizes of the messages. The payload of a digital message with boolean as its value is considerable smaller than of an analog message. Since messages are not terminated, the receiving application would need to know how many bytes it has to read for the next message.

The first part of the solution is a message type that describes a container class, UDPMSG in Listing 4.1. `Oneof` is an operator defined in protobuf. As the name suggests the only field contained in a particular message of type UDPMSG is one of the formats defined. This has the advantage that only one type of message has to be read from the socket and after reading, the type of message can be accessed via switch-case instructions.

As the length of the contained messages varies, the length of the whole UDPMSG changes as well. This is solved by transmitting the length of the message prior to the message itself. The byte size is therefore written to the transmit buffer as an `int32` and then followed by the message itself. The receiving end reads the `int32` message and then the number of bytes that was specified by that `int32` message.

boost

How sockets work in general and how they are setup stays the same across most operating systems and programming languages. Unfortunately, the abstraction level of the OS layer handling the socket communication itself, most of the time is not enough to completely ignore the OS in the C++ implementation. In Windows, a different C++ library has to be used than in Unix based systems. Windows requires winsocket.h, while under Unix, the default sys/socket can be used.
Boost libraries provide peer reviewed C++ libraries for free. They can be used for non commercial and commercial implementations and provide functionalities for nearly every aspect of C++ programming. PowerRPDEVS already uses the boost library for multi-threading and for UDP sockets. When using boost, all OS specific operations are handled by the library and no distinction has to be made. In addition, the libraries provide added functionality to the standard socket libraries. This includes resolving of IP addresses and ports to endpoints of the UDP connection, as well as asynchronous (non blocking) read and send operations.

4.2.2 UDP-Source

UDP-Source and UDP-Sink blocks are already part of the default PowerRPDEVS implementation. For this thesis they have been extended to function as the Simulation Backend. While the UDP-Sink can operate pretty much on its own in the real time aspect of the simulation, the UDP-Source needs special handling since the reception of data can happen at any given time asynchronously to the normal operation cycle of the simulation itself.

As described in section 3.2 at simulation start, the init function is executed for every model block. Its purpose is to read block parameter values and initialize state variables. In case of the UDP-Source, this includes the UDP port for the later setup of the UDP socket and reading the parameter which decides if simulation has to be started via a *CtrlCmd* UDP message. In this case, a simple blocking socket read is executed waiting for the corresponding START *CtrlCmd*. Since the simulation does not continue before the init function is completed at every model block, this blocks all further execution of the simulation until the *CtrlCmd* is received and the read operation returns. To then start data reception, an internal event is triggered, causing the execution of the delta function.
The purpose of the delta function is to create a UDP-thread that asynchronously receives data without being connected to the operation cycle of the simulation.

Thread-Handling

When the UDP-thread is started, it exists as a separate entity in the simulation framework. Processes in the thread do not block any simulation steps and the reception of messages is not bound to any timing of the simulation engine. At first, in the UDP thread a ready signal is transmitted at the UDP socket. For other applications listening at that UDP port, this indicates that the UDP-Source is ready to receive data. Then an asynchronous read on the socket is started and if a UDP-message is received, the data is written to a global memory. At this point, the data exists globally but has to be reintegrated into the timing of the simulation engine. To not overwrite or change the stored message, the thread closes after every single message reception.
Reintegration happens in the *root_simulator* of the model. As the name suggests it is the root of the tree representing the hierarchy of model blocks. If a simulation step is executed, the simulation tries to join the UDP-thread via the `try_join_until` function contained in the boost library. Joining a thread means that the calling process waits for the thread to quit. So that for the duration of the real-time step, the root node waits for the thread to finish. If it finishes or has already finished, a message was received and is ready to be read from the global buffer. If the thread has not finished until the time of the next simulation step, the simulation will continue with the thread as a separate entity in the background. By calling the time advance function of the UDP-Source block with a negative parameter value when a message was received, an internal event is triggered at the UDP-Source and the lambda function executes. Lambda handles the parsing of

the received message. Afterwards the delta function is called which will restart the whole thread cycle.

Message handling

Messages are read from the input buffer in the lambda function of the UDP-Source block. The buffer is converted into an input stream. From that input stream a delimited message can be extracted, by reading an `int32` from the stream followed by the number of bytes that was specified by that `int32`. The protobuf message in the defined format is extracted from the stream with the help of a parser function defined by the protobuf library. Since all messages are of type UDPMSG, a `switch case` is used. Listing 4.2 shows a part of this process, the entire code can be seen in subsection 9.2.1. The case values can be read from the .h file that is generated when the protobuf makefile is executed on the .proto file.

```
switch(udp_msg.HMI_case())
{
  case proto::UDPMSG::kCtrl:
    //control messages are handled in delta
  break;

  case proto::UDPMSG::kDigMsg:
    output = udp_msg.digmsg().value();
    output.t_exp = t;
    add_output(&output,(unsigned int)1, udp_msg.digmsg().id());
  break;
  .
  .
  .
}
```

Listing 4.2: Snippet of protobuf message handling

`add_output` is a function defined by PowerRPDEVS. The parameters are the value defined by the message that was read, the output port (1 is the second port of the block) and the id of the message. Since PowerRPDEVS connections are vectorial, this ID is the index of the value in the output vector.

4.2.3 UDP-Sink

The purpose of the UDP-Sink (see appendix, section 9.2.2 for the entire code) is to transmit the values arriving at its inputs to the Simulation Frontend. As opposed to the UDP server that is implemented in the UDP-Source, the Sink utilizes a UDP client. According to the RPDEVS simulation algorithm [PHK], due to the iterative lambda execution, the final (stable) input messages are only available in the delta function. Thus,

the transmission of the input messages over UDP is implemented in the delta function of the UDP-Sink block.

Message handling

The first step of transmitting messages over UDP is to create the socket and generate an endpoint pointing to it. This process can be done via the boost library.

```
udp::resolver resolver(io_serv);
udp::resolver::query query(udp::v4(), dest_IP, UDP_port,
   boost::asio::ip::resolver_query_base::flags());
receiver_endpoint = *resolver.resolve(query);
socket.open(udp::v4());
```

Listing 4.3: Boost endpoint resolving

The parameters of the query function shown in Listing 4.3 are populated in the init function. The values can be set via the PowerRPDEVS Editor by double clicking the UDP-Sink block. The snippet shown in Listing 4.3 is located inside a try-catch block. If the resolver fails, the messages are not sent to the specific recipient, but as broadcasts into the network.

For the messages themselves, the process is practically the same as with the UDP-Source in the other direction though. An output stream is created and the messages are encapsulated in the UDPMSG format. The size is evaluated and written to the stream, followed by the message itself. Finally, the output stream is written to the socket and the function finishes. For the detailed implementation see appendix, subsection 9.2.2.

4.3 Simulation Frontend

The purpose of the Simulation Frontend is to link the data from the simulation to emulated BACnet devices. The implementation is written in JAVA. While the Simulation Backend requires additional libraries to function on every OS, the JAVA-runtime handles any differences on the OS level providing a uniform execution platform.
To communicate with the Simulation Backend and provide BACnet devices, parameters have to be set accordingly. For this there are two ways, setting the parameters via command line or setting them via a translator xml file.

4.3.1 Command line parameters

When no parameters are given, certain default values are used.

- -i[PORT]
  The UDP port of the UDP-Sink block. Default: 2017

- -o[PORT]
  The UDP port of the UDP-Source block. Default: 2018

- -m
  M stands for mirror, a debug function that redirects all packages coming from the UDP-Sink to the UDP-Source. If this parameter is given, no BACnet devices are created.

- -t[URL]
  The location of the translator xml file, by default the BACnet folder environment on the BBB is used.

- -s
  The simulation model has to be started and stopped by transmitting START/STOP control commands.

- -e[URL]
  The location of the model file to be executed. By default the BACnet folder environment on the BBB is used.

- -noexec
  If this parameter is given, the application will not execute the simulation model.

The default values of these properties can be changed by editing the *frontendDefaults.porperties* file located in the resources folder of the source-code.

4.3.2 Translation

Data concerning the BACnet communication and the device setup has to be provided via a translator xml file. The location of this file is either the default directory of the BBB installation or provided via the command line. Listing 4.4 presents an example configuration of a single BACnet device with two objects.

```
<?xml version = "1.0"?>
  <bacnetDevices>
  <broadcast>255.255.255.255</broadcast>
  <ip>192.168.222.144</ip>
  <device id = "0">
    <name>Room 1 Control</name>
    <object objectId = "0">
      <name>tempSens1</name>
      <description>temperature sensor room 1</description>
      <type>analog</type>
      <direction>i</direction>
      <default>20.0</default>
      <cov>0.1</cov>
    </object>
```

```
    <object objectId = "1">
      <name>heater1</name>
      <description>heater room 1</description>
      <type>binary</type>
      <direction>o</direction>
      <default>1</default>
    </object>
  </device>
  </bacnetDevices>
```

Listing 4.4: translator xml file

All devices are capsuled inside of a `<bacnetDevices>` block. The first information that has to be given is called `<broadcast>` and `<ip>`. Here the IP address of the controller, the PLC in this thesis, and the broadcast address of the underlying network are entered. The number of devices is only limited by the number of free IP addresses in the network. Every device requires a unique *ID* across all given devices and all objects inside a device have to be unique as well.

Device properties are not required to be set since they are by default populated with values according to the Beckhoff PLC BACnet devices. All values can be changed by providing the name and value of the property in the translator file.
For objects, the user has to provide the object type, and the default value. Since BACnet provides objects with the same data type but different directions, the two parameters have to be set separately with the keywords `<type>` and `<direction>`. The `<cov>` (change of value) property has to be provided in order to be accessible via the subscription service. Failure to set this parameter leads to a BACnet error when subscribing to the object.

If required properties are not provided, or there was an invalid property, the application fails with a corresponding error message. Properties that are classified as optional by BACnet can be left out of the translator file. They remain empty or are populated with default values.

4.3.3 BACnet4j

The BACnet stack is handled by the *BACnet4j* library. It is a full implementation of the BACnet specification in JAVA. The version used in this thesis is provided by *infiniteautomation* on GITHub. Currently, Version 5.0.0 is available to download via [BAC]. Originally the library was written to supervise BACnet communication but since then evolved into a platform capable of fully emulating BACnet objects and functions as a controller itself connecting to remote devices.
The library can be included into the JAVA project by the means of *maven* in the *pom.xml* file, or via the source code directly. For this thesis the source code was copied into the project to fully understand and debug the underlying BACnet communication.
To store BACnet devices and all information that is associated with them, a `BACnetDevice`

class (see appendix, section 9.1.3 for the entire code) was introduced that serves as getter and setter not only for the objects, but also for other information that is necessary for the stateful execution of the HIL simulation. To store the objects a global HashMap called `localDeviceHashMap` is used.
A key component of the BACnet stack is the networking functionality within the IP network. When using the library, a default transport class has to be constructed. This class handles the underlying UDP-communication of the BACnet specification. Creation of the UDP-handler is shown in Listing 4.5. The entire interaction in the powerrp_frontend class can be seen in the appendix, subsection 9.1.1.

```
//a transport is created for the device linking it into the network
localDeviceHashMap.get(localDeviceID).setTransport(
new DefaultTransport(new IpNetworkBuilder()
.withReuseAddress(true)
.withPort(devicePort)
.withBroadcast(broadcastAddr.getHostAddress(),24)
.build())
);
```

Listing 4.5: Default transport generation of BACnet4j

BACnet communication can either happen as a direct communication (request/response) or as a broadcast, but every service is carried out on the same single UDP port. This creates a problem if the user wants to host more than one BACnet device on a single machine. UDP ports can only be used per IP address, meaning that only a single server can be opened. Similarly, only one client can be created. If there were multiple clients on the same machine (with the same IP) listening at the same UDP port, it is not given that every listening client receives the UDP packet. Additionally, the goal of this thesis is to emulate devices across multiple rooms, with every device having its own IP address.

4.3.4 Networking

UDP sockets can be bound to a specific IP address. Therefore, if the local machine can be accessed via two different IP addresses, one socket can be bound to one of the addresses and subsequently only receives UDP packets addressed to this IP address. Multiple IP addresses on a single network device are called virtual networks or in this case, only virtual addresses. Virtual addresses can be set up both in Windows and in Linux.

Under Windows the user can create a virtual address for every device and simply bind the default transport to it. Then, every other device in the network will be able to address it and BACnet communication will show different IP addresses for the devices.

Under Linux this is not possible. Virtual networks can be created via a single console command that extends a certain network adapter with a virtual address.

The problem is that UDP sockets are regarded as broadcast only, this means that by definition they should not be able to only listen on a specific IP address. Furthermore,

```
1 ifconfig eth0:bacnet0 128.130.56.110 netmask 255.255.255.128
```

this behavior is not wanted since it opens up the machine to *man-in-the-middle* attacks. A single machine could receive the packet and respond with the IP address of a different machine. This is the reason why under Linux binding broadcast sockets like UDP to a specific IP address is prohibited by the kernel itself.
Executing the application with address binding does not fail, the UDP sockets are generated and open. If a UDP broadcast of a different machine reaches the local one, the packets are received by the network adapter (this can be observed using *wireshark* [WIR]), but they are never forwarded to a socket. Instead, they are dropped silently in the background of the OS. This happens to both broadcasts and packets that are addressed for a single IP address. Due to this, a custom network adaption structure had to be implemented in this thesis.
The main goal is to access all devices (with different IPs) at the same UDP port. Therefore, the first step is to create a single UDP socket listening on this port.

BacnetNetworkProvider

The `BacnetNetworkProvider` (see appendix, section 9.1.4 for the entire code) is a single UDP socket listening on the main BACnet port. Its goal is to receive broadcast messages on the main IP address of the BBB and forward them to all individual devices. Inside the implementation every BACnet device communicates on a different port. So the `BacnetNetworkProvider` checks if a received UDP packet originated from the controller IP address and forwards it to every internal port of the implementation. Only broadcasts are received on the main IP address of the BBB. The check if they originated from the controller is necessary to avoid loops in the communication. A device is not supposed to react to its own transmission.

LocalDeviceListener

An identical transformation has to happen in the other direction. The *LocalDeviceListener*(see appendix, section 9.1.5 for the entire code) creates a socket listening for UDP packets on the internal port of a single device. So packets that are sent from the device to the network as response to a broadcast. Received packets are then forwarded via a unique output socket to the broadcast address with the BACnet port.

The whole interaction of the network adapter interface is shown in Figure 4.2. Lightning bolts indicate broadcast messages to the global BACnet port. Black arrows represent messages on the internal UDP ports of the devices.

The Simulation Frontend opens a single UDP socket listening on the BACnet port and two internal sockets for every device, one for input and one for output of the packets. With this setup BACnet devices can be discovered via the *WhoIs* service.

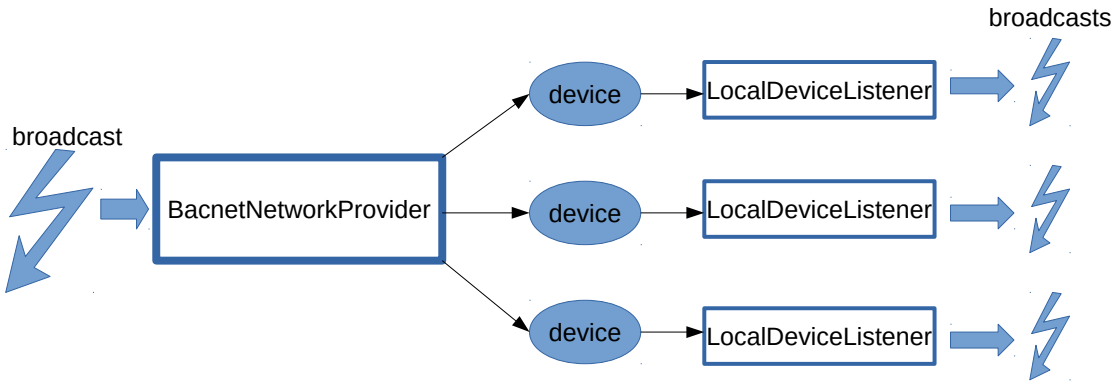


Figure 4.2: Networking for broadcast UDP packets

WhoIs is a broadcast service. Thus, it is received by the `BacnetNetworkProvider` and forwarded to every device. The devices respond with *Iam*, that is received by their specific `LocalDeviceListener` and then sent at the single BACnet port of the controller.
Thus, the controller discovers multiple devices with different IP addresses, but with different source ports. All packets were sent to the right port but originated from another port. This can lead to problems during further communication. Additionally, only broadcasts can be handled that way, because the `BacnetNetworkProvider` can not differentiate between a broadcast and an addressed packet since UDP cannot read the destination address of a packet. Further steps are needed for full emulation:

Linux iptables

The firewall under Linux can be configured by the user using so called tables. Inside these tables are routing configurations or rules about what should happen to a specific packet type, or what should happen if it originates from a specific address. However, these tables can also be used to change parameters of a packet at certain places of the packet handling, marked gray in Figure 4.3.

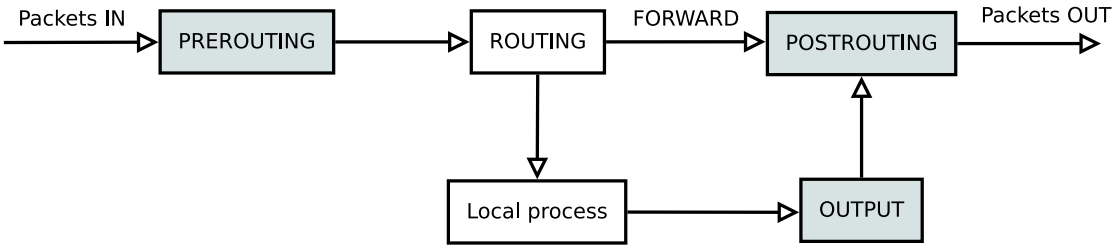


Figure 4.3: Packet routing under Linux

Packets are received and run through the *prerouting* instructions of the firewall. *Routing*

either forwards the packet or connects it to the local process. This connection to the local process is what prohibits UDP sockets to be bound. Outgoing packets are run through the *postrouting* instructions and are then sent via the network adapter. To emulate the devices, it is necessary to add certain rules to the pre and postrouting tables.

PREROUTING

With prerouting instructions, parameters of incoming packets can be changed before they are routed through to a specific port. This can be used to redirect incoming packets directly to an internal port of a device as opposed to the *BacnetNetworkProvider* that is necessary for broadcasts. An example is as follows: The single line command adds a

```
1 iptables -t nat -A PREROUTING -p udp -d 128.130.56.110 –dport 47808 -j
    DNAT –to-destination 128.130.56.79:47824
```

new instruction to the PREROUTING table. The parameters in order of occurrence correspond to the following.

- -t NAT
  The table to which the command belongs to, NAT stands for Network Address Translation.
- -A PREROUTING
  The chain element that is supposed to execute the instruction.
- -p udp
  Only udp protocols are regarded for the instruction
- -d 128.130.56.110
  The destination address of the packet. Unlike with the socket implementation in JAVA at this point the UDP packet is still encapsulated in the network layer, so the destination address can be read.
- -dport 47808
  The destination port of the packet.
- -j DNAT
  The action that is performed on the packet, DNAT stands for Destination Network Address Translation.
- –to-destination 128.130.56.79:47824
  The new destination address of the packet.

So the iptables instruction does the following. If a UDP packet is received by the machine and its destination is the address of a device (virtual address) with the global BAC0 port,

the packets destination is changed to the IP address of the main network adapter hosting all UDP sockets and the internal port of the specific device. The packet is therefore routed directly to the UDP socket of the BACnet device without changing the sender address.

POSTROUTING

The devices respond to service calls by addressing the caller. So the packet is always sent to the address that initiated the service in the first place which means that although the devices are created on a different port, responses are always sent to the BACnet port since the call originated there. Broadcasts that were forwarded from the `BacnetNetworkProvider` are therefore sent to 127.0.0.1 and are picket up by the `LocalDeviceListener`. Packets that went through the prerouting rule have the IP address of the controller. The only thing left to do is to mask the source address and port of the outgoing packets, so the controller sees different IP addresses on a single port. This can be done via the following command.

```
1 iptables -t nat -A POSTROUTING -p udp –sport 47825 -j SNAT –to-source
    128.130.56.110:47808
```

- -t NAT
  The table to which the command belongs to, NAT stands for Network Address Translation.
- -A POSTROUTING
  The chain element that is supposed to execute the instruction.
- -p udp
  Only udp protocols are regarded for the instruction
- -sport 47825
  The source port of the packet.
- -j SNAT
  The action that is performed on the packet, SNAT stands for Source Network Address Translation.
- –to-source 128.130.56.110:47808
  The new source address of the packet.

Then all UPD-packets that are scheduled to be transmitted by the network adapter are checked if they originated from the given source port and if this is the case, the source address is changed to the address of the device (virtual address) and the BACnet port.

4.3.5 Frontend-Backend Communication

Backend to Frontend communication corresponds to an update of the present value of a certain device, while the opposite direction corresponds to writing to a device by a remote device.

Backend to Frontend

This direction of the communication is handled by a thread called `SinkServerThread` (see appendix, section 9.1.6 for the entire code). It provides a UDP server that awaits packages from the UDP-Sink of the PowerRPDEVS simulation. As in the Backend, the packet content is converted into a stream. Unlike the C++ library of protobuf, the JAVA version offers a function called `parseDelimitedFrom`, that directly reads the delimited message from the stream. Handling the message can happen in two different ways, with or without mirror mode. If mirror mode is active, the message is directly sent to the UDP-Source of the PowerRPDEVS simulation. If mirror mode is not active, the message is extracted and the value is written to the present value of the device/object specified by the ID(which is part of the message). The device and the object are addressed in the following format [DEVICE_ID][OBJECT_ID]. Thereby, the values are always 3 digits filled with '0', for example device 1, object 1 is identified in PowerRPDEVS with index 001001.
This will generate a BACnet subscription event if the change of the present value is greater as the defined COV value.

Frontend to Backend

Communication from the Frontend to the Backend is necessary when a remote device writes a value to an object. If this object is of type output the value has to be sent to the UDP-Source of the PowerRPDEVS model. Services like `writeProperty` are already handled in the background by the BACnet4j library. To receive a notification as well as the data when a write is carried out, the `BacnetDevice` class that is already used to store the device data, extends the `DeviceEventHandler` of BACnet4j. The library by default executes functions for certain events like *WhoIs* receptions or property writes. By deriving from the `DeviceEventHandler`, the functions can be overloaded and custom code is executed. In the case of this thesis the `propertyWritten` function is overwritten.
From the object the ID of the encapsulating device as well as the object ID is extracted and the PowerRPDEVS id is generated. If the object is of type output, the value and the ID are composed to an analog or a digital message and subsequently encapsulated in a UDPMSG.

Due to the thread synchronization process of the UDP-Source implementation, sending the messages as fast as they appear can lead to a lot of dropped packets, since the thread listening might not be running. To eradicate this problem, a buffered sender class called `SourceCommunication` (see appendix, section 9.1.7 for the entire code) was

implemented. Messages that have to be sent are given to the class and are stored there inside a send queue. Elements can be sent to the UDP-Source if the ready message is received on the control port. After that, all elements of the queue are combined into a single message and sent at once. With that the UDP-Source is considered busy and new messages wait for the return of the ready message.

4.3.6 Model Execution

To close the Frontend and the simulation a prompt will be presented to the user.

4.3.7 Simulation

If the noexec parameter is not given when executing the Frontend, the model located in the given or default directory is executed. With this execution, the model is bound as a child process to the current process of the Frontend, meaning that the execution of the simulation is bound to the execution of the parent process. When the setup process is done and the model is running in the background, the user is prompted a simple user interface allowing him to start the simulation (i.e. sending the START *CtrlCmd* to the UDP-Source) or to leave and close the Frontend. If the UDP-Source is configured to wait for the signal, the GnuPlot window will open but no data is generated. If this option is not active, the simulation will start immediately. This behavior allows for two different approaches:

- PLC programming
  To configure the hardware of the Beckhoff PLC controller, the BACnet devices have to be discovered. This process and the subsequent programming of the PLC program, do not require the simulation to be running. So during this process the Frontend can be started with the *noexec* parameter. This means that when the setup is done all devices with their default values are fully capable to interact with the controller, but no resources are used for the simulation. When the setup is done, the Frontend is closed with 'e' (exit).

- Simulation (HIL-Testing)
  The Frontend is executed and a model is started. When the prompt is presented to the user, the hardware under test can be switched to running. At this point the default values of the devices and objects can be read. When the hardware is ready, 's'(start) is entered and the simulation is started by issuing the START command to the UDP-Source. When enough data is captured, the simulation can be stopped with 'e' (exit), the Frontend will close and the simulation data will be available in the output folder of the BACnet installation on the BBB.

By entering 'e' (exit) in the prompt, all sockets are closed and the STOP *CtrlCmd* is sent to the simulation model.
Forcing the Frontend to close will also stop the execution of both processes. However

since no STOP *CtrlCmd* is sent, the simulation will not exit gracefully. Parts of the simulation can stay active, i.e. if GnuPlot is used the window will not close and has to be closed by force by the user.

CHAPTER 5

Hardware

5.1 BeagleBone Black

The BBB (BeagleBone Black) is a credit card sized mini computer running an AM335x 1GHz ARM® Cortex-A8 processor. It is designed as a development station for embedded applications or monitoring applications in the private as well as the industrial sector.

5.1.1 Processor and Memory

An AM335x 1GHz ARM® Cortex-A8 processor featuring an 32-Bit Armv7 architecture, is the brain of the BBB. Processors such as this one are considered as the future of mobile devices because they can be produced in a small package and operate with a low power consumption. In combination with the 512Mb RAM the BBB is well suited for console applications while other comparable mini computers such as the Raspberry Pi 3 which features a quad core processor with 1024Mb RAM, are more suitable for graphical applications.
The OS and user data are stored in a 4Gb eMMC (embedded multi media card). This type of memory is comparable to SD-cards. They can be produced cheap and in small packages making them well suited for the use in mobile devices such as mini-computers, smart-phones etc. Since the OS data alone needs around 3.5Gb, there is little space left for custom user software. To eliminate this problem, a microSD card can be equipped. The OS memory space can then be extended onto the card or it can be used as an external mass storage.

5.1.2 Interfaces

Figure 5.1 depicts the BBB and locations of the interfaces that are available.

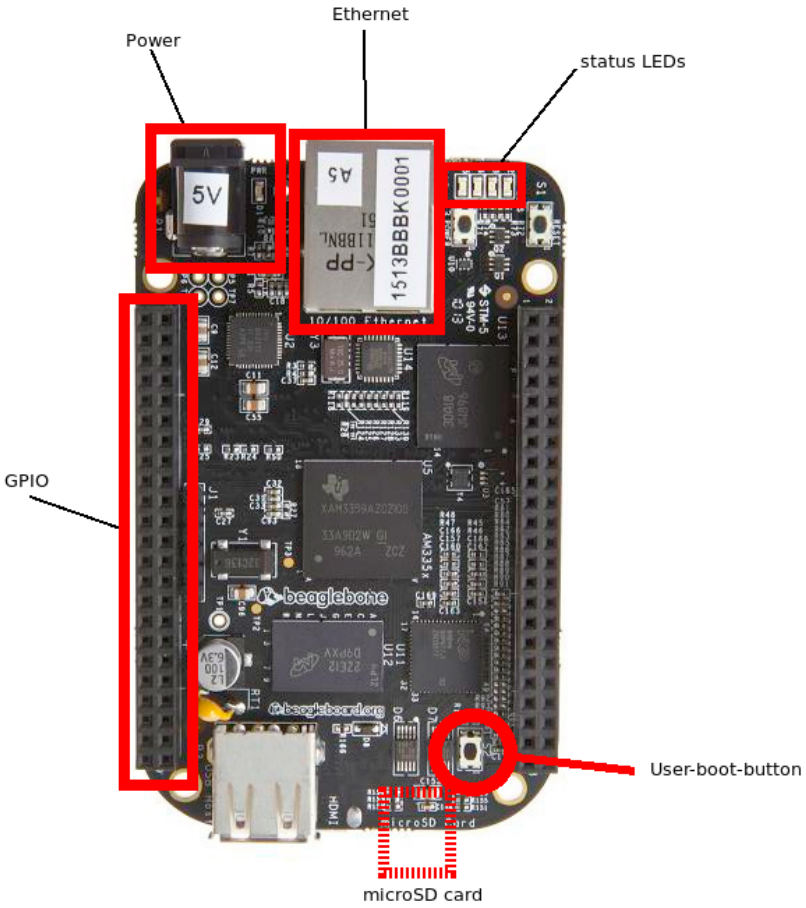


Figure 5.1: BeagleBone Black, picture taken from [BBB]

Power

Power can be provided to the BBB via two interfaces, the 5V low power jack, or the miniUSB interface.
For quick setup of the BBB, USB power can be provided directly from the host PC. Due to the limitations of the output to just 0.5A, this can lead to power problems and crashes when deploying resource heavy applications or using other IO-interfaces such as GPIO and USB.
For longer operation and after the setup, power should be provided by an external 5V power supply with at least 2A.

USB

Two different USB interfaces are present:

miniUSB The miniUSB interface is implemented as a USB-host and provides a quick setup interface. When plugged into a host PC, the BBB shows up as a read only mass storage device. Containing setup information as well as network drivers. When these drivers are installed on the host PC, the USB connection acts as a network over USB emulating an Ethernet connection. By default the IP address of the BBB is 192.168.7.1 while the host PC is assigned 192.168.7.2. Additional addresses are not assigned.

USB The USB-port is implemented as a USB-client and can be used to connect storage or control devices to the BBB. When powering external hardware via this interface, it is important to keep power consumption to a minimum. The 2A of the power supply or the even less 0.5A of the miniUSB port need to be divided between the processor, internal hardware and the external hardware connected to the USB port. When using multiple devices such as a mouse and keyboard it is suggested to use a USB hub with an external power supply.

Ethernet

A RJ45 port provides 10/100Mbits/s network access to the BBB. When connected, the interface acts as the primary network adapter. The two LEDs display the power state (green) and ongoing communication (orange).

HDMI

Via the integrated miniHDMI-port, displays can be connected to the BBB. By default, full size BeagleBone distributions boot into the command line. However, an X-server is installed and after login, the BBB can switch into a desktop environment. As discussed in subsection 5.1.1, the BBB is not suited to run resource heavy applications in addition to hosting the X-desktop on HDMI.
For use in IoT or strictly embedded application cases, smaller OS distributions can be installed which do not implement an X-server, so only the console can be displayed via the HDMI-Port.

GPIO (General Purpose Input Output)

92 GPIO pins can be accessed via two connection headers on either side of the BBB. These pins provide simple GPIO operations such as input and output to interact with the hardware, as well as SPI, UART and I2C bus communication. Additionally, they can be used as interfaces for extension boards called capes. The cape function set can range from displays to motor drivers. They are provided directly from the vendor of the BBB, as well as by a number of public community projects.

5.1.3 Operating System

As an operating system, a Debian Linux distribution is used. The image itself is provided by the BeagleBoard Foundation and is tailored specifically for this application. Meaning

that certain default services for easy setup are already onboard at the time of purchase. The user can update the operating system by downloading a new image and burning it onto a microSD card. The newly burned image can be used to flash the internal eMMC memory of the BBB overwriting the old image in the process, or as the default boot medium keeping the old image untouched.
After a boot phase, LED3 of the board starts blinking in a heartbeat pattern and the BBB is ready to be accessed either via USB Ethernet, Ethernet or it can be used as a desktop PC if a display is connected to the HDMI-port.

Users

If the preinstalled image is used, two users are present:

- root
  The super user of the system. This account is not protected by a password.
- debian
  This is the default user of the system with password: temppwd

If an image is downloaded from the BeagleBoard Foundation and installed, only the user called *debian* is present. There is no root account. If the user requires root permission, *sudo* has to be used with the password of the *debian* user.

5.2 Beckhoff...Hardware under Test

To interface with BACnet devices, most PLCs require either a specialized hardware module, special software or are specific BACnet controllers themselves. The choice in this thesis was to either use a DESIGO Siemens controller or a Beckhoff PLC. DESIGO is a special branch of Siemens, that is focused on building automation. Although the controller is BACnet ready, it is focused on communication with other DESIGO devices. The Beckhoff CX5010-1111, on the other hand, is a PLC controller without specialization. BACnet support can be installed via an image that is burned onto the memory card. To use the functionality though, a license has to be purchased which is bound to one of the two Ethernet-adapters on the main body of the controller. The advantage of the Beckhoff PLC is that it is not locked by any vendor restrictions in the communicating with other BACnet devices. Due to this fact and the fact that the engineering software (TwinCAT) can be sourced for free from the producer website, the CX5010 was chosen as the hardware under test in this thesis.

The Beckhoff PLC framework consists of two main parts, the engineering part, that handles the PLC programming as well as the hardware setup and the runtime that executes the configuration generated by the engineering process.
Due to this separation, a PLC program is highly portable and can be deployed on nearly every processor or hardware. When downloading the software, the user can choose to

either download the engineering software as a stand alone or in combination with the runtime.

5.2.1 Engineering

The engineering tool for Beckhoff devices is called TwinCAT and comes, at the time of this thesis, in two main versions, TC2 (TwinCAT2) and TC3 (TwinCAT3). The version needed is determined by the Beckhoff controller. Both versions are available for Windows only.
TC2 was implemented as a pure 32 bit application. In order to install the software on a 64 bit architecture, a separate package has to be installed. Due to this, it is not possible to install the runtime on a 64 bit OS. TC3 for 64 bit includes the engineering tool, as well as the runtime.
The CX5010 used in this thesis can only be engineered with TC2. Therefore, the following sections are focused purely on this version.

System Manager

The System Manager handles the hardware configuration of the PLC, as well as the connection between hardware and software. The software outputs a set of instruction set that tells the PLC which hardware modules are connected, the configuration of these modules as accessible IO devices, as well as the linking between these devices and the PLC program. To add modules to the system, either the real PLC is used as a scanner that detects connected modules, or the management is done offline on the local system at first and is later deployed on a target hardware. With the runtime installed, the scanning process can also be done on the engineering PC itself. For this purpose, the modules have to be connected to the PC.
If a program is created with the PLC-Control (described in the next section), a configuration file containing global variables is created that is then loaded into the System Manager. These variables then become hardware dependent by mapping them to IO modules of the same type.

When the System Manager is connected to a hardware that is hosting a runtime, the configuration can be activated, this means that three separate configurations are transmitted to the controller, the hardware info, global variables contained in the PLC program and the mapping between these two. The PLC program itself is not included. Activating a configuration purges everything that was prior installed on the controller. Flashing the program is handled by the PLC-Control.

PLC-Control

While the System Manager handles the hardware, the PLC-Control is used to implement the functionality of the controller. In that sense, it works like other comparable engineering tools for PLCs. The user can load libraries or use the default functions to program POUs

(Programming Organization Units) in the languages typically defined by the IEC 61131 (IL, LD, FBD...).
Since this application is a separate block of software, the order of operation can be chosen by the user, meaning that the hardware can be configured before or after the PLC program is written. An advantage of configuring the hardware first is that the IO modules can be exported from the System Manager and then imported into the PLC-Control. When programming the PLC, the generated variables can be used without having to define them manually. The mapping in the System Manager can then be generated automatically.

When the implementation is finished and the configuration is activated via the System Manager, the PLC is runable and can be flashed with the PLC program. With the activation all previous programs are purged from the system. Flashing is done by the means of the *Login* menu of the top menu. When the user logs into the runtime, either the whole implementation or just changed parts are written to the PLC controller. A new program always starts in mode STOP, if RUN is activated, the software starts getting executed and the current values are displayed in the PLC-Control.

5.2.2 Runtime

The runtime is the executing entity in the Beckhoff system. It is a software that sits on top of an existing OS and abstracts certain behavior for the software that is executed, much like a JAVA runtime. The advantage of this system is that the execution of the PLC is not strictly limited to a single processor or OS configuration. Beckhoff PLCs themselves are sold with different OS installed. The CX5010 in this thesis is running on a Win CE6 OS, while a CX1010, for example, is hosting Win XP.
The abstraction of the runtime also means that not necessarily a PLC controller has to be used to execute the hardware configuration and PLC program. Under certain circumstances both can be installed and run on the engineering PC itself. For this to work, the TC2 runtime requires a 32 bit architecture. If this is given, the configuration by the System Manager can be activated with the local system as target hardware and the same is possible for the PLC-Control. Implementations can therefore be tested by manually applying values to the mapping and observe the effect in PLC-Control, or by actually connect hardware to the engineering PC itself. For certain modules like binary and analog inputs/outputs, this might not be possible. On the other hand, bus communication like BACnet used in this study, can be connected to a network adapter of the PC, provided that the adapter is made by Intel due to a limitation in the driver.

5.2.3 CX5010-1111

The CX5010 is an embedded PC produced by Beckhoff with Win CE6 OS running on an Intel-Atom Z510 processor. Additional information about the capabilities of the controller is given by the number extension following the product identification. In the case of this thesis, the numbers are 1111 [abcd] having the following meaning:

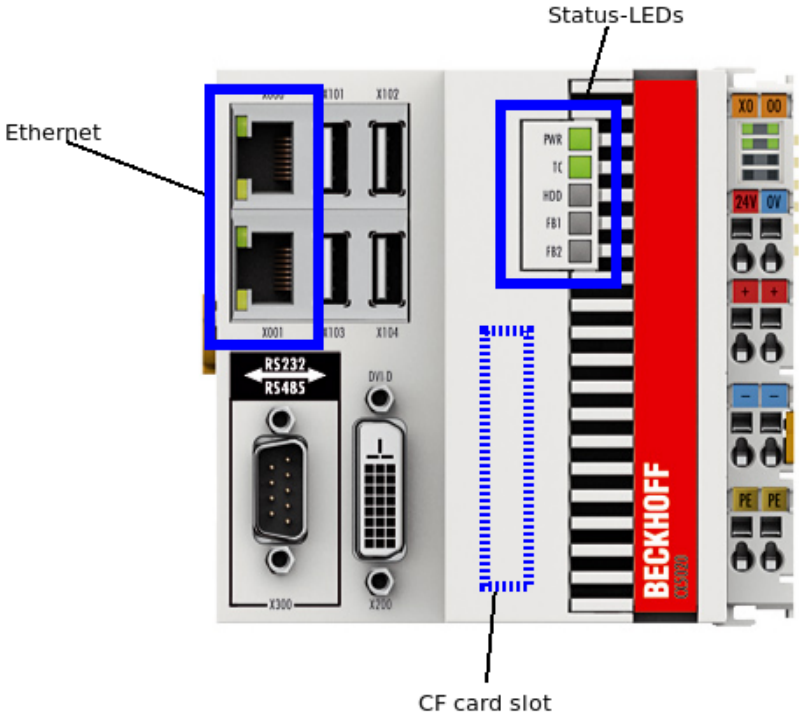


Figure 5.2: Beckhoff CX5010, picture taken from [CX]

- a:1
  K-Bus (Kommunikationsbus) interface for bus terminals. This means that IO modules can be attached to the controller body.
- b:1
  b is always 1 for the CX5000 series of controllers since they do not offer any options, unlike other PLC controllers.
- c:1
  Operating system is Windows Embedded CE Version 6
- d:1
  TwinCAT 2 PLC Runtime

Interfaces

The following interfaces are used in the thesis. The location of the interfaces can be seen in Figure 5.2

Ethernet On the left side of the controller body, two RJ45 Ethernet ports are located. These interfaces provide 10/100/1000 Mbit/s Ethernet communication to the controller.

It is important to note that the two ports are strictly separated and not connected by an internal switch like on other CX type controllers offered by Beckhoff. By default, port 1, the lower port, is configured as an EtherCAT adapter, a special communication protocol used by Beckhoff. Due to this, the controller can be accessed by TwinCAT straight out of the box. IP addresses have to be either provided via DHCP from the network connected to the controller, or a special module is added in the System Manager to provide DHCP from the controller to the network.

LEDs LEDs on the front provide visual feedback to the user in order to identify the current state of the controller. From top to bottom the meanings of the LEDs are:

- Power
  Green if power is connected, off otherwise
- Mode
  blue for CONFIG mode, green for RUN and red for STOP
- HDD
  Write or read operation on the internal memory card (CF card)
- FB1
  Status of fieldbus
- FB2
  Status of fieldbus

CF-Card (Compact Flash) The memory containing the OS and the PLC program, is provided by a CF card that has to be inserted into the CF card slot on the front of the controller. The interface is protected by a latched cap.
Via this card, the OS can easily be updated or configured if it is inserted into an engineering PC.

BACnet The CX5010 is one of a range of controllers offered by Beckhoff that provide BACnet support. While the embedded PCs can be purchased and operated without BACnet, the support software can be purchased as a supplement that is then installed on the CF-card in combination with the OS. Important to note is that the use of the BACnet functionality requires a special supplement license which is bound to a certain MAC address. Meaning that in case of this controller with two Ethernet ports, only one can be used for BACnet or a second license has to be purchased for the second port.

5.3 Hardware Setup

For the implementation of this thesis to work, the BBB has to be set up accordingly. Further, the Beckhoff CX5010 has to be able to use BACnet revision 12. In order to setup both systems, the following steps are necessary:

5.3.1 BeagleBone Black

By default an old version of the BeagleBoard Debian distribution is installed. While it can be used for most applications, the older version lacks support for certain appliances like displays or BeagleBoard capes (extension boards). So a good practice is to install a new OS on the board.

Installing a new operating system

Old and new BBB distribusions can be found under the following link: [BEA]. In this thesis, version 'Debian 9.5 2018-10-07 4GB SD LXQT' is used. By clicking on the version of choice, a .tar containing the image can be downloaded to the local machine.
The extracted image must be burned onto a micro-SD card. To accomplish this under both Windows and Linux, the software *Etcher* can be used (available: [ETC]). To burn the image, the microSD card is inserted into the PC. *Etcher* will recognize every drive connected to the PC so it is imperative to choose the right device to burn to, as the first step of the software is to format it without any further checks. After the burn process is completed the user can use the burned image and card in two different ways:

- As a boot medium
  When the microSD card is inserted into the BBB and power is connected while the user boot button is pressed, the board will run of the microSD card instead of the internal eMMC memory. This operation mode can be useful to develop a working OS version with additional functionality and then distribute the image by reading it from the microSD card. Other BBB can then be flashed with this OS version from the microSD card.

- As a flash medium
  The most efficient way to run a OS on the BBB is the internal eMMC memory of 4GB. To be able to flash the OS to the internal memory, a command, indicated by a comment "#enable BBB: eMMC Flasher:", has to be uncommented in the /boot/uEnv.txt file. This can either be done by running the BBB from the microSD card and editing the file via ssh from another PC, or by inserting the card into a Linux PC and editing it directly.
  When inserting the card and connecting power while pressing the user boot button, the flash process will start, this is indicated by a running light displayed on the LEDs. The flashing will be done after 10-45 minutes. It is important to remove the microSD card afterwards to prevent the BBB from reflashing with every boot.

BACnet HIL Setup

Since the BBB is designed as a development environment for every level of programmers, there is a lot of overhead included in the 3.5Gb default distribution. This data has to be removed since the dependencies of the PowerRPDEVS simulation framework do not fit into the 500 Mb that are left.

The following commands remove unnecessary overhead from the BBB:

Uninstall remote desktop and the apache2 webserver as well as all unused dependencies.

```
1 sudo apt-get purge xrdp
2 sudo apt-get purge apache2 apache2-mpm-worker apache2-utils apache2.2-bin
    apache2.2-common
3 sudo apt-get autoremove
```

Uninstall and delete all data of the cloud 9 web based IDE.

```
1 sudo systemctl stop cloud9.service
2 sudo systemctl stop cloud9.socket
3 sudo systemctl disable cloud9.service
4 sudo systemctl disable cloud9.socket
5 sudo rm -rf /var/lib/cloud9
6 sudo rm -rf /opt/cloud9
7 sudo rm /etc/default/cloud9
8 sudo rm /lib/systemd/system/cloud9.*
9 sudo systemctl daemon-reload
```

Uninstall and delete all data of the bonescript programming language.

```
1 sudo systemctl stop bonescript-autorun.service
2 sudo systemctl stop bonescript.service
3 sudo systemctl stop bonescript.socket
4 sudo systemctl disable bonescript-autorun.service
5 sudo systemctl disable bonescript.service
6 sudo systemctl disable bonescript.socket
7 sudo rm /lib/systemd/system/bonescript*
8 sudo rm -rf /usr/local/lib/node_modules/bonescript
9 sudo systemctl daemon-reload
```

Delete source code of 3rd party example applications and kernel patches that are already

installed.

```
1 sudo rm -r /opt/sources
```

Uninstall avahi daemon (useful for webservers) and chromium browser

```
1 sudo apt-get purge avahi-daemon
2 sudo apt-get purge chromium-browser
```

PowerRPDEVS

At this point the dependencies for the PowerPRDEVS simulation framework can be installed via the following command:

```
1 sudo apt-get install libprotobuf-dev libprotobuf-java protobuf-compiler pkg-config
    qt5-qmake qt5-default libqt5svg5-dev libboost-chrono-dev libboost-system-dev
    libboost-thread-dev libhdf5-dev gnuplot gnuplot-x11 gtkwave kdbg
```

The PowerRPDEVS model editor is not runnable on the BBB, installing the dependencies although enables the BBB to execute the Makefile and downloads and compiles all necessary ARM libraries. The PowerRPDEVS installation itself is too big to fit into the internal memory, due to this a microSD card formatted as ext4 can be used as a external memory. When inserting a microSD card into the BBB or rebooting it, the memory is not automatically mounted. To configure the BACnet HIL directory structure for the Simulation Frontend as well as the simulation itself, the following steps are needed:

```
1 mkdir /media/BACnet
2 sudo mount /dev/mmcblk0p1 /media/BACnet
```

The first command creates a directory and only needs to be executed once, the second one mounts the content of the microSD card to the newly created folder.

PowerRPDEVS can be downloaded into this folder via the following reference, [pow].

JAVA

To execute the Simulation Frontend, a JAVA runtime has to be installed. This is accomplished by downloading the runtime from the oracle website. The version used in this thesis is the jdk-8u211-linux-arm32-vfp-hflt runtime that can be downloaded from [JAV]. To save internal memory, the archive is extracted into the /media/BACnet folder.

For executing a JAVA application, the OS needs to know that the runtime included in this folder has to be used when the command 'java' is entered. This is achieved via the following commands.

```
1 sudo update-alternatives –install /usr/bin/java java
    /media/BACnet/jdk/jdk1.8.0_211/bin/java 100
2 sudo update-alternatives –install /usr/bin/javac javac
    /media/BACnet/jdk/jdk1.8.0_211/bin/javac 100
```

To check if the installation was successful, the following command should result in a similar response.

```
1 java -version
2 java version "1.8.0_211"
3 Java(TM) SE Runtime Environment (build 1.8.0_211-b12)
4 Java HotSpot(TM) Client VM (build 25.211-b12, mixed mode)
```

5.3.2 Beckhoff CX5010

The PLC controller used in this thesis is already integrated into a laboratory environment. Due to the age of the device, an old runtime version (2.11.2224) was still installed. When connecting to the controller via TwinCAT 2, old programs were able to be executed. When a BACnet device was added, the PLC crashed, the mode was stuck in CONFIG and no further actions were possible until the PLC was flashed with a clean, empty project.
The reason for this problem was that although the runtime installed is BACnet ready and the functionality was enabled, only BACnet revision 6 devices were accepted. The solution was an update of the runtime to a newer version.

Much like the BBB, the OS of the Beckhoff CX5010 runs of a memory card, a CF-card in this case. To update the whole OS and with that the runtime, the CF-card is connected to a PC. Old data can be backed up by copying the content from the card. To make sure that no content of the old OS interferes with the new one, the card has to be formatted. The new image can be sourced from the Beckhoff support and is installed by copying it onto the card.
At first boot with the new card, the controller configures and takes approximately 5 minutes to switch into the running mode indicating that the PLC is ready.

CHAPTER 6

Build-Process and Deployment

The implementation in this thesis is deployed on two devices. A BBB hosts the Simulation Backend and Simulation Frontend, while the CX5010 PLC controller functions as the hardware under test, as shown in Figure 6.1.

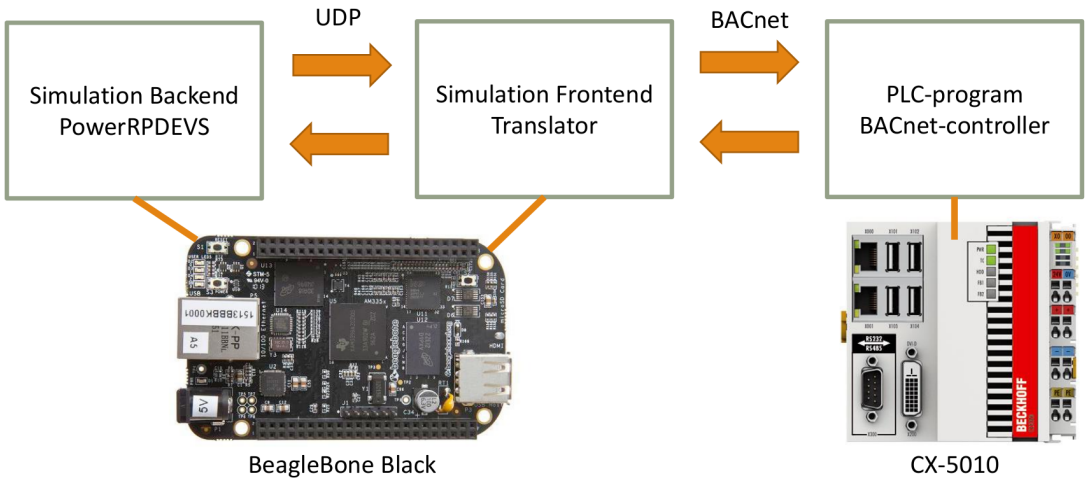


Figure 6.1: Component overview BACnet HIL simulation

An overview of the deployment on these two devices is given in Figure 6.2.

6.1 Model-Generation

While a JAVA .jar file is completely independent from the OS, meaning that it can be compiled on Windows and run on the BBB, the C++ model of the PowerRPDEVS simulation can only be run on the host it was compiled on. The resources on the BBB

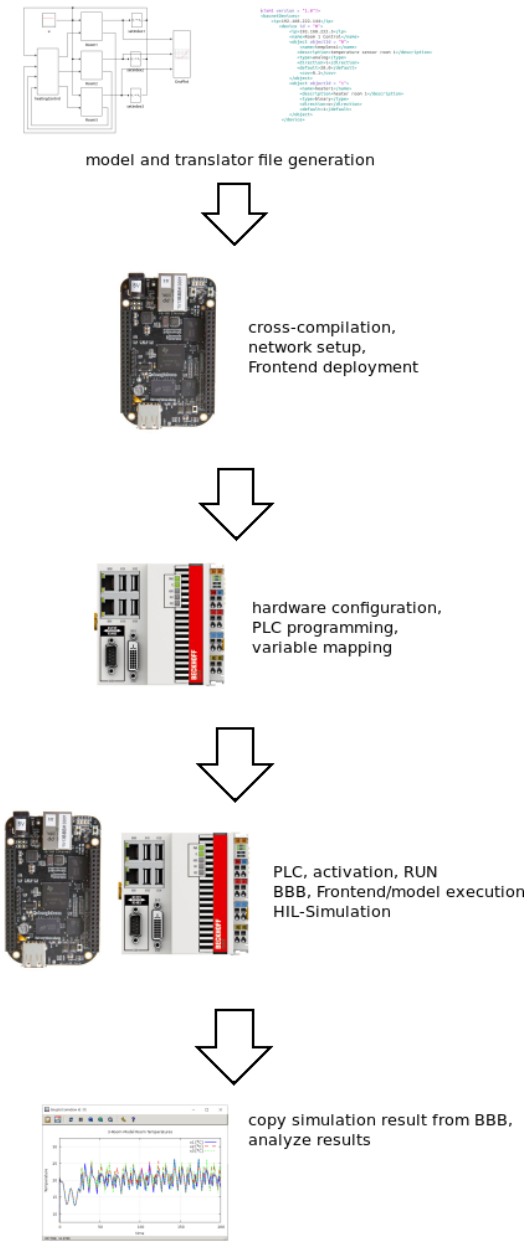


Figure 6.2: Overview of the HIL simulation workflow

are quiet limited and the simulation engine is not fully compatible with the underlying ARM architecture. Meaning that the model has to be cross-compiled on a PC and then copied to the BBB. Cross-compilation is the process of compiling an executable for a different platform and is often used when the other platform is limited in resources or less accessible.

6.1.1 PowerRPDEVS Build Process

The normal build process of PowerRPDEVS is accessible via the 'Simulate' menu. Contents of these menus are user configurable and the configuration file *powerdevs.ini* is located in the bin folder of the installation. Inside this file, the action of a menu click is defined. When clicking 'Simulate' or the play button on the top of the window, the software *pdppt* is executed. *Pdppt* handles two things, first it creates a dynamic Makefile for the current model that only compiles blocks that are used and second it executes this dynamic and a static Makefile. The static Makefile compiles parts that are always needed, like e.g. the `root_simulator`.
Afterwards, an executable called *model* is present in the output folder of the installation. The 'Simulate' button additionally starts a window that allows the user to directly execute the newly generated model.

6.1.2 Cross-Compilation

A new menu entry in the PowerRPDEVS model editor is generated to execute a cross-compilation. For this, the following line is added to the menu section of the powerdevs.ini file:

```
1 Simulation\CrossCompile=./pdppt, -arm %F, , ./images/beaglebone.png, 1, 1
```

This line adds a CrossCompile option to the drop down menu 'Simulation'. Additionally, an image is provided meaning that the BeagleBone logo is available as a clickable option at the top of the model editor window. Clicking either of this options will execute pdppt with the parameter '-arm'.

pdppt

Inside pdppt the parameter '-arm' will set an indicator variable for cross compilation. Under Linux there are cross-compilers available for many platforms with ARM like the Raspberry Pi or the BBB. The compiler for both C and C++ can be installed with:

```
1 sudo apt-get install libc6-armel-cross libc6-dev-armel-cross
    binutils-arm-linux-gnueabi libncurses5-dev
```

When executing a Makefile, variables can be set that are later used at execution time. To specify a custom compiler, a new variable called 'COMPILER' is created. Depending on whether cross-compilation is chosen or not, *arm-linux-gnueabihf-g++* is used or for normal compilation the default *g++* compiler. In both Makefiles, dynamic and static, the 'CXX' variable has to be replaced with 'COMPILER' in order to use the custom set variable. In the static Makefile this change can be entered directly. The dynamic Makefile is generated by the *codegenerator* class of *pdppt*, here the 'CXX' needs to replace the in the source code.

When the compilation finishes, a window will pop up, indicating that the model file located in the output directory is ready.

pdevslib

The pdevslib library is responsible for interaction with other applications. This includes output of simulation data to gnuplot, but also to scilab. This output form is remnant of the original PowerDEVS implementation. It requires system dependencies that are not available on the ARM architecture. There are already different pdevslib libraries for Windows(*pdevslib.win.cpp*), Linux(*pdevslib.linux.cpp*) and Rtai(*pdevslib.rtai.cpp*), so a new one was added called pdevslib.arm.cpp. In this file, all scilab references were removed making it compatible to the BBB.

To include pdevslib.arm.cpp instead of pdevslib.linux, the pdevslib.cpp file is modified by adding a new option to the include section shown in Listing 6.1.

```
#ifdef __Windows__
typedef short _stdcall (*inpfuncPtr)(short portaddr);
typedef void _stdcall (*oupfuncPtr)(short portaddr, short datum);
#include "pdevslib.win.cpp"
#else
#ifdef RTAIOS
#include "pdevslib.rtai.cpp"
#else
#ifdef __ARM__
#include "pdevslib.arm.cpp"
#else
#include "pdevslib.linux.cpp"
#endif
#endif
#endif
```

Listing 6.1: ARM selection in pdevslib

In pdevslib.common.cpp, the same set of define macros prevents the including of the incompatible hdf5.h library.

Makefile

The static Makefile is called *Makefile.include* and is located in the folder *engine* of the PowerRPDEVS base directory. For implementing the separation between cross and normal compilation, in Makefile.include, the keyword 'CXX', which specifies the default C++ compiler, needs to be replaced with 'COMPILER'. The dynamic include process shown in Listing 6.1 requires the variable ___ARM___. Therefore, it is defined in the Makefile if the OS is Linux but the compiler is not set to g++.
Libraries become hardware dependent when they are installed. Because of this, they cannot be used for cross compilation. Instead, the ARM compiled dependencies are copied from the BBB to a folder called *ARM* in the build folder of the PowerRPDEVS installation. When cross compilation is chosen, the linker in the Makefile is told to use the ARM compiled libraries instead of the system defaults.

Since a Makefile only recompiles and builds the parts of the code that have changed, a switch from normal compilation to cross compilation first needs to clean up all precompiled code. This can be done via the CleanUp option of the Simulate drop down menu.

6.2 BeagleBone Black

The BeagleBone Black is connected to the engineering PC via the USB cable. After the boot process is done a new external drive called 'BEAGLEBONE' will show up, indicating that the board is accessible via the USB-Ethernet. If no driver is available, it can be installed by executing a setup file located in the external drive under 'Drivers'. When the Ethernet communication is established the BBB can be accessed via SSH at IP 192.168.7.2 with username *debian* and password *temppwd*.

- Windows
  requires 3rd party software to communicate via SSH. One such software is called *Putty* and can be downloaded for free under [PUT]. To connect to the BBB the IP address is entered as hostname and SSH with port 22 is chosen.

  After this, the remote console will appear and ask for the username and password to login. At successful connection the user will be located in the *debian* users home directory.

- Ubuntu
  can communicate via the command line with the BBB by entering the following command:

```
1 ssh debian@192.168.7.2
```

  This command already specifies the username. After the password entry, the user is located in the debian user home directory.

To execute JAVA or any other part of the BACnet HIL Simulation the microSD card has to be mounted. This is done via the following command:

```
1 sudo mount /dev/mmcblk0p1 /media/BACnet/
```

The different parts of the simulation are located in the following directories:

- PowerRPDEVS installation
  /media/BACnet/powerrpdevs-code/
- model file
  /media/BACnet/powerrpdevs-code/output/model
- Frontend jar file
  /media/BACnet/powerrp_frontend.jar
- Translator file
  /media/BACnet/powerrpdevs-code/output/translator.xml
- GnuPlot outputs
  /media/BACnet/powerrpdevs-code/output/plots/

To establish this structure or to update either one of these files there are two different ways:

- Ubuntu
  In Ubuntu remote folders can be accessed via the ssh connection. In the Ubuntu file browser *Nautilus*, *Other Locations* is the option to connect to a server. To access the BBB the following server has to be entered: `sftp://192.168.7.2/`. After login the user is located in the root directory'/'.
- Windows
  requires 3rd party software to copy data via SSH. One such software is called WinSCP and can be downloaded for free from [SCP]. On start the application presents a login window, here the user has to enter the IP address as well as the username and password. After connection, the main window is split in two sides, the engineering PC on the left and the BBB on the right. Files can be transferred by drag and drop from one machine to the other.

6.2.1 Network

The Ethernet port of the BBB has to be connected to the laboratory network with the 128.130.56.X range. It is important that the BBB receives an IP address from the network and is reachable before configuring any virtual networks or iptables. The first step is to add three virtual networks for the 3 rooms. This is done via the following commands:

```
1 ifconfig eth0:bacnet0 128.130.56.110 netmask 255.255.255.128
2 ifconfig eth0:bacnet1 128.130.56.111 netmask 255.255.255.128
3 ifconfig eth0:bacnet2 128.130.56.112 netmask 255.255.255.128
```

The second step is to add the iptable rules for routing the packages.

```
1 iptables -t nat -A PREROUTING -p udp -d 128.130.56.110 –dport 47808 -j
    DNAT –to-destination 128.130.56.79:47824
2 iptables -t nat -A POSTROUTING -p udp –sport 47825 -j SNAT –to-source
    128.130.56.110:47808
3 iptables -t nat -A PREROUTING -p udp -d 128.130.56.111 –dport 47808 -j
    DNAT –to-destination 128.130.56.79:47826
4 iptables -t nat -A POSTROUTING -p udp –sport 47827 -j SNAT –to-source
    128.130.56.111:47808
5 iptables -t nat -A PREROUTING -p udp -d 128.130.56.112 –dport 47808 -j
    DNAT –to-destination 128.130.56.79:47828
6 iptables -t nat -A POSTROUTING -p udp –sport 47829 -j SNAT –to-source
    128.130.56.112:47808
```

The BBB is now ready for the BACnet HIL Simulation. To check if the virtual networks have been configured, the addresses can be pinged from other devices. The virtual networks and the iptables reset with every reboot. It is recommended to not automatically start the script since it is not given that the network connection is already up and running when the script is executed, which would result in a wrong routing.

6.3 PLC (CX5010)

6.3.1 Hardware Configuration

The CX5010 is booted by providing power and connecting the Ethernet cable to the bottom RJ45 port on the front of the machine. The hardware configuration is done first by starting the TwinCAT System Manager. The window depicted in Figure 6.3 will open up.

The System Manager is by default connected to the local runtime (even if it is not installed) and is in CONFIG mode, which can be seen in the bottom right corner of the window.
To connect to the PLC the button 'Zielsystem wählen' is pressed. The user is redirected to the window shown in Figure 6.4. In this window the target PLC is already listed as CX-0E264E. The name consists of CX for Beckhoff devices and the last 3 bytes of the

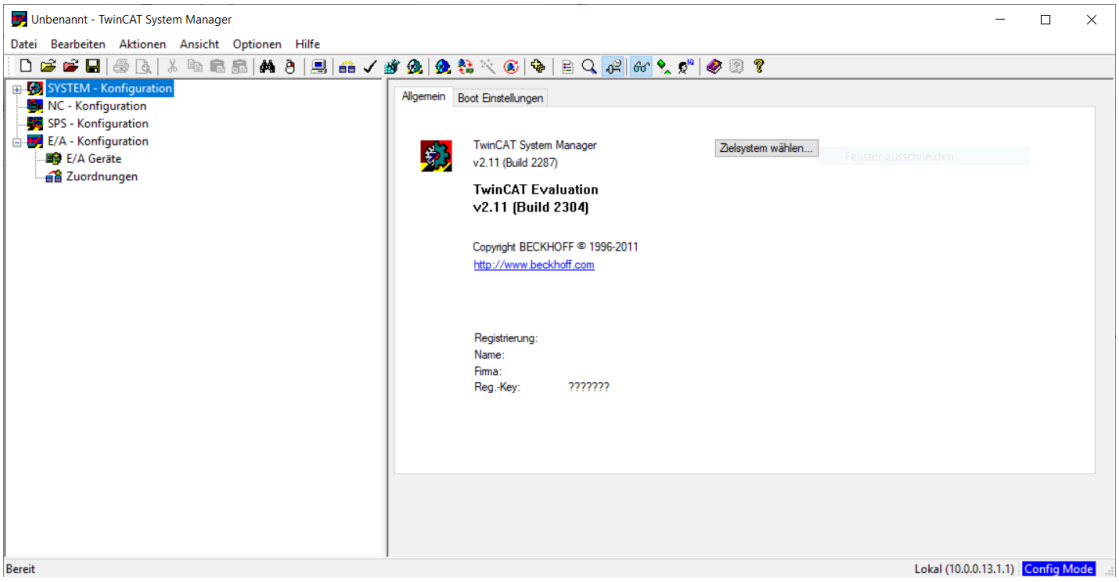


Figure 6.3: Empty TC2 SystemManager project

MAC address. If the target is already listed, it can be selected by clicking on it and hitting OK. If the target is not shown, the network can be searched via the 'Suchen (Ethernet)' button. The search window is shown in Figure 6.5.

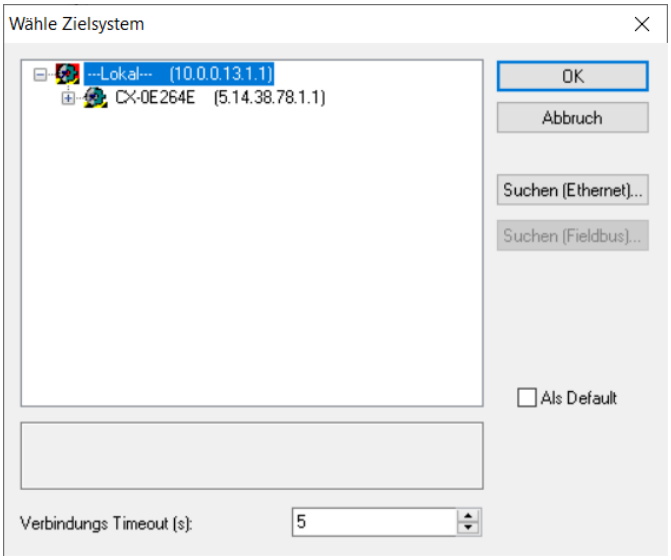


Figure 6.4: Target selection window

The list in the search window will be empty at first. To display all devices on the network, 'Broadcast Search' has to be clicked. The PLC will show up if it is powered up and connected to the network. It is important that the TwinCAT version is 2.11.2259, for

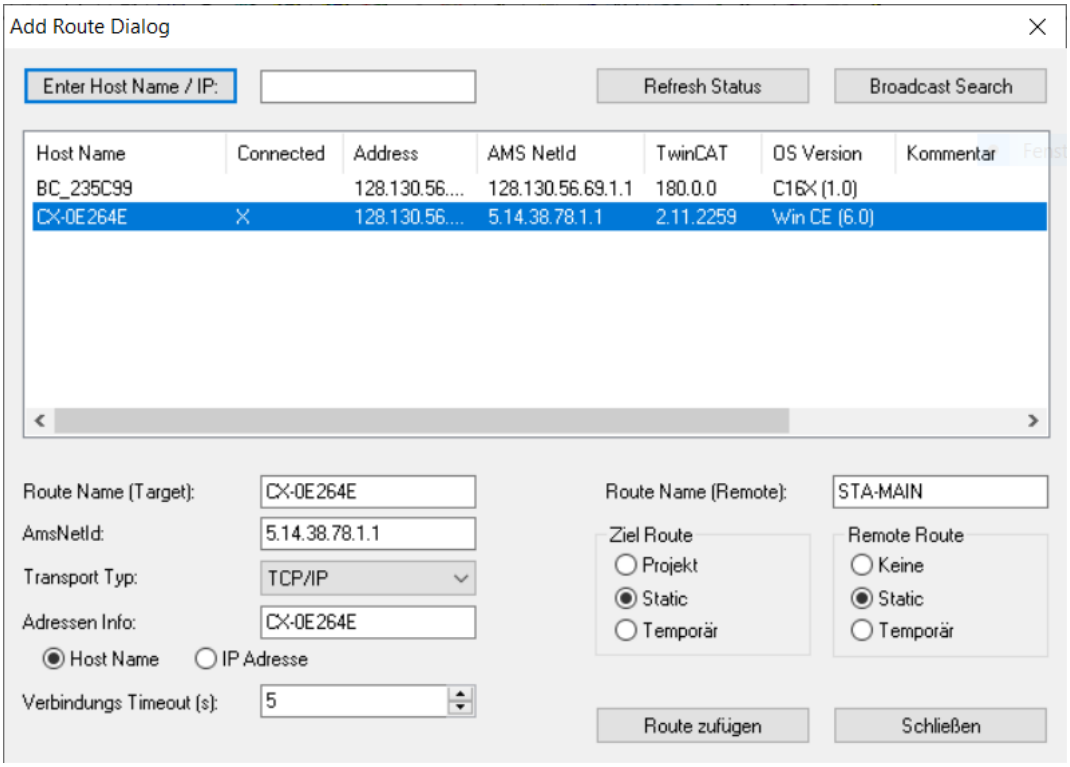


Figure 6.5: Ethernet search window

the implementation in this thesis to work. The controller is selected by marking it and clicking 'Route zufügen'. A login window will pop up were the username is 'Administrator' and the password is a single blankspace ' '. After this, the window can be closed and the target is selectable in the target window.

The next step is to add the BACnet adapter to the hardware configuration. On the left side of the main window the menu item 'E/A-Konfiguration' will reveal the option 'E/A-Geräte'. A rightclick and 'Gerät hinzufügen' will open a form to choose which adapter shall be added. For this thesis a BACnet revision 12 adapter is added as shown in Figure 6.6. The name has to be entered at the bottom of the window. It cannot be changed later.

This will add an entry to the 'E/A-Geräte' tab. Clicking it will reveal the properties. Before showing the properties for the first time a network adapter has to be chosen in a pop up window, as depicted in figure 6.7.

The BACnet license is connected to the MAC address. For the CX5010 in our laboratory, a license was purchased only for the adapter marked in Figure 6.7. Selecting the second adapter will lead to a crash of the PLC at runtime.
The settings tab of the device properties will reveal options to configure the adapter as well as information about the IP address of the PLC. At this point the E/A devices

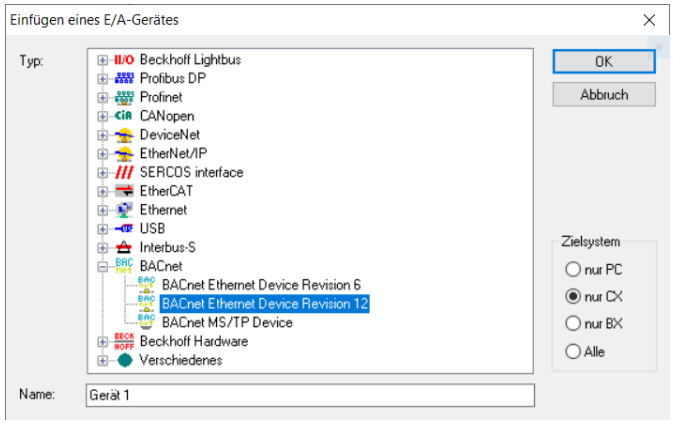


Figure 6.6: BACnet device addition

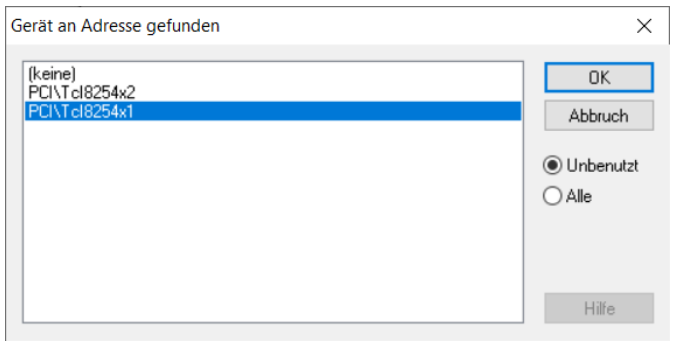


Figure 6.7: Network adapter selection for BACnet

have to be reloaded via a reload button at the top of the main window (Figure 6.3). The license key section at the middle right of window 6.8 has to show 'valid BACnet supplement key'. If this is not the case, the key can be entered via the 'Insert Key' button.

The MAC address and license key pair for the laboratory CX5010 is:

```
1 00:01:05:0E:26:4E
2 162C-4A02-1D3F-3F2D
```

The BACnet adapter allows to either run one BACnet server that host objects, or to function as a BACnet client that communicates with other devices. While only one server is possible per BACnet adapter, there is no limit for clients, since every client corresponds to a remote device.

Clients or devices can be added to the configuration in two different ways, manually by right clicking the adapter and adding a BACnet client the same way as with the adapter

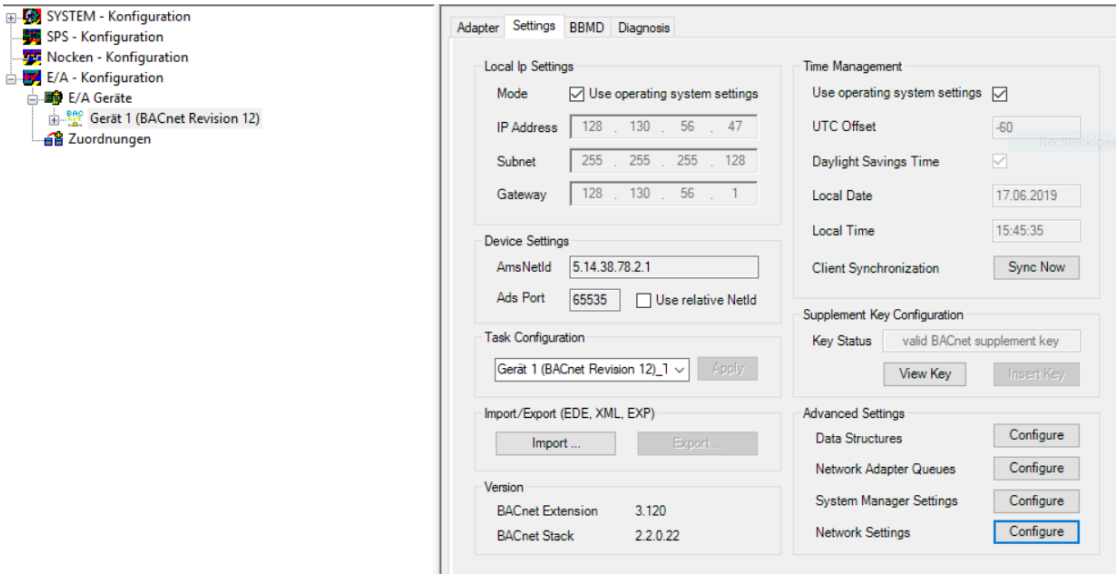


Figure 6.8: Settings of the BACnet device

itself and continuing by adding objects to the client, or automatically by scanning the network.
The possibility to scan the network is presented when right clicking on the BACnet adapter.

To actually find devices, they have to be online which means that the powerrp_frontend.jar file is running on the BBB with the correct translator file. To start the Simulation Frontend, the following command is executed via SSH on the BBB:

```
1 java -jar powerrp_frontend.jar -i2019 -o2017 -noexec
```

This line will start the Frontend that in turn will create the BACnet devices without executing the simulation model. When the prompt to start or end the simulation is presented the devices are ready to be discovered.

When clicking on 'Scan Devices' in the right click menu of the BACnet adapter the dialog in Figure 6.9 will open presenting the user with the discovered devices.

The devices that should be added to the hardware configuration have to be checked with the checkbox and added by clicking the 'Add Clients' button. The object names and descriptions can be wrong initially as seen in the picture, but when clicking the add button, the client information will be read again and the correct values are displayed.

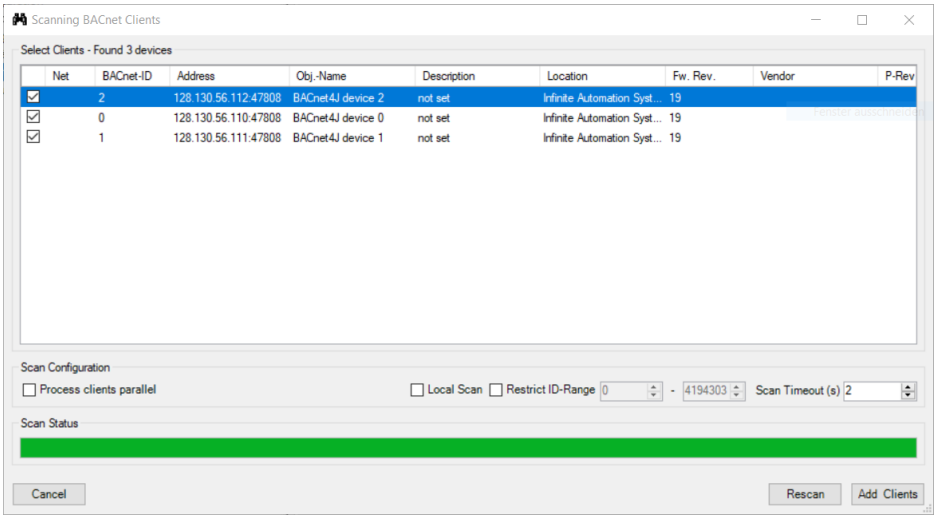


Figure 6.9: BACnet devices that were discovered in the network

Adding the devices manually or automatically should result in an E/A device configuration like the one depicted in Figure 6.10.

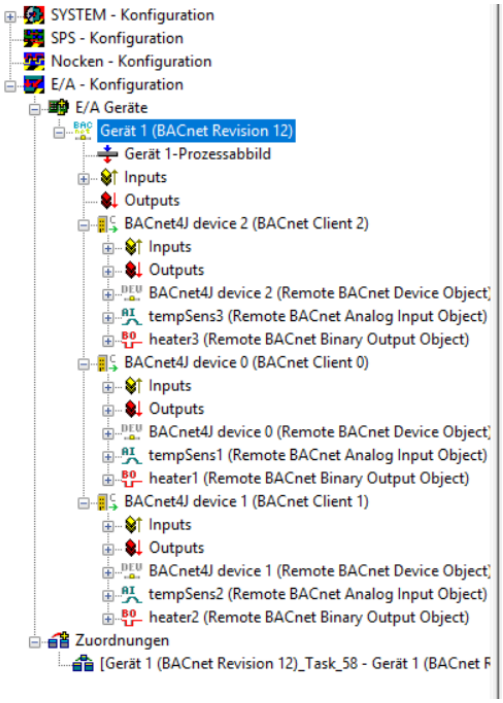


Figure 6.10: BACnet devices that were discovered in the network

The ordering of the devices might be out of order due to the discovery process. To work

with the devices the E/A objects have to be reloaded by clicking the reload symbol in the top menu. Additionally, the configuration can be activated which will switch the PLC into RUN mode, reading the values of the BACnet properties. Clicking a device will reveal the properties. An example for Room 1 is shown in Figure 6.11.

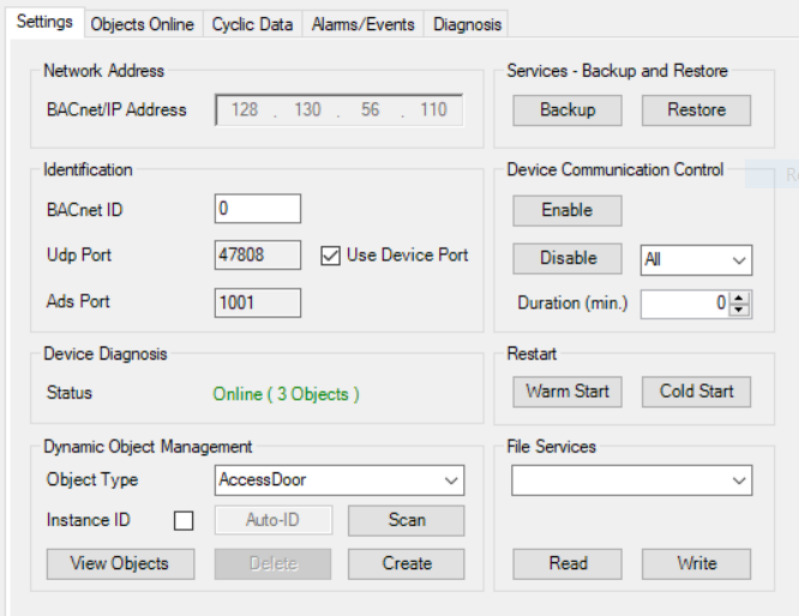


Figure 6.11: BACnet devices that were discovered in the network

The device diagnosis shows three online objects (the device itself is counted as an object). In the 'Objects Online' tab the properties of the devices can be read.

For further configuration, the PLC has to be reset to CONFIG mode. Expanding the objects like tempSens1 will reveal inputs and outputs without any properties shown. To later link the properties to PLC variables the software needs to be told which properties should be subscribed to. TempSensX is always of type AnalogInput, so only inputs are needed. Especially two properties are required, PresentValue and StatusFlags. To add them, there is a tab called 'Cyclic In' in the properties window of the object which lists all available BACnet input properties. To activate the two properties, the checkboxes have to be checked like shown in Figure 6.12. The same thing needs to be done with the StatusFlags.

HeaterX requires both input and output properties. Input properties are needed to read the initial value of the PresentValue and the StatusFlags and output properties to be able to write a new value to the PresentValue. Inputs are added the same way as with the TempSensX. Outputs are added via the 'Cyclic Out' tab of the object properties. The activation of the PresentValue output is presented in Figure 6.13.

The hardware configuration is finished at this point. To simplify the PLC programming the BACnet devices can be exported in the 'Settings' tab of the BACnet adapter properties.

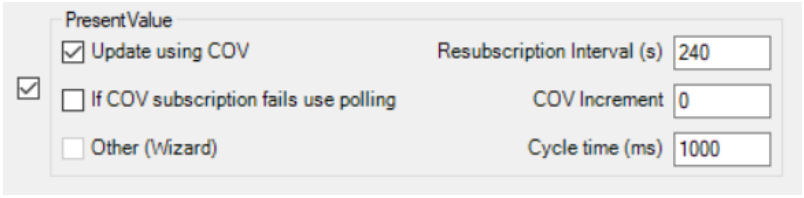


Figure 6.12: Activation of the PresentValue subscription

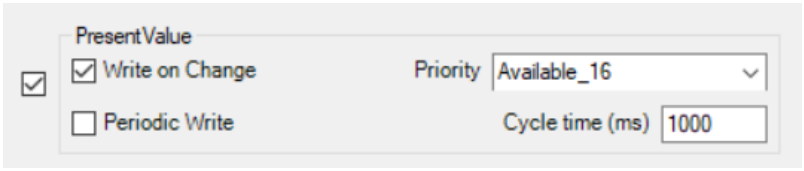


Figure 6.13: Activation of the PresentValue write

6.3.2 PLC program

The implementation of the PLC program is done in the second part of TwinCAT, in PLC-Control. Starting the application for the first time and creating an empty project will reveal an empty MAIN POU (Programming Organization Unit). It is the starting point of the PLC program execution. To keep the program modular every room is handled in a separate POU. To add a POU to the POUs folder in the project explorer on the left, it has to be right clicked and the add option has to be selected. The window depicted in Figure 6.14 will appear.

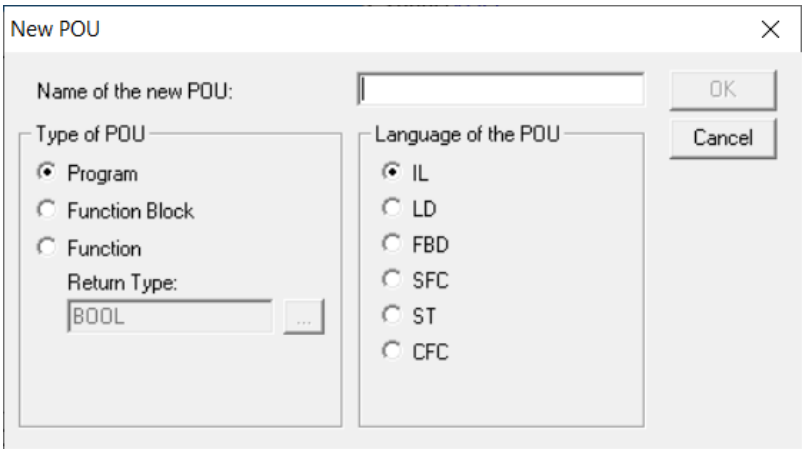


Figure 6.14: Adding another POU to the project

For the type of the POU, Program is selected. Program is equal to an OB (Organisationsbaustein) in Siemens PLC programming. This means that the execution will keep values between multiple cycles of the PLC execution.
FBD (Function Block Diagram) was chosen as the programming language.

The next step is to add the previously exported BACnet configuration to the project. In Project/Import the location of the file can be chosen. After clicking OK, the project explorer looks like Figure 6.15.

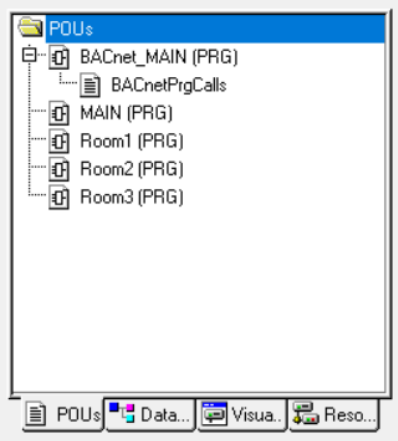


Figure 6.15: POUs of the BACnet HIL simulation

On the bottom of the explorer are 4 different tabs. The next step requires switching into the Resources tab on the right. To be able to use the imported BACnet structure in the PLC program, the corresponding library has to be added via the Library Manager. By right clicking into the top left field of the manager, a new library can be added. The necessary BACnet libraries are already contained in the Lib folder of the TwinCAT installation. The libraries and the location of the Library Manager are shown in Figure 6.16.

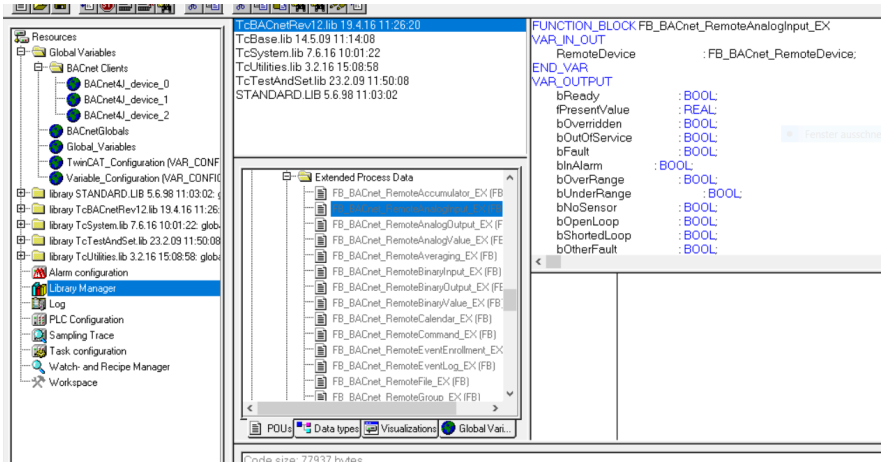


Figure 6.16: Library manager in PLC-Control

On the top left of the Resources window in Figure 6.16, the three BACnet devices can be seen as global variables. Clicking on one of the devices reveals the name and type of

the variable that can be used to read and write data to and from those devices. Figure 6.17 shows the result for device 0.

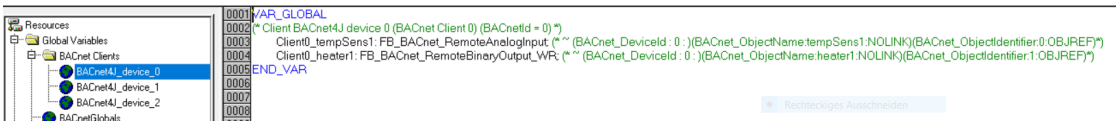


Figure 6.17: Objects of Room 1 represented as global variables

To check the implementation and finish the PLC programming the project needs to be compiled via the top menu of the PLC-Control. After successful compilation the project is ready to be mapped.

6.3.3 Mapping

After both, hardware configuration and PLC program are finished, the two worlds need to be connected. Practically all that needs to be done is to connect the "physical" ports of the E/A devices with the software ports so that values can be exchanged. To do this, the System Manager shows the point 'SPS-Konfiguration' in the explorer on the left. With a right click and Add, the previously compiled PLC-Control object can be imported. The in and outputs of the PLC program will be shown after expanding the 'Standard' option. To connect for example `fPresentValue` of room 1 with the BACnet object, double clicking the variable opens a window to create the mapping (see Figure 6.18). TwinCAT will automatically show only compatible and unused hardware inputs. This means that if there are multiple options the user has to chose the right hardware input from the list.

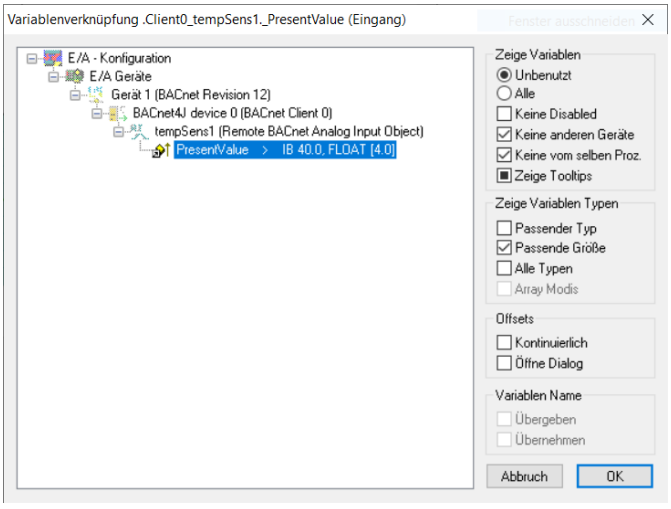


Figure 6.18: Mapping of PLC program input with hardware

It is important that all variables that are shown in the 'SPS-Konfiguration' are linked to physical devices, otherwise the PLC blocks will return a *wrong mapping* error.

Mapped variables or hardware ports will be indicated by a linked symbol, as displayed in Figure 6.19.

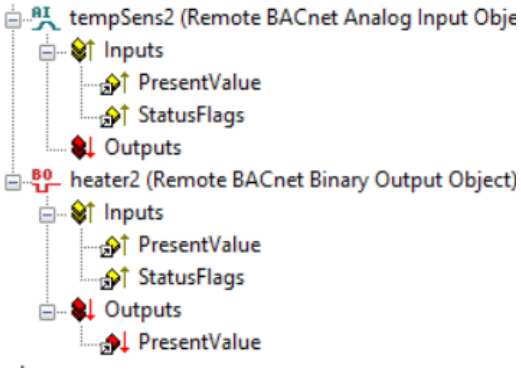


Figure 6.19: Mapped hardware ports of Room 2

6.4 Simulation

To conduct a simulation the Simulation Frontend has to be started. If GnuPlot is used, the display window can be redirected to the engineering PC by forwarding the X screen via SSH. In Ubuntu this can be achieved by adding the -x parameter to the SSH connection command.

```
1 ssh -x debian@192.168.7.2
```

In Windows, a software to display the X server has to be installed. In this thesis *Xming* is used, the application can be downloaded from [XMI]. After launching Xming, Putty can be configured. In the left menu, under Connection/Serial exists the point X11. There X11-Forwarding has to be enabled and for the x display location, 'localhost:0' has to be entered.

The Frontend on the BBB is started via the following command:

```
1 java -jar powerrp_frontend.jar -i2019 -o2017 -s
```

Then, the GnuPlot window opens on the engineering PC, but the simulation does not start automatically if the corresponding option in the PowerRPDEVS UDP-Source Block is set to wait for a start signal..

In the System Manger the configuration is activated via the button in the top menu and the PLC is switched to RUN mode. When reconfiguring the hardware, the PLC program is purged. In the PLC-Control, under the option Login in the top menu, the user can

logon to the PLC. This process will flash changed or completely new PLC programs if the pop up window is accepted.
To run the PLC program the option Run in the Login drop down menu has to be activated. The PLC-Control will switch into the online mode that shows the current values of the components in the network. The temperature values correspond to the default values specified in the translator file.

The simulation is started by entering 's' for start in the input prompt of the Frontend. The online data in the PLC control will start to change. An example can be seen in Figure 6.20.

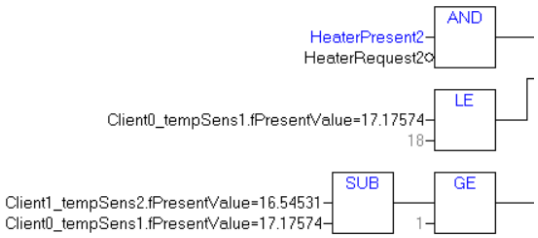


Figure 6.20: Online view of the PLC-Control

Boolean TRUE is displayed blue, FALSE black and for REALs the current value is written.

The GnuPlot window will start to draw trajectories according to the simulation data and the control of the PLC. If an end time was configured in PowerRPDEVS the window closes after the time is over. The simulation continues but no new data will be added to the plot. To end the simulation, 'e' for exit has to be entered into the input prompt of the Frontend.

CHAPTER 7

Evaluation

7.1 Use Case

For the simulation model a benchmark for hybrid simulation engines is used. The model can be found in the paper [FI04] and is called the *room heating benchmark*. The premise is that there are three rooms that interact with each other and with the outside via heat transfer. The task for the temperature controller is to keep the three room temperatures above given thresholds by scheduling two available heaters for the three rooms. It is a good model to evaluate the performance of hybrid simulation engines since it directly combines continuous calculations of room temperatures with the discrete heater selection and control algorithm.
The temperature controller was implemented both, as simulation model and on the Beckhoff PLC. This allowed us to compare the results from the HIL simulation with the results of the pure simulation.

The heater control algorithm that was exported from the *room heating benchmark* consists of a deterministic and a non deterministic behavior.

The deterministic constraints for switching a heater from room j to room i are as follows:

- room i and j need to be neighbors
- room i has no heater
- room j has a heater
- temperature in room i is less or equal to a constant get_i
- the difference of the temperature in room j and room i is greater or equal a constant dif_i

Additionally, the heater in a room i is on if the room temperature is below a constant on_i and off if the temperature is above a constant off_i.

The non deterministic behavior occurs when two rooms concurrently require a heater. Then, the procedure is to randomly choose one room.
The constants for switching the heaters were chosen equal for every room. Constant get_x is 18°C, $diff_x$ equals 1°C, on_x equals 20°C and off_x is 21°C.

7.1.1 PowerRPDEVS Simulation Model

The simulation model for PowerRPDEVS is already included in the *examples/BuildingSimulation* folder of the PowerRPDEVS installation. The model can be seen in Figure 7.1. In this implementation if there are multiple rooms requesting a heater, a random choice is made. For the outside temperature 4°C are chosen. The deterministic behavior is implemented according to subsection 7.1.1.The rooms are modeled as coupled blocks that calculate the room temperature according to a heat transfer matrix that can be found in [FI04].

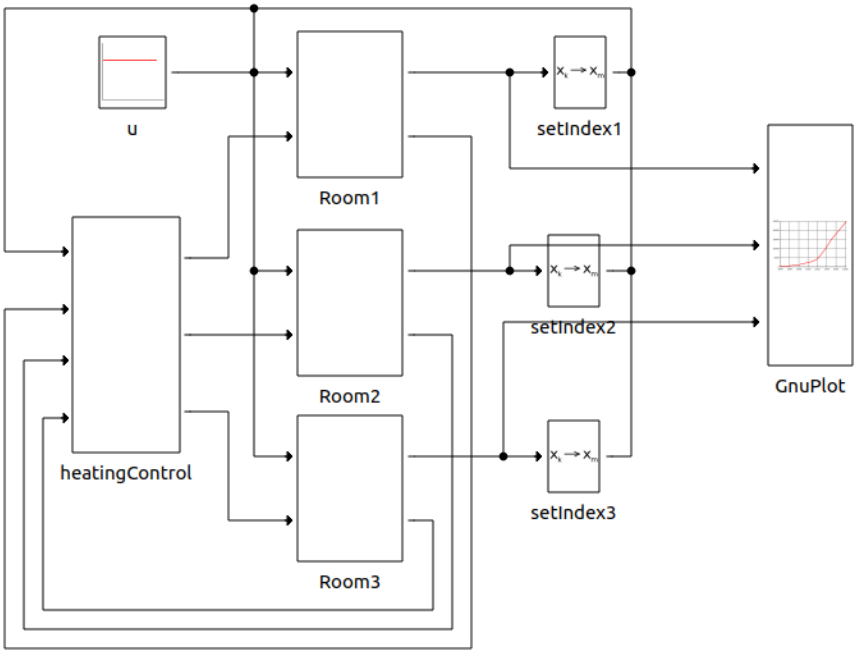


Figure 7.1: 3 Room Benchmark in PowerRPDEVS

7.1.2 HIL Simulation Model

PowerRPDEVS

For the HIL simulation, all control structures are removed from the 3 room model and instead the outputs and inputs of the rooms were connected to a UDP-Sink and UDP-Source. So the benchmark in the domain of PowerRPDEVS was reduced to a room temperature simulation that is presented in Figure 7.2.

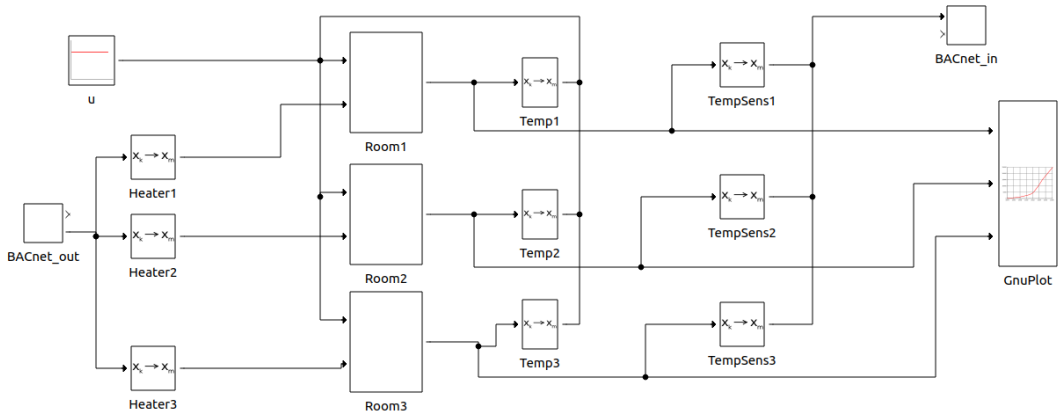


Figure 7.2: 3 Room temperature model for HIL

Since there are three rooms, the BACnet structure will represent 3 temperature sensors and 3 binary heater control elements. In the model, the output of the UDP-Source corresponds to the heater controls and thus, need to be split up by SelectIndex blocks that transform the BACnet ID of the Frontend to index 0 since the rooms expect a single value at index 0 as on/off signal.
In the same way the temperature outputs of the rooms are transformed to BACnet IDs that are feed to the UDP-Sink.
Visualization of the room temperatures is carried out by a GnuPlot block. Since the Simulation is set to be open end, the end time of the GnuPlot Window has to be set manually by clicking on the block and replacing %tf (final time) with the desired value (60s in this case).

7.1.3 Translator File

To configure the BACnet devices that are represented in the simulation model, a translator file (xml) is created. It tells the Frontend, which BACnet devices are simulated and which properties they have. BACnet structures often encapsulated certain objects or sensors into devices when they belong together. In this case the chosen structure is shown in Figure 7.3. Dotted lines represent the BACnet devices. In this case, every device contains a *Temperature Sensor* and a *Heater* object. Temperature sensors are

represented as AnalogInput objects, while control elements like the heaters are represented as BinaryOutput objects.

As exemplarily depicted for Room 1 in Listing 7.1, the initial room temperature is set to 20°C. Further, the heaters in Room 1 and 2 are activated. COV reporting is enabled by setting the COV of the Analog objects to 0.1 meaning that a 0.1 degree difference in temperature triggers a subscription service.
The IP address of the controller, as well as the broadcast address of the laboratory Ethernet can be seen in Figure 7.3. The IP addresses of the devices is later defined by the ordering in the translator file in combination with the network setup.

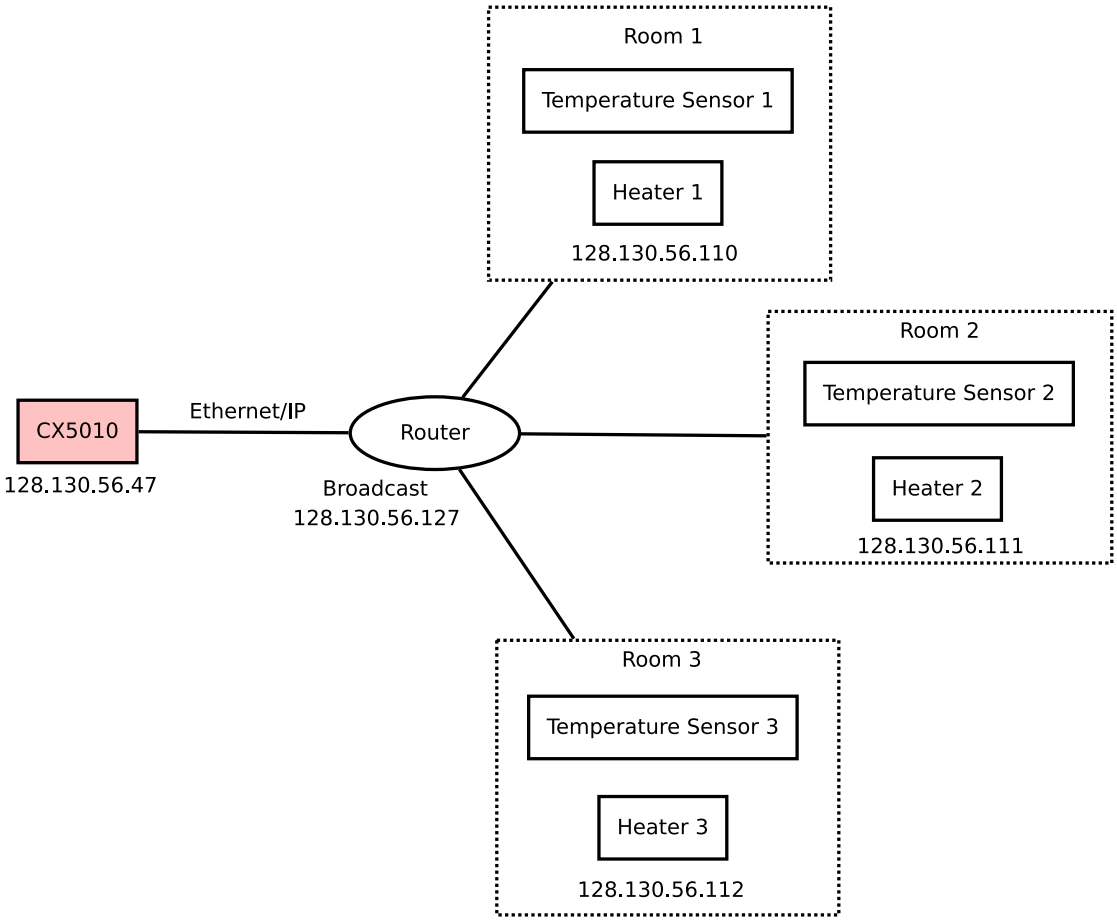


Figure 7.3: 3 Room BACnet structure

Listing 7.1 shows the definition of device 1. The definition of the other devices is analog (9.1.2).

```
<?xml version = "1.0"?>
<bacnetDevices>
<ip>128.130.56.47</ip>
```

```
<broadcast>128.130.56.127</broadcast>
<device id = "0">
<name>Room 1 Control</name>
<object objectId = "0">
<name>tempSens1</name>
<description>temperature sensor room 1</description>
<type>analog</type>
<direction>i</direction>
<default>20.0</default>
<cov>0.1</cov>
</object>
<object objectId = "1">
<name>heater1</name>
<description>heater room 1</description>
<type>binary</type>
<direction>o</direction>
<default>false</default>
<cov>1</cov>
</object>
</device>
```

Listing 7.1: IP setup and Room 1 translator file

PLC

For the PLC controller the heater control algorithm from [FI04] is implemented in FBD. For this the non deterministic behavior of choosing a random room is changed to a deterministic priority ordering of the rooms. If two rooms require a heater, the room with the lowest number is granted the heater. Figure 7.4 presents the implementation of the Room 1 heater request algorithm as an example. The entire code is included in the appendix 9.3.

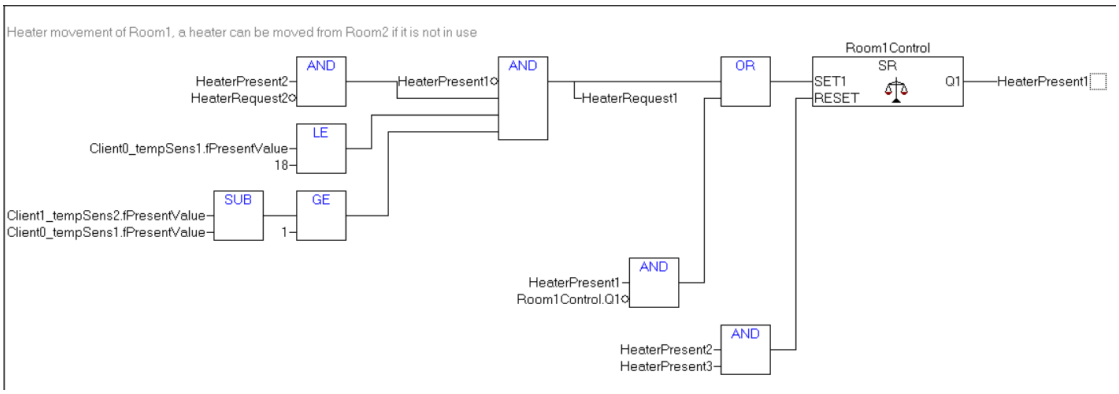


Figure 7.4: Room 1 heater control implementation

The priority ordering of the rooms is caused by the use of a SR (set dominant) latch.

The rooms are executed sequentially by the PLC controller. So if Room 1 requires a heater, its latch sets although the Room 2 currently holds the heater and the reset input of the latch is active. The next step is the execution of Room 2 which will loose the heater as the set input is pulled inactive by the activation of Room 1.
Generally, if the BACnet configuration was imported, it is not necessary to include a block for a BACnet input variable like, for example, AnalogInput. The internal variables of these blocks are globally accessible if the function block variable itself is global. This can be seen in Figure 7.4 with the use of the Client2_tempSens3.fPresentValue. The variables that are included in a block can be seen when opening the block in the Library Manager.
Output blocks need to be included in the network since they need control values to allow the write process to be carried out, as seen in Figure 7.5. It is important to choose the right prefix for the block according to the global variables, WR in this case for the Client2_heater3. No prefix would mean that the block is read only, EX would mean the block is able to read/write.

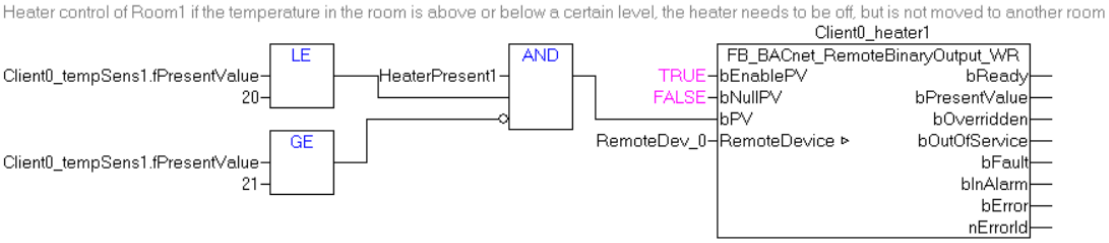


Figure 7.5: Room 1 heater on/off control

7.2 Results

To compare the results of both, the simulation only model and the HIL simulation, both simulations were executed for 60s.

7.2.1 Pure Simulation Model

The result of the simulation can be seen in Figure 7.6.

At simulation start, heaters 1 and 2 are active, this can be seen by the trajectories of the two rooms. While temperature x3 (Room 3) drops quickly, temperature x2 rises and x1 drops slower than x3. The average room temperature starts at 20°C and quickly drops to an average temperature of approximately 16.5°C. When this temperature is reached, the simulation continous with a sawtooth pattern. A room heats up when a heater is present and cools down when the heater moves to another room or is deactivated by the off_i level.

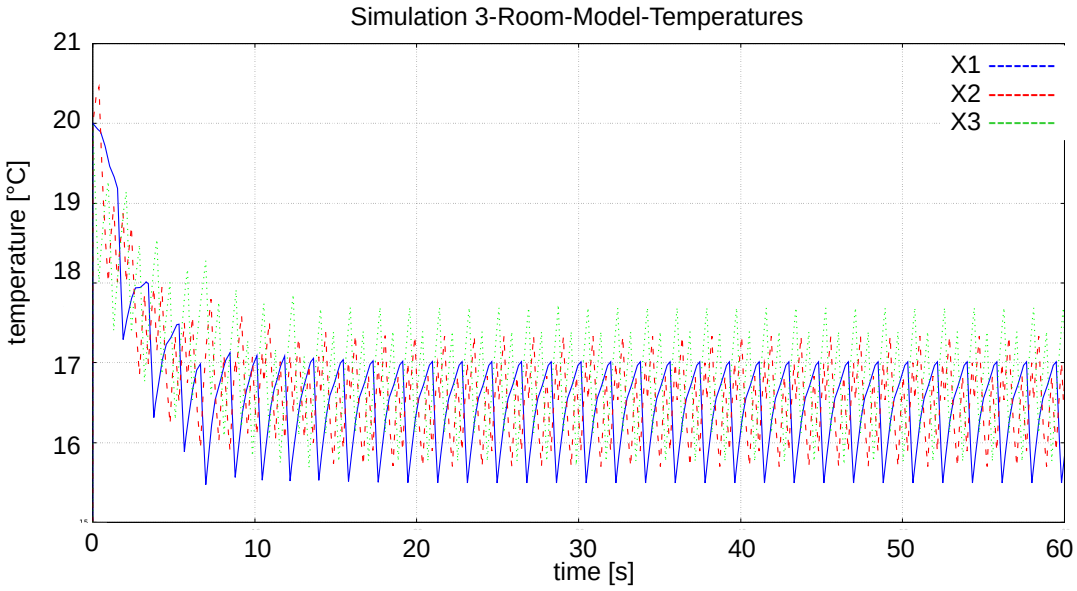


Figure 7.6: 3 Room Benchmark executed as pure simulation

Due to the parameter selection defined for the benchmark in [FI04], the dynamic of the system is very fast. While in a real room the temperature changes very slowly the emphasis of the benchmark is to produce comparable and easily observable results.

The drop at the start is not deterministic and changes with every execution. Additionally, in Figure 7.6 the rooms have different dynamics and average temperatures. Room 1 (blue) changes its temperature the slowest and has the lowest temperature. Room 2

(red) features a dynamic that is generally faster than Room 1 but is often interrupted since two rooms are able to receive a heater from it. Room 3 (green) has the fastest dynamic and the highest average temperature. This behavior can be explained by the room arrangement shown in Figure 7.7 that is encoded in the heat transfer function of the benchmark.

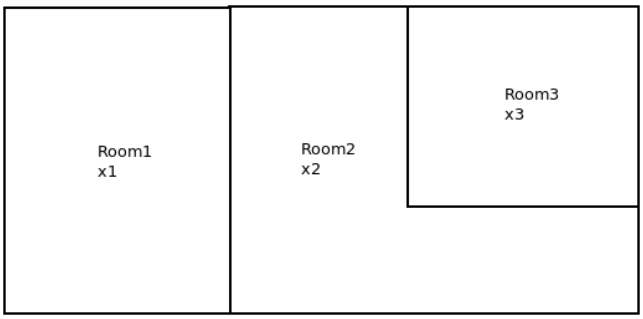


Figure 7.7: Room arrangement in the Room Heating Benchmark

The slow dynamic and low temperature of Room 1 is caused by three outer walls and only a single neighboring wall. Room 2 features a slightly smaller outer wall area and is heated by either two neighbors or an own heater, resulting in faster heating and a slightly slower cooling than Room 1. Room 3 heats and cools at the same speed as it has the same area of outside and inside walls. Due to the small size, it reaches a higher temperature.

7.2.2 HIL Simulation

For the Hardware in the Loop Simulation presented in this thesis the result of the plot looks like 7.8.

The HIL-Simulation features the same drop from 20°C degrees and after that an average temperature of 16.5°C. The same dynamics as in the pure simulation model can be observed.

7.2.3 Comparison

However, certain differences appear. The most notable is the lower speed of the HIL simulation. This is caused by two factors.

- COV raster
  For the COV value 0.1 degrees was chosen which means that only a change of 0.1 degrees Celsius in the room temperature triggers a subscription event. This value was chosen since it more accurately corresponds to real world values in building automation. The simulation triggers as soon as the thresholds are reached. The result is therefore a smaller positive and negative offset from the average value in the pure simulation.

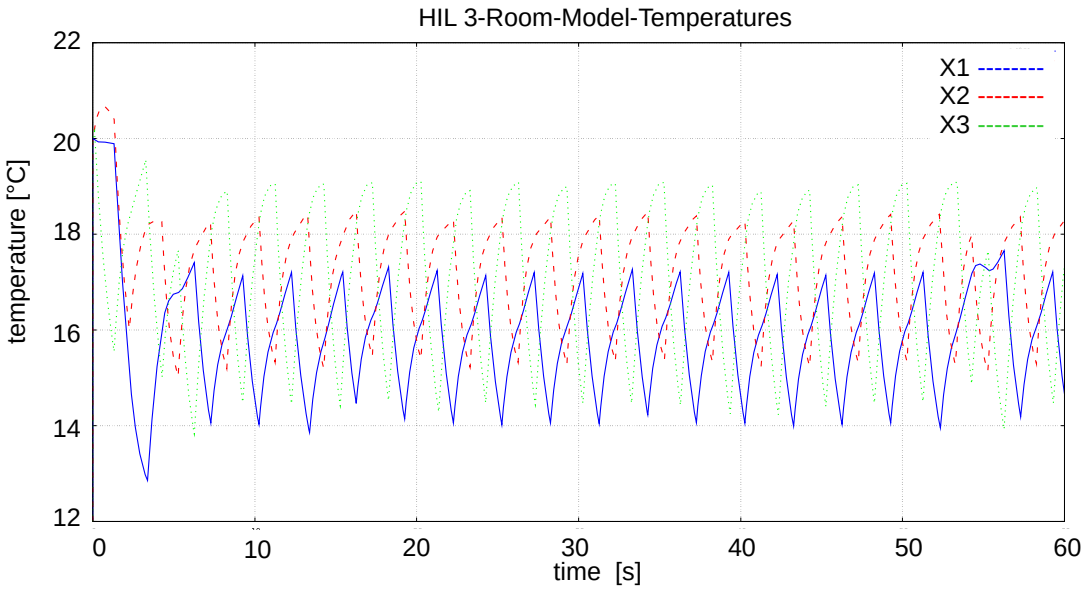


Figure 7.8: 3 Room Benchmark executed as HIL-Simulation

- communication delay
  Much like the real world there are different communication delays combined in the HIL-Simulation. Backend to Frontend, BACnet subscription event to actual read event and the execution cycle of the PLC as well result in a much higher delay for control values than the direct execution of the simulation model.

Another difference is inconsistent spikes which are caused by the priority orderings of the rooms when it comes to heater transactions. Furthermore, varying delays due to the execution cycle of the PLC in combination with the network speed. Additionally, BACnet communication is an UDP socket communication which means that there are no safeguards or certainties that the packets for the subscription are received by the PLC. Bad network connections will lead to more packets being dropped, like in real life BAS systems.

CHAPTER 8

Conclusion and Future Work

The results of the simulation show that the implementation is capable of hosting multiple BACnet devices on a single BBB. The devices and their properties can be controlled either via input from the PowerRPDEVS simulation, by the Simulation Backend, or by inputs from the PLC-controller that functions as the hardware under test.
The combination between the BBB and the Beckhoff CX5010 presents a practical way of testing BAS, without the need to connect the physical BACnet devices. The implementation of a control algorithm can therefore be tested cheaper and faster. From the view of the PLC controller, there is no difference in the network structure of the BACnet environment. Additionally, the setup could be an educational tool in the laboratory environment. Instead of providing the same fixed hardware configuration for every team or project, the simulation model can easily be changed and would not hinder the learning process, since the same program would be operational with real life hardware. The risk of damaging hardware is minimized, if the program is shown to work in the simulation, it can be deployed on the real hardware.

Future work could include an extension of the whole BACnet Stack, meaning that not only Analog and Binary in and outputs are possible but also MultiState or other BACnet object types. The translator file generation could be automated by introducing software that either extracts data from the PowerRPDEVS simulation or that uses the TwinCAT BACnet configuration file as input much like the PLC-Control.
Error case behavior of the system could be tested by introducing new blocks into the PowerPRDEVS library. These blocks, for example, could switch the state of a BACnet device to not available or OutOfService.
While this thesis focuses on BACnet, other protocols could be implemented instead, like, for example, KNX. Thereby, this thesis could be used as template.

CHAPTER 9

Appendix

9.1 Simulation Frontend

9.1.1 powerrp_frontend.java

```
import com.serotonin.bacnet4j.LocalDevice;
import com.serotonin.bacnet4j.npdu.ip.IpNetworkBuilder;
import com.serotonin.bacnet4j.obj.BACnetObject;
import com.serotonin.bacnet4j.transport.DefaultTransport;
import com.serotonin.bacnet4j.type.enumerated.*;
import com.serotonin.bacnet4j.type.primitive .*;
import messages.HBASimProtos;
import org. slf4j .Logger;
import org. slf4j .LoggerFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import java.lang.Double;
import java.net.InetAddress;
import java.io .*;
import java.net .*;
import java. util .Properties;
import java. util .concurrent.ConcurrentHashMap;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
```

```
/**
 * Main class of the BACnet Hardware-In-The-LOOP implementation
 *
 * This class implements the central function of a BACnet HIL Simulation
 * interface, it coordinates Threads to send and read data from the
 * powerrpDEVS simulation enginge as well as setup of the BACnet Devices
 * that are emulated.
 *
 * @author Stefan Adelmann <e01633044@student.tuwien.ac.at>
 */
public class powerrp_frontend{

    private static DatagramSocket sourceSocket = null;
    private static DatagramSocket providerSocket = null;
    private static InetAddress address = null;
    private static int sourcePort = 0;
    private static boolean sinkSet = false;
    private static boolean startStopSet = false;
    private static DatagramSocket sinkSocket = null;
    private static ConcurrentHashMap<Integer, BacnetDevice>
        ↪ localDeviceHashMap = new ConcurrentHashMap<Integer, BacnetDevice
        ↪ >();
    private static Process model = null;
    private static int internalPort = 0xBAD0;
    private static final Logger LOG = LoggerFactory.getLogger(powerrp_frontend.
        ↪ class);
    private static int sinkPort = 0;
    private static boolean sourceSet = false;
    private static boolean mirrorSet = false;
    private static boolean executeSet = true;
    private static String modelLocation = null;
    private static String translatorDir = null;
    private static SourceCommunication sourceCommunication = null;

    public static void main(String[] args) {
        InetAddress broadcastAddr = null;
        int bacnetPort = 0xBAC0;
        InetAddress controllerAddr = null;

        System.out.println("BACnetHIL-Simulation starting....");

        try (InputStream input = powerrp_frontend.class.getClassLoader().
            ↪ getResourceAsStream("frontendDefaults.properties")) {
```

```
        Properties prop = new Properties();

        prop.load(input);

        translatorDir  = prop.getProperty("bac.translatorDir");
        modelLocation = prop.getProperty("bac.modelDir");
        broadcastAddr = InetAddress.getByName(prop.getProperty("bac.
            ↪ broadcast"));
        bacnetPort = Integer.parseInt(prop.getProperty("bac.port"),16);
        internalPort  = Integer.parseInt(prop.getProperty("bac.helperPort"),16);


    } catch (IOException ex) {
        ex.printStackTrace();
    }

    //Read and decode the information given by console parameters
    if (args.length > 0)
    {
        try
        {
            for (String s: args)
            {
                decodeConsoleParameter(s);
            }
        } catch (Exception e)
        {
            LOG.error("console parameter failed: "+e.getMessage());
        }
    }

    //if a source port was provided
    if (sourceSet)
    {
        try
        {
            //open a new datagram client socket and connect it to the
                ↪ communication thread
            sourceSocket = new DatagramSocket();
            sourceCommunication = new SourceCommunication(sourceSocket,
                ↪ address, sourcePort);
```

```
//if the mirror parameter was not provided bacnet devices are
    ↪ created
if (!mirrorSet)
{
    //the xml file containing the device information is opened
    File inputFile = new File(translatorDir);
    DocumentBuilderFactory dbFactory = DocumentBuilderFactory.
        ↪ newInstance();
    DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
    Document doc = dBuilder.parse(inputFile);
    doc.getDocumentElement().normalize();

    //devices are given as <device>
    NodeList deviceList = doc.getElementsByTagName("device");

    //if no custom network broadcast address is given use the
        ↪ default value for
    //the automation laboratory
    if(doc.getElementsByTagName("broadcast").item(0) != null)
        broadcastAddr = InetAddress.getByName(doc.
            ↪ getElementsByTagName("broadcast").item(0).
            ↪ getTextContent());

    //if more than one device is given, a custom network provider
        ↪ and listener is created
    if(deviceList.getLength() > 1)
    {
        //the ip address of the hardware under test
        controllerAddr = InetAddress.getByName(doc.
            ↪ getElementsByTagName("ip")
                                                .item(0)
                                                .getTextContent
                                                    ↪ ());

        //create a datagram socket that emulates one BACnet port
            ↪ for all broadcast operations
        providerSocket = new DatagramSocket(bacnetPort);

        //start the thread emulating the central network node
        new BacnetNetworkProvider(providerSocket,
            ↪ localDeviceHashMap, controllerAddr).start();
    }
```

```
if(deviceList.getLength()>0)
{
    //iterate through the devices specified in the xml file
    for(int i = 0; i< deviceList.getLength(); i++)
    {
        Node device = deviceList.item(i);

        NodeList objectList = device.getChildNodes();

        Element deviceData = (Element) device;

        //read the id and check if no duplicate device is
            ↪ specified
        int localDeviceID = Integer.parseInt(deviceData.
            ↪ getAttribute("id"));

        if(localDeviceHashMap.containsKey(localDeviceID))
            throw new Exception("Multiple device id");

        //create a new device and store it in the device
            ↪ database
        localDeviceHashMap.put(localDeviceID, new
            ↪ BacnetDevice(sourceCommunication));

        int devicePort = bacnetPort;

        //providerSocket != null indicates more than one device
        if(providerSocket != null)
        {
            //create a thread that provides a capsuled network
                ↪ for the device alone
            localDeviceHashMap.get(localDeviceID).setLocalPort(
                ↪ internalPort);
            new LocalDeviceListener(bacnetPort,
                ↪ localDeviceHashMap.get(localDeviceID).
                ↪ getLocalPort(), broadcastAddr).start();
            devicePort = localDeviceHashMap.get(localDeviceID).
                ↪ getLocalPort();

            //a device always occupies two ports, a send and a
                ↪ receive port
            internalPort += 2;
        }
```

```
//a transport is created for the device linking it into
    ↪ the network
localDeviceHashMap.get(localDeviceID).setTransport(
        new DefaultTransport(new IpNetworkBuilder()
                .withReuseAddress(true)
                .withPort(devicePort)
                .withBroadcast(broadcastAddr.
                    ↪ getHostAddress(),24)
                .build())
);

localDeviceHashMap.get(localDeviceID).setLocalDevice(
    ↪ localDeviceID);

//device properties are set according to the xml file,
    ↪ otherwise default values are used
configureDevice(deviceData, localDeviceHashMap.get(
    ↪ localDeviceID).getLocalDevice());

//iterate through the objects of the device
for(int j=0; j < objectList.getLength(); j++) {
    Node object = objectList.item(j);
    if (object.getNodeName().equals("object") &&
        ↪ object.getNodeType() == Node.
        ↪ ELEMENT_NODE)
    {
        Element objectData = (Element) object;
        int id = Integer.parseInt(objectData.
            ↪ getAttribute("objectId"));

        ObjectType type = decodeObjectType(
            ↪ objectData.getElementsByTagName("type"
            ↪ )
                                                .
                                                    ↪ item
                                                    ↪ (0)
                                                    ↪ .
                                                    ↪ getTextConte
                                                    ↪ ()
                                                    ↪ ,
                                                    ↪
```

```
                                    objectData.getElementsByTagName("
                                        ↪ direction").item(0).getTextContent
                                        ↪ ());

                                if (type==null)
                                    throw new Exception("Wrong object type
                                        ↪ specified");

                                if (localDeviceHashMap.get(localDeviceID).
                                    ↪ objectExists(id))
                                    throw new Exception("Multiple object id");

                                //add the object to the local device and
                                    ↪ configure its properties
                                localDeviceHashMap.get(localDeviceID).
                                    ↪ addObject(id,type,objectData);
                            }
                        }
                    }
                }
            }
        }
        catch (Exception e)
        {
            LOG.error("Device setup failed: "+e.getMessage());
            exit();
        }
    }

    //if the model should be executed by this application
    if (executeSet)
    {
        try
        {
            ProcessBuilder pBuilder = new ProcessBuilder("./model","-rt");
            pBuilder.directory(new File(modelLocation));

            model = pBuilder.start();
        }
        catch (Exception e)
        {
            LOG.error("Model exec failed:"+e.getMessage());
            exit();
```

```
        }
    }

    //if a sink port was given by the parameters
    if(sinkSet)
    {
        try
        {
            //create a datagram server socket and start listening for udp
                ↪ packages from the simulation
            sinkSocket = new DatagramSocket(sinkPort);

            //in mirrored mode only a listener and a echo is implemented
            if(mirrorSet && sourceSet)
                new SinkServerThread(sinkSocket, sourceSocket, sourcePort,
                    ↪ address).start();
            else
                new SinkServerThread(sinkSocket,localDeviceHashMap).start();
        }
        catch (Exception e)
        {
            LOG.error("Sink communication start failed: "+e.getMessage());
            exit();
        }
    }

    //wait for the press of the "enter" key to end the simulation
    BufferedReader reader = new BufferedReader(new InputStreamReader(
        ↪ System.in));
    boolean exit = false;

    System.out.println("Please press:");
    System.out.println("'s' to start simulation");
    System.out.println("'e' to exit");

    try {

        while(!exit)
        {
            System.out.print(">");
            String input = reader.readLine();

            if(input.equals("s"))
```

```
                {
                    System.out.println("Start Simulation");
                    //start the simulation and initialize the output objects to the
                        ↪ present values given in the xml
                    startSimulation();
                }
                else if(input.equals("e"))
                {
                    System.out.println("Exit BACnetHIL-Simulation");
                    exit = true;
                }
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }

        exit();
    }

    /**
     * function decodes the parameters given via the console execution and sets
         ↪ variables accordingly
     * @param s single parameter read from the command line
     * @throws Exception
     */
    private static void decodeConsoleParameter(String s) throws Exception
    {
        if(s.startsWith("-i"))
        {
            String port = s.substring(2);
            port = port.trim();

            sinkPort = Integer.parseInt(port);
            sinkSet = true;
        }
        else if (s.startsWith("-o"))
        {
            String port = s.substring(2);
            port = port.trim();

            sourcePort = Integer.parseInt(port);
            address = InetAddress.getByName("127.0.0.1");
```

```
            sourceSet = true;
        }
        else if (s.startsWith("-m"))
        {
            mirrorSet = true;
        }
        else if(s.startsWith("-t"))
        {
            translatorDir = s.substring(2);
            translatorDir = translatorDir.trim();
        }
        else if(s.startsWith("-s"))
        {
            startStopSet = true;
        }
        else if(s.startsWith("-e"))
        {
            modelLocation = s.substring(2);
            modelLocation = modelLocation.trim();

            executeSet = true;
        }
        else if(s.startsWith("-noexec"))
        {
            executeSet = false;
        }
        else
            usage();
    }

    /**
     * function prints a usage message to the command line
     */
    private static void usage()
    {
        System.out.println("PowerRPDevs-HMI frontend");
        System.out.println("-i[port]... Sink-Port");
        System.out.println("-o[port]... Source-Port");
        System.out.println("-m...mirror udp packages (no bacnet)");
        System.out.println("-t[location]... translation file location");
        System.out.println("-e[location]... model file to execute");
        System.out.println("-s ... send start/stop command");
        System.out.println("-noexec...model file will not be executed");
```

```
        exit();
    }

    /**
     * configures a device with the properties given in the translator file, default
     *     values otherwise
     * @param deviceData data of the device read from the translator xml file
     * @param local the local device to configure
     */
    private static void configureDevice(Element deviceData, LocalDevice local)
    {
        if (deviceData.getElementsByTagName("objectName").item(0) != null)
        {
            String name = deviceData.getElementsByTagName("objectName").item
                (0).getTextContent();
            local.writePropertyInternal(PropertyIdentifier.objectName, new
                CharacterString(name));
        }

        if (deviceData.getElementsByTagName("apduSegmentTimeout").item(0) !=
            null)
        {
            Unsigned16 timeout = new Unsigned16(Integer.parseInt(deviceData.
                getElementsByTagName("apduSegmentTimeout").item(0).
                getTextContent()));
            local.writePropertyInternal(PropertyIdentifier.apduSegmentTimeout,
                timeout);
        }

        if (deviceData.getElementsByTagName("apduTimeout").item(0) != null)
        {
            Unsigned16 timeout = new Unsigned16(Integer.parseInt(deviceData.
                getElementsByTagName("apduTimeout").item(0).getTextContent()
                ));
            local.writePropertyInternal(PropertyIdentifier.apduTimeout, timeout);
        }

        if (deviceData.getElementsByTagName("databaseRevision").item(0) != null)
        {
            Unsigned8 dataRv = new Unsigned8(Integer.parseInt(deviceData.
                getElementsByTagName("databaseRevision").item(0).
                getTextContent()));
```

```
        local.writePropertyInternal(PropertyIdentifier.databaseRevision,dataRv);
            ↪
    }

    if(deviceData.getElementsByTagName("firmwareRevision").item(0) != null)
    {
        CharacterString firmware = new CharacterString(deviceData.
            ↪ getElementsByTagName("firmwareRevision").item(0).
            ↪ getTextContent());
        local.writePropertyInternal(PropertyIdentifier.firmwareRevision,
            ↪ firmware);
    }

    if(deviceData.getElementsByTagName("modelName").item(0) != null)
    {
        CharacterString model = new CharacterString(deviceData.
            ↪ getElementsByTagName("modelName").item(0).getTextContent());
        local.writePropertyInternal(PropertyIdentifier.modelName, model);
    }

    if(deviceData.getElementsByTagName("protocolRevision").item(0) != null)
    {
        Unsigned8 proto = new Unsigned8(Integer.parseInt(deviceData.
            ↪ getElementsByTagName("protocolRevision").item(0).
            ↪ getTextContent()));
        local.writePropertyInternal(PropertyIdentifier.protocolRevision, proto);
    }

    if(deviceData.getElementsByTagName("vendorIdentifier").item(0) != null)
    {
        Unsigned16 vendor = new Unsigned16(Integer.parseInt(deviceData.
            ↪ getElementsByTagName("vendorIdentifier").item(0).getTextContent
            ↪ ()));
        local.writePropertyInternal(PropertyIdentifier.vendorIdentifier, vendor);
            ↪
    }

    if(deviceData.getElementsByTagName("vendorName").item(0) != null)
    {
        CharacterString vendor = new CharacterString(deviceData.
            ↪ getElementsByTagName("vendorName").item(0).getTextContent());
        local.writePropertyInternal(PropertyIdentifier.vendorName, vendor);
    }
```

```
        if (deviceData.getElementsByTagName("maxApduLengthAccepted").item(0)
            ↪ != null)
        {
            Unsigned16 maxApdu = new Unsigned16(Integer.parseInt(deviceData.
                ↪ getElementsByTagName("maxApduLengthAccepted").item(0).
                ↪ getTextContent()));
            local.writePropertyInternal(PropertyIdentifier.maxApduLengthAccepted,
                ↪  maxApdu);
        }

        if (deviceData.getElementsByTagName("maxSegmentsAccepted").item(0) !=
            ↪ null)
        {
            Unsigned16 maxSeg = new Unsigned16(Integer.parseInt(deviceData.
                ↪ getElementsByTagName("maxSegmentsAccepted").item(0).
                ↪ getTextContent()));
            local.writePropertyInternal(PropertyIdentifier.maxSegmentsAccepted,
                ↪ maxSeg);
        }
    }

    /**
     * function starts the simulation by transmitting a START cmd and intitializes
         ↪ the source
     * with the present values of output objects
     */
    private static void startSimulation()
    {
        try
        {
            ByteArrayOutputStream output = new ByteArrayOutputStream(1024);

            if (sourceSet)
            {
                //if the simulation needs to be started manually
                if (startStopSet)
                    startStopOperation(HBASimProtos.CtrlCmd.CmdType.START);

                messages.HBASimProtos.Digital digMsg;
                messages.HBASimProtos.Analog anMsg;
                messages.HBASimProtos.UDPMSG udpmsg;
```

```
output.reset();

//start synchronisation with the powerrpDEVS simulation
if(sourceCommunication != null)
    sourceCommunication.startSyncService();

//iterate through the local devices
for(int devId : localDeviceHashMap.keySet())
{
    //iterate through all objects contained in the device
    for(int objId : localDeviceHashMap.get(devId).
        ↪ getObjectHashMap().keySet())
    {
        //calculate the object id for powerrpDEVS
        //[xxx][xxx] = [DeviceID][ObjectID]
        int id = devId*1000 + objId;
        BACnetObject obj = localDeviceHashMap.get(devId).
            ↪ getObject(objId);
        udpmsg = null;

        //if the object is of type output, create a protobuf
            ↪ message
        if(obj.getId().getObjectType() == ObjectType.
            ↪ analogOutput)
        {
            anMsg = messages.HBASimProtos.Analog.newBuilder().
                ↪ setId(id)
                    .setValue(Double.parseDouble(obj.readProperty(
                        ↪ PropertyIdentifier.presentValue).toString())
                        ↪ ).build();
            udpmsg = messages.HBASimProtos.UDPMSG.
                ↪ newBuilder().setAnMsg(anMsg).build();
        }

        if(obj.getId().getObjectType() == ObjectType.
            ↪ binaryOutput)
        {
            digMsg = messages.HBASimProtos.Digital.newBuilder().
                ↪ setId(id)
                    .setValue(obj.readProperty(PropertyIdentifier.
                        ↪ presentValue).equals(BinaryPV.active)).
                        ↪ build();
```

```
                        udpmsg = messages.HBASimProtos.UDPMSG.
                            ↪ newBuilder().setDigMsg(digMsg).build();
                    }

                    //add the message to the output buffer
                    if (udpmsg != null)
                    {
                        sourceCommunication.addMessageToSendQueue(udpmsg);
                            ↪
                    }
                }
            }
        }
    }catch (Exception e)
    {
        LOG.error("simulation start failed : "+e.getMessage());
        exit();
    }
}

/**
 * translates the string parameters of the object type and direction
 * to bacnet4j types
 * @param type the general type of the object (binary/analog)
 * @param direction the direction of the object (input/output)
 * @return the object type encoded in the bacnet4j type
 *         null if a wrong type was provided
 */
private static ObjectType decodeObjectType(String type, String direction)
{
    if (type.startsWith("analog"))
    {
        if (direction.equals("i"))
            return ObjectType.analogInput;
        else
            return ObjectType.analogOutput;
    }
    else if (type.startsWith("binary"))
    {
        if (direction.equals("i"))
            return ObjectType.binaryInput;
        else
            return ObjectType.binaryOutput;
```

```
        }
        else
            return null;
    }

    /**
     * exit function of the HIL application, BACnet devices are terminated and
     ↪ sockets closed
     * a STOP cmd is sent to the simulation
     */
    private static void exit()
    {
        for(int devId : localDeviceHashMap.keySet())
        {
            localDeviceHashMap.get(devId).getLocalDevice().terminate();
        }

        if(providerSocket != null)
            providerSocket.close();

        if (sinkSocket != null) {
            sinkSocket.close();
        }

        if (sourceSocket != null) {

            if(startStopSet)
            {
                try
                {
                    startStopOperation(HBASimProtos.CtrlCmd.CmdType.STOP);
                }
                catch (Exception e)
                {
                    LOG.error("Model stop operation failed: "+e.getMessage());
                }
            }

            sourceSocket.close();
        }

        System.exit(1);
    }
```

```
    /**
     * sends ctrl commands to the sim to start and stop simulations
     * @param op protobuf cmd (START/STOP)
     * @throws Exception
     */
    private static void startStopOperation(HBASimProtos.CtrlCmd.CmdType op)
        ↪ throws Exception
    {
        messages.HBASimProtos.CtrlCmd startCmd = messages.HBASimProtos.
            ↪ CtrlCmd.newBuilder()
                .setCmd(op)
                .build();

        byte[] output = startCmd.toByteArray();

        DatagramPacket packet = new DatagramPacket(output, output.length,
            ↪ address, sourcePort);

        sourceSocket.send(packet);
    }
}
```

9.1.2 translator.xml

```
<?xml version = "1.0"?>
<bacnetDevices>
      <ip>128.130.56.47</ip>
   <broadcast>128.130.56.127</broadcast>
   <device id = "0">
      <name>Room 1 Control</name>
      <object objectId = "0">
          <name>tempSens1</name>
         <description>temperature sensor room 1</description>
         <type>analog</type>
         <direction>i</direction>
         <default>20.0</default>
          <cov>0.1</cov>
      </object>
      <object objectId = "1">
          <name>heater1</name>
          <description>heater room 1</description>
```

```
            <type>binary</type>
            <direction>o</direction>
            <default>false</default>
        <cov>1</cov>
        </object>
    </device>

    <device id = "1">
    <name>Room 2 Control</name>
    <object objectId = "0">
        <name>tempSens2</name>
        <description>temperature sensor room 2</description>
        <type>analog</type>
        <direction>i</direction>
        <default>20.0</default>
        <cov>0.1</cov>
    </object>
    <object objectId = "1">
        <name>heater2</name>
        <description>heater room 2</description>
        <type>binary</type>
        <direction>o</direction>
        <default>false</default>
       <cov>1</cov>
    </object>
    </device>

    <device id = "2">
    <name>Room 3 Control</name>
    <object objectId = "0">
        <name>tempSens3</name>
        <description>temperature sensor room 3</description>
        <type>analog</type>
        <direction>i</direction>
        <default>20.0</default>
        <cov>0.1</cov>
    </object>
    <object objectId = "1">
        <name>heater3</name>
        <description>heater room 3</description>
        <type>binary</type>
        <direction>o</direction>
        <default>false</default>
```

```
        <cov>1</cov>
    </object>
    </device>
</bacnetDevices>
```

9.1.3 BacnetDevice.java

```
import com.serotonin.bacnet4j.LocalDevice;
import com.serotonin.bacnet4j.event.DeviceEventAdapter;
import com.serotonin.bacnet4j.obj.BACnetObject;
import com.serotonin.bacnet4j.service.Service;
import com.serotonin.bacnet4j.transport.Transport;
import com.serotonin.bacnet4j.type.constructed.Address;
import com.serotonin.bacnet4j.type.constructed.PropertyValue;
import com.serotonin.bacnet4j.type.constructed.StatusFlags;
import com.serotonin.bacnet4j.type.enumerated.*;
import com.serotonin.bacnet4j.type.primitive.CharacterString;
import com.serotonin.bacnet4j.type.primitive.Real;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.w3c.dom.Element;

import java.util.HashMap;

/**
 * class implements starefull operation of a BACnet device
 * it stores the device as well as all parameters neccesarry to use
 * it in the HIL simulation
 *
 * @author Stefan Adelmann <e01633044@student.tuwien.ac.at>
 */
public class BacnetDevice extends DeviceEventAdapter {
    private int localPort;
    private LocalDevice localDevice;
    private Transport transport;
    private HashMap<Integer, BACnetObject> objectHashMap = new HashMap<
        ↪ Integer, BACnetObject>();
    private SourceCommunication sender;
    private static final Logger LOG = LoggerFactory.getLogger(BacnetDevice.class);
        ↪

    /**
```

```
     * constructor of the device
     * @param sender link to the source communication object
     */
    public BacnetDevice(SourceCommunication sender)
    {
        this.sender = sender;
    }

    /**
     * stores the internal port of the device in case there is more than one device
     * this port is used in the capsuled network, the device occupies this port and
     * the port+1 (read and write port)
     *
     * @param port udp port that is open for the device
     */
    public void setLocalPort(int port)
    {
        this.localPort = port;
    }

    /**
     * getter function of the device port
     * @return the internal port of the device
     */
    public int getLocalPort()
    {
        return this.localPort;
    }

    /**
     * create and initialize a new local device affiliated with this object of class
     ↪ BacnetDevice
     * @param id BACnet id of the device
     * @throws Exception
     */
    public void setLocalDevice(int id) throws Exception
    {
        this.localDevice = new LocalDevice(id, transport);
        this.localDevice.initialize();

        //add a ecent handler to this class, class extends EventHandler class of
        ↪ bacnet4j
        this.localDevice.getEventHandler().addListener(this);
```

```
}

/**
 * getter function of local device
 * @return the local device stored in this instance of BacnetDevice
 */
public LocalDevice getLocalDevice()
{
    return this.localDevice;
}

/**
 * set the transport and default values of it
 * @param transport transport instance to be stored
 */
public void setTransport(Transport transport)
{
    this.transport = transport;
    this.transport.setTimeout(500000);
    this.transport.setSegTimeout(15000);
}

/**
 * getter function of the transport variable
 * @return instance of transport
 */
public Transport getTransport()
{
    return this.transport;
}

/**
 * add a new object to the local object storage of the device and connect it to
    ↪ the local device
 * configure the properties of the object according to the data provided by the
    ↪ translator xml file
 * @param id id of the new object
 * @param type type of the new object (analog/binary input/output)
 * @param objectData data read from the xml file
 * @throws Exception
 */
public void addObject(int id, ObjectType type, Element objectData) throws
    ↪ Exception
```

```
{
    this.objectHashMap.put(id, new BACnetObject(this.localDevice,type,id));
    this.localDevice.addObject(this.objectHashMap.get(id));

    BACnetObject obj = this.objectHashMap.get(id);

    if(objectData.getElementsByTagName("name").item(0) != null)
    {
        String name = objectData.getElementsByTagName("name").item(0).
            ↪ getTextContent();
        obj.writePropertyInternal(PropertyIdentifier.objectName,new
            ↪ CharacterString(name));
    }

    if(objectData.getElementsByTagName("default").item(0) != null)
    {
        if(obj.getId().getObjectType() == ObjectType.analogInput || obj.getId
            ↪ ().getObjectType() == ObjectType.analogOutput)
            obj.writePropertyInternal(PropertyIdentifier.presentValue, new Real(
                ↪ Float.parseFloat(objectData.getElementsByTagName("default")
                ↪ .item(0).getTextContent())));
        else
        {
            obj.writePropertyInternal(PropertyIdentifier.presentValue, java.lang
                ↪ .Boolean.parseBoolean(objectData.getElementsByTagName("
                ↪ default").item(0).getTextContent()) ? BinaryPV.active :
                ↪ BinaryPV.inactive);
            obj.writePropertyInternal(PropertyIdentifier.polarity, Polarity.
                ↪ normal);
        }
    }

    if(objectData.getElementsByTagName("description").item(0) != null)
        obj.writePropertyInternal(PropertyIdentifier.description, new
            ↪ CharacterString(objectData.getElementsByTagName("description").
            ↪ item(0).getTextContent()));

    if(objectData.getElementsByTagName("unit").item(0) != null)
        obj.writePropertyInternal(PropertyIdentifier.units, EngineeringUnits.
            ↪ forId(Integer.parseInt(objectData.getElementsByTagName("unit").
            ↪ item(0).getTextContent())));

    if(objectData.getElementsByTagName("cov").item(0) != null)
```

```
        {
            obj._supportCovReporting(new Real(Float.parseFloat(objectData.
                ↪ getElementsByTagName("cov").item(0).getTextContent())),null);
            obj.writePropertyInternal(PropertyIdentifier.covIncrement, new Real(
                ↪ Float.parseFloat(objectData.getElementsByTagName("cov").item(0)
                ↪ .getTextContent())));
        }

        obj.writePropertyInternal(PropertyIdentifier.statusFlags, new StatusFlags(
            ↪ false, false, false, false));
        obj.writePropertyInternal(PropertyIdentifier.eventState, EventState.normal);
    }

    /**
     * getter function of the object storage, contains all objects connected and
         ↪ managed by the local device
     * @return instance of the object storage
     */
    public HashMap<Integer, BACnetObject> getObjectHashMap()
    {
        return objectHashMap;
    }

    /**
     * getter function of a certain object
     * @param id id of the object to be fetched
     * @return object with id
     */
    public BACnetObject getObject(int id)
    {
        return this.objectHashMap.get(id);
    }

    /**
     * check if a object with a given id already exists
     * @param id id to be checked
     * @return true if object exists, false else
     */
    public boolean objectExists(int id)
    {
        return this.objectHashMap.containsKey(id);
    }
```

```
/**
 * function overloads the default event handler of the bacnet device,
 * this function is triggered if a BACnet writeRequest was received by
 * bacnet4j
 * the value contained in the request is written to the object and a
 * protobuf message is created to send the new value to the simulation
 * if the object is of type output
 *
 * @param var1 the address of the caller
 * @param var2 the bacnetobject that is to be written
 * @param var3 the value of the propertie that is to be written
 */
public void propertyWritten(Address var1, BACnetObject var2, PropertyValue
    ↪ var3)
{
    messages.HBASimProtos.Digital digMsg;
    messages.HBASimProtos.Analog anMsg;
    messages.HBASimProtos.UDPMSG udpmsg = null;

    //calculate the powerrpDevs id of the object
    int id = localDevice.getInstanceNumber()*1000 + var2.getId().
        ↪ getInstanceNumber();

    //only create a message if the object is of type output
    if(var2.getId().getObjectType() == ObjectType.analogOutput)
    {
        anMsg = messages.HBASimProtos.Analog.newBuilder().setId(id)
                .setValue(Double.parseDouble(var3.getValue().toString())).build
                    ↪ ();
        udpmsg = messages.HBASimProtos.UDPMSG.newBuilder().setAnMsg(
            ↪ anMsg).build();

        LOG.info("write: AO: "+id+" value: "+anMsg.getValue());
    }
    else if(var2.getId().getObjectType() == ObjectType.binaryOutput)
    {
        digMsg = messages.HBASimProtos.Digital.newBuilder().setId(id)
                .setValue(var3.getValue().equals(BinaryPV.active)).build();

        udpmsg = messages.HBASimProtos.UDPMSG.newBuilder().setDigMsg(
            ↪ digMsg).build();

        LOG.info("write: BO: "+id+" value: "+digMsg.getValue());
```

```
        }

        //if a message is to be sent to the source node of the simulation
        if (udpmsg != null)
        {
            //add the message to the send queue of the sourceCommunication class
            sender.addMessageToSendQueue(udpmsg);
        }
    }

    /**
     * function overloads the default event handler of the bacnet device,
     * this function is triggered if a BACnet request of any kind was received by
     * bacnet4j
     * the purpose of this function is to debug BACnet communication
     * @param from address of the caller
     * @param service service that was requested by the caller
     */
    public void requestReceived(Address from, Service service)
    {
        LOG.info(service.toString());
    }
}
```

9.1.4 BacnetNetworkProvider.java

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.net.*;
import java.util.concurrent.ConcurrentHashMap;

/**
 * class acts as a central network node that provides broadcast capapilities to
    ↪ encapsuleted
 * low level networks of seperate BACnet devices
 * incoming broadcasts on the port of this provider are distributed to the devices
 * outgoing broadcasts are redirected to the broadcast address of the network
 *
 * @author Stefan Adelmann <e01633044@student.tuwien.ac.at>
 */
```

```
public class BacnetNetworkProvider extends Thread {
    private DatagramSocket bacnetSocket;
    private ConcurrentHashMap<Integer, BacnetDevice> localDeviceHashMap;
    private InetAddress address;
    private static final Logger LOG = LoggerFactory.getLogger(
        ↪ BacnetNetworkProvider.class);

    /**
     * constructor of the network provider
     * @param bacnetSocket the global socket all devices should communicate on
     * @param localDeviceHashMap the storage that containes all devices emulated
         ↪ by this application
     * @param address the broadcast address of the network
     */
    public BacnetNetworkProvider(DatagramSocket bacnetSocket,
        ↪ ConcurrentHashMap<Integer, BacnetDevice> localDeviceHashMap,
        ↪ InetAddress address) {
        this.bacnetSocket = bacnetSocket;
        this.localDeviceHashMap = localDeviceHashMap;
        this.address = address;
    }

    public void run(){

        byte[] buffer = new byte[1024];

        try {
            while (true) {

                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                    ↪ //set up packet
                //blocking wait for packages
                bacnetSocket.receive(packet);                    //receive a packet
                    ↪ from a transfer server
                //if the packets originates from the broadcast address
                if(packet.getAddress().equals(address))
                {
                    //distribute the package to every internal network
                    for(int id: localDeviceHashMap.keySet())
                    {
                        int port = localDeviceHashMap.get(id).getLocalPort();
                        packet = new DatagramPacket(packet.getData(), packet.
                            ↪ getLength(), InetAddress.getByName("127.0.0.1"), port
```

```
                        ↪ );

                    bacnetSocket.send(packet);
                }
            }//if the packet originates from a local device
            else if(packet.getAddress().getHostAddress().equals("127.0.0.1"))
            {
                //emulate a packet sent by the central port and send it to the
                    ↪ hardware under test
                int outPort = packet.getPort()+1;
                packet = new DatagramPacket(packet.getData(), packet.
                    ↪ getLength(), address, bacnetSocket.getLocalPort());
                DatagramSocket outputSocket = new DatagramSocket(outPort);
                outputSocket.send(packet);
                outputSocket.close();
            }
        }
    } catch (SocketException e) {
        LOG.error("Network Provider socket failed: "+e.getMessage());
    } catch (IOException e) {
        LOG.error("Network Provider IO failed: "+e.getMessage());
    }
    catch (Exception e)
    {
        LOG.error("Network Provider failed: "+e.getMessage());
    }
  }
}
```

9.1.5 LocalDeviceListener.java

```
import java.io.IOException;
import java.net.*;

/**
 * class implements the capsuled internal network of a single BACnet device
 * all outgoing packets of the device are rerouted by this thread to the broadcast
 * address of the network emulating a single network with a central port
 *
 * @author Stefan Adelmann <e01633044@student.tuwien.ac.at>
```

```
 */
public class LocalDeviceListener extends Thread {
    private DatagramSocket localSocket;
    private int providerPort;
    private InetAddress broadcast;

    /**
     * constructor of the network emulator
     * @param providerPort central port of all devices
     * @param port port of the single device
     * @param broadcast broadcast address of the network
     * @throws Exception
     */
    public LocalDeviceListener(int providerPort,int port, InetAddress broadcast)
        ↪ throws Exception
    {
        this.localSocket = new DatagramSocket(port);
        this.localSocket.setReuseAddress(true);
        this.broadcast = broadcast;
        this.providerPort = providerPort;
    }

    public void run() {

        byte[] buffer = new byte[1024];
        DatagramPacket packet;

        try {
            while (true) {

                packet = new DatagramPacket(buffer, buffer.length); //set up packet

                //blocking wait for packets
                localSocket.receive(packet);                    //receive a packet from
                    ↪ a transfer server

                //redirect the package to the broadcast address and the central port
                packet = new DatagramPacket(packet.getData(), packet.getLength(),
                    ↪ broadcast, providerPort);

                DatagramSocket outputSocket = new DatagramSocket(localSocket.
                    ↪ getLocalPort()+1);
                outputSocket.send(packet);
```

```
                outputSocket.close();
            }
        } catch (SocketException e) {
            System.out.println("Error while listening on UDP-socket: " + e.
                ↪ getMessage());
        } catch (IOException e) {
            System.out.println("IO error: "+e.getMessage());
        }
        catch (Exception e)
        {

        }
    }
}
```

9.1.6 SinkServerThread.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.util.concurrent.ConcurrentHashMap;
import com.google.protobuf.InvalidProtocolBufferException;
import com.serotonin.bacnet4j.exception.BACnetServiceException;
import com.serotonin.bacnet4j.type.constructed.PropertyValue;
import com.serotonin.bacnet4j.type.constructed.ValueSource;
import com.serotonin.bacnet4j.type.enumerated.BinaryPV;
import com.serotonin.bacnet4j.type.enumerated.PropertyIdentifier;
import com.serotonin.bacnet4j.type.primitive.Real;
import messages.HBASimProtos;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * class implements a thread that listens for incoming packages from the sink of the
 * powerrpDEVS simulation
 * incoming protobuf messages are decoded and the values are written to the objects
    ↪ of
 * the BACnet emulation
```

```
 *
 * @author Stefan Adelmann <e01633044@student.tuwien.ac.at>
 */
public class SinkServerThread extends Thread {

    private DatagramSocket sinkSocket;
    private DatagramSocket sourceSocket;
    private InetAddress address;
    private int sourcePort;
    private ConcurrentHashMap<Integer, BacnetDevice> localDeviceHashMap = null
        ↪ ;
    private static final Logger LOG = LoggerFactory.getLogger(SinkServerThread.
        ↪ class);

    /**
     * constructor of the class for normal operation including BACnet
     * @param sinkSocket the socket to listen on
     * @param localDeviceHashMap the list of all devices emulated by this
         ↪ application
     */
    public SinkServerThread(DatagramSocket sinkSocket, ConcurrentHashMap<
        ↪ Integer, BacnetDevice> localDeviceHashMap) {
        this.sinkSocket = sinkSocket;
        this.localDeviceHashMap = localDeviceHashMap;
    }

    /**
     * constructor for mirrored operation
     * packets are received from the sink and retransmitted to the source
     * @param sinkSocket the socket to listen on
     * @param sourceSocket the socket of the source to retransmit the packets to
     * @param sourcePort the port of the source
     * @param address the address of the powerrpDEVS simulation
     */
    public SinkServerThread(DatagramSocket sinkSocket, DatagramSocket
        ↪ sourceSocket, int sourcePort, InetAddress address) {
        this.sinkSocket = sinkSocket;
        this.sourceSocket = sourceSocket;
        this.address = address;
        this.sourcePort = sourcePort;
    }

    public void run() {
```

```
byte[] buffer;
DatagramPacket packet;
int deviceId;
int objectId;

try {
    while (true) {
        buffer = new byte[1024];

        packet = new DatagramPacket(buffer, buffer.length); //set up packet

        //blocking wait for packets
        sinkSocket.receive(packet);                    //receive a packet from
            ↪ a transfer server

        ByteArrayInputStream input = new ByteArrayInputStream(packet.
            ↪ getData());
        ByteArrayOutputStream output = new ByteArrayOutputStream
            ↪ (1024);

        HBASimProtos.UDPMSG msg;

        try
        {
            //read messages from the input stream
            while((msg= HBASimProtos.UDPMSG.parseDelimitedFrom(
                ↪ input)) != null)
            {
                switch(msg.getHMICase())
                {
                    case ANMSG:
                        //LOG.info("analog: id="+msg.getAnMsg().getId() +
                            ↪ " value="+msg.getAnMsg().getValue());

                        if (sourceSocket != null)
                        {
                            //mirror the packet to the source
                            HBASimProtos.UDPMSG udpmsg =
                                ↪ HBASimProtos.UDPMSG.newBuilder().
                                ↪ setAnMsg(msg.getAnMsg()).build();

                            udpmsg.writeDelimitedTo(output);
```

```
                output.flush();
            }
            else
            {
                //calculate the device and object id from the
                    ↪ powerrpDEVS id
                deviceId = msg.getAnMsg().getId()/1000;
                objectId = msg.getAnMsg().getId()%1000;

                if(localDeviceHashMap.containsKey(deviceId))
                    if(localDeviceHashMap.get(deviceId).
                        ↪ objectExists(objectId))
                        localDeviceHashMap.get(deviceId).
                            ↪ getObject(objectId).writeProperty
                            ↪ (new ValueSource(),new
                            ↪ PropertyValue(PropertyIdentifier.
                            ↪ presentValue, new Real((float)msg.
                            ↪ getAnMsg().getValue())));
            }

            break;

        case DIGMSG:
            //LOG.info("digital: id="+msg.getDigMsg().getId()
                ↪ + " value="+msg.getDigMsg().getValue());

            if(sourceSocket != null)
            {
                //mirror the packet to the source
                HBASimProtos.UDPMSG udpmsg =
                    ↪ HBASimProtos.UDPMSG.newBuilder().
                    ↪ setDigMsg(msg.getDigMsg()).build();

                udpmsg.writeDelimitedTo(output);

                output.flush();
            }
            else
            {
                //calculate the device and object id from the
                    ↪ powerrpDEVS id
                deviceId = msg.getDigMsg().getId()/1000;
```

```
                                objectId = msg.getAnMsg().getId()%1000;

                                if (localDeviceHashMap.containsKey(deviceId))
                                    if (localDeviceHashMap.get(deviceId).
                                        ↪ objectExists(objectId))
                                        localDeviceHashMap.get(deviceId).
                                            ↪ getObject(objectId).writeProperty
                                            ↪ (new ValueSource(), new
                                            ↪ PropertyValue(PropertyIdentifier.
                                            ↪ presentValue, msg.getDigMsg().
                                            ↪ getValue() ? BinaryPV.active :
                                            ↪ BinaryPV.inactive));

                                }

                                break;
                            }
                        }
                    }catch (InvalidProtocolBufferException e)
                    {
                        //no more data to read
                    }
                    catch (BACnetServiceException e)
                    {
                        LOG.error("BACnet Service Exception: "+e.getMessage());
                    }

                    //transmit the mirror data to the source
                    if (sourceSocket != null)
                    {
                        packet = new DatagramPacket(output.toByteArray(), output.size
                            ↪ (), address, sourcePort);
                        sourceSocket.send(packet);
                    }
                }
            } catch (SocketException e) {
                LOG.error("Sink Listener Socket exception: "+e.getMessage());
            } catch (IOException e) {
                LOG.error("Sink Listener exception: "+e.getMessage());
            }
        }
    }
```

9.1.7 SourceCommunication.java

```
import messages.HBASimProtos;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.util.concurrent.*;

/**
 * class provides a centralized communication service to send messages to the source
 * of the powerrpDEVS simulation
 *
 * the communication is synchronized to the source and messages are buffered till
    ↪ the
 * source is ready to receive messages
 *
 * @author Stefan Adelmann <e01633044@student.tuwien.ac.at>
 */
public class SourceCommunication {
    private DatagramSocket sourceSocket;
    private DatagramSocket syncSocket;
    private ConcurrentLinkedQueue<HBASimProtos.UDPMSG> msgBuffer;
    private InetAddress address;
    private int port;
    private boolean sourceReady = false;
    private Executor sendExecutor;
    private static final Logger LOG = LoggerFactory.getLogger(
        ↪ SourceCommunication.class);

    /**
     * constructor of the class
     * @param sourceSocket socket of the powerrpDEVS source
     * @param address address of the source
     * @param port port of the source
     * @throws Exception
     */
    public SourceCommunication(DatagramSocket sourceSocket, InetAddress address,
        ↪ int port) throws Exception
```

```
{
    this.sourceSocket = sourceSocket;
    msgBuffer = new ConcurrentLinkedQueue<>();
    this.address = address;
    this.port = port;

    //the sync socket listens on port 2013
    syncSocket = new DatagramSocket(2013);

    //a thread poll with size 1 is used to shedule the message sending in the
        ↪ background
    sendExecutor = Executors.newSingleThreadExecutor();
}

/**
 * function adds a message to the transmit buffer
 * if the source is already ready to receive messages the message is sent
 * immediately
 *
 * @param msg message to be sent
 */
public void addMessageToSendQueue(HBASimProtos.UDPMSG msg)
{
    msgBuffer.add(msg);

    if(sourceReady)
        sendExecutor.execute(this::sendThread);
}

/**
 * Thread that handles the actual sending of the messages to the source
 */
private void sendThread()
{
    ByteArrayOutputStream output = new ByteArrayOutputStream(1024);
    try {
        while (msgBuffer.size() > 0)
        {
            HBASimProtos.UDPMSG msg = msgBuffer.poll();
            msg.writeDelimitedTo(output);
        }
```

```
            DatagramPacket packet = new DatagramPacket(output.toByteArray(),
                ↪ output.size(), address, port);

            sourceReady = false;
            sourceSocket.send(packet);
        }
        catch (IOException e)
        {
            LOG.error("Send Thread failed: "+e.getMessage());
        }
    }

    /**
     * function starts the synchronization with the powerrpDEVS source
     * @throws Exception
     */
    public void startSyncService() throws Exception
    {
        Thread t = new Thread() {
            public void run() {
                powerrpDevsSourceSync();
            }
        };
        t.start();
    }

    /**
     * Thread that handles the sync with the powerrpDEVS source
     */
    public void powerrpDevsSourceSync()
    {
        byte[] buffer;
        DatagramPacket packet;

        try {
            while (true) {
                buffer = new byte[1024];

                packet = new DatagramPacket(buffer, buffer.length); //set up packet

                //blocking wait for a ready message
                syncSocket.receive(packet);                    //receive a packet from
                    ↪ a transfer server
```

```
                //source is ready to receive
                sourceReady = true;

                //check if there are messages to send, shedule the send thread if
                    ↪ there are
                if(msgBuffer.size()>0)
                    sendExecutor.execute(this::sendThread);
            }
        } catch (SocketException e) {
            LOG.error("Sync socket failed: "+ e.getMessage());
        } catch (IOException e) {
            LOG.error("Sync operation failed: "+e.getMessage());
        }
    }
}
```

9.2 Simulation Backend

The implementation in this thesis is based on existing function blocks that were extended. Code that was edited is marked by a doxygen comment for the function that specifies the author.

9.2.1 udp_source.cpp

```
#include "udp_source.h"
using namespace google::protobuf::io;
// custom code:
#include "root_simulator.h"
#include "root_coupling.h"

/**
@brief function sets up the udp-port and waits for a start signal if configured

The parameters are given by the GUI configuration window in PowerRPDEVS.
Primary parameters for HIL are the UDP port and if the simulation has to wait for
a START command to be received.

Edited for BACnet HIL Simulation bachelor thesis
@authtor Stefan Adelmann <e01633044@student.tuwien.ac.at>
**/
void udp_source::init(double t ,...) {
```

```
//The 'parameters' variable contains the parameters transferred from the editor.
va_list parameters;
va_start(parameters,t);
//To get a parameter: %Name% = va_arg(parameters,%Type%)
//where:
//        %Name% is the parameter name
// %Type% is the parameter type
printLogAtLevel(_INFO, "%s, %d, t=%G: Init \n",name,myself,t);

////////////////////////////////////////////////////////////
/*** Read parameters and initialize state variables here ***/
////////////////////////////////////////////////////////////

/* Parameters are extracted from 'parameters' by
%Name% = va_arg(parameters,%Type%) in the order of their
occurence in the parameters dialogue.
For example: to extract a parameter of type int, type:
int par = va_arg(parameters, int); (does not realy work properly)
If the next parameter is of type String, it can be fetched
by:
std::istringstream strstr;
strstr.str(std::string(va_arg(parameters,char*)));
strstr.str() returns the String containing the parameter.
Any other of type coded in the string can be extracted with ">>":
for example:
    double d;
    int i;
    strstr >> d >> i;
... if the string contains a floating point value followed by an
integer, separated by a space. With floating point values, the '.'
is used as comma-separation.

When reusing 'strstr' to retriev further string parameters, it has
to be cleared first. For example:
    strstr.clear();
    strstr.str(std::string(va_arg(parameters,char*)));
    double x;
    strstr >> x;
*/
std::istringstream strstr;
strstr.str(va_arg(parameters,char*));
strstr >> UDP_port;
std::string udp_ss(va_arg(parameters,char*));
```

```
    if( 0 == udp_ss.compare("true") || 0 == udp_ss.compare("TRUE") || 0 ==
        ↪ udp_ss.compare("True")) {
      udp_start = true;
    } else {
      udp_start = false;
    }
     if(udp_start) { // wait blocking for initial start signal
      using boost::asio::ip::udp;
        proto::CtrlCmd start_cmd;

      udp::socket socket(io_serv, udp::endpoint(udp::v4(), UDP_port)); //control
          ↪ message port
      udp::endpoint remote_endpoint;
      std::array<char, 256> recv_buf;
      printLogAtLevel(_ERROR,"%s, %d, t=%G: Wait for start signal!\n",name,
          ↪ myself, t);
      while(1) {
        size_t len = socket.receive_from(boost::asio::buffer(recv_buf),
            ↪ remote_endpoint);
         if (!start_cmd.ParseFromArray(recv_buf.data() , len) ) {
          printLogAtLevel(_WARNING,"%s, %d, t=%G: init: expected to receive
              ↪ control message, but received something else! \n",name, myself, t);
         } else {
          if(proto::CtrlCmd::START == start_cmd.cmd() ) {
            break;
          }
          printLogAtLevel(_WARNING,"%s, %d, t=%G: init: expected to control
              ↪ message start, but received something else! \n",name, myself, t);
         }
      }
     }

    UDP_buffer.clear();
    // set sigma initially zero to create the upd-server-thread inside the state
        ↪ transition function
    sigma = 0;
    rootS = NULL;

     // always end the init-function with:
     va_end(parameters);
}

double udp_source::ta(double t) {
```

```
    //This function returns a double.
    /* return double value with the time to the next internal event here. */

    printLogAtLevel(_INFO, "%s, %d, t=%G: ta \n", name,myself,t);

    if(t<0) { // if simulation time t is negative, it means that ta(t)
        // was called directly by the root_simulator, because a UDP message
        // arrived. The UDP server thread stored the time of the arrival
        // (just now) into t_rec.
        tn = t_rec;
    }
    return(sigma);
}
void udp_source::delta(double t) {
    printLogAtLevel(_INFO, "%s, %d, t=%G: delta \n", name,myself,t);
    /* All messages to be sent have to be instances of a class which is
    derived from class DEVSMessage. For QSS-Signals, there is the
    class QSSDoubleArray. For Entities, there is the class 'Entity'.
    Further classes can be derived if needed.
    Since signals are in general vectorial, every input message has
    an attribute 'index', which tells the index inside the vector.
    The default case is 'index=0'.
    Therefore, 'x_b[port][index]' is a 'std::vector<DEVSMessage>',
    containing a list of messages arrived at port 'port' with index
    'index'. However, 'x_b[port][index]' does only exist, if there
    arrived at least one message at port 'port' with index 'index'.
    Otherwise, 'x_b[port].empty() == true'.
    'x_b[port]' does only exist, if input port 'port' is connected
    to some other port within the corresponding coupled model.

    Thus, the easiest way to access input messages at input port
    'port' is to type:

        DEVSMessage *in_msg;
        if(pop_input(port,&in_msg)) {
            DerivedClass *input = (DerivedClass*)in_msg;
            ...
        }

    The index of the input message can then be determined via:
        'input->index'

    The function 'pop_input' can be called repeatedly and returns
```

```
true, as long as there is an input message at the given port,
which has not been popped so far. After calling 'pop_input'
succesfully, 'in_msg' points to the popped input message.
Otherwise, 'in_msg=NULL'.

The input bag will be cleared after the call of 'delta'
(= this function). Therefore, if the input message shall be
stored beyond this function call (e.g. in the systems' state),
a deep copy has to be made. This is done by:

    // inside the state - section (left):
       DerivedClass *input;
    //

    input = (DerivedClass*)in_msg->getCopy();

However, in this case you have to remember to deallocate
the corresponding memory, before you reasign 'input' the
next time. For example:

    if(NULL!=input) {
       delete input;
       input = NULL,
    }
    input = (DerivedClass*)in_msg->getCopy();

 */

if(t + (tn-t) > t) { // if pure external transition
    // DEVSMessage *in_msg;
    // if(pop_input(0,&in_msg)) // if input message at input port 0
    // phase = newphase
    // sigma = ...
} else if(!input_bag_empty()) { // if confluent transition
    // DEVSMessage *in_msg;
    // if(pop_input(0,&in_msg)) // if input message at input port 0
    // phase = newphase
    // sigma = ...
} else { // if pure internal transition

    if(NULL == rootS) { // get pointer to root_simulator, to be able to create a
        ↪ upd-server-thread a register it
       rootS = getRootSimulator(this);
```

```
            //RootCoupling* rootC;
            //Coupling* fath = this->father;

            //while(NULL != fath->father) {
            // fath = fath->father;
            //}
            //rootC = (RootCoupling*)fath;
            //rootS = rootC->rootSim;
            if(udp_start) {
                if(rootS->timed) {
                    rootS->tf = INF;
                }
                udp_start = false;
            }
        }

        if(rootS->timed) { // if the simulaiton is timed, create a new UDP server
            ↪ thread, waiting for udp messages
            proto::CtrlCmd start_cmd;
            if(start_cmd.ParseFromString(UDP_buffer.c_str()) )
            {
                if(proto::CtrlCmd::STOP == start_cmd.cmd() ) {
                    rootS->tf=t;
                }

                UDP_buffer.clear();
            }

            if(NULL == rootS->UDP_thread_ptr) {
//
    ↪ ////////////////////////////////////////////////////////////////////////////////
    ↪
//////////////////////// new code for setting thread priority
    ↪ ////////////////////
//
    ↪ ////////////////////////////////////////////////////////////////////////////////
    ↪

                boost::thread::attributes attrs;
                // set thread priority of udp server thread to maximum
                #if defined(BOOST_THREAD_PLATFORM_WIN32)
                    // ... window version
```

```
    BOOL res = SetThreadPriority(attrs.native_handle(),
        ↪ THREAD_PRIORITY_TIME_CRITICAL);
    if(res == FALSE) {
      int err = GetLastError();
      printLogAtLevel(_ERROR,"%s, %d, t=%G: failed to set scheduler
          ↪ priority for UDP-server task! error code: %d\n",name, myself
          ↪ , t, err);
    }
#elif defined(BOOST_THREAD_PLATFORM_PTHREAD)
    // ... pthread version
    struct sched_param param;
    int res, policy;

     if ((policy = sched_getscheduler(getpid())) == -1)
     {
       printLogAtLevel(_ERROR,"%s, %d, t=%G: failed to get scheduler
           ↪ policy!\n",name, myself, t);
         perror("sched_getscheduler");
     }

    if ((res = sched_getparam(getpid(), &param)) == -1)
    {
      printLogAtLevel(_ERROR,"%s, %d, t=%G: failed to get scheduler
          ↪ parameter!\n",name, myself, t);
      perror("sched_getparam");
    }
    if(-1 == (param.sched_priority = sched_get_priority_max(policy)))
        ↪ {
      printLogAtLevel(_ERROR,"%s, %d, t=%G: failed to get max
          ↪ priority!\n",name, myself, t);
      perror("sched_getparam");
    }

     res=pthread_attr_setschedpolicy(attrs.native_handle(), policy);
     if(res) {
       printLogAtLevel(_ERROR,"%s, %d, t=%G: failed to set scheduler
           ↪ policy with error number: %d\n",name, myself, t, res);
     }
     res=pthread_attr_setschedparam(attrs.native_handle(), &param);
     if(res) {
       printLogAtLevel(_ERROR,"%s, %d, t=%G: failed to set scheduler
           ↪ parameter with error number: %d\n",name, myself, t, res);
     }
```

```
                #else
                #error "Boost threads unavailable on this platform"
                #endif

//
    ↪ ////////////////////////////////////////////////////////////////////////////////
    ↪
//
    ↪ ////////////////////////////////////////////////////////////////////////////////
    ↪
//
    ↪ ////////////////////////////////////////////////////////////////////////////////
    ↪

                    // create thread
                    rootS->UDP_thread_ptr = new boost::thread(attrs, boost::bind(&
                        ↪ UDP_Server_Thread,&UDP_buffer, UDP_port, &rootS->
                        ↪ t_start,&t_rec,&io_serv, &mtx, &(rootS->udpReceived)));
                    mtx.lock();
                    mtx.lock();
                    mtx.unlock();
                    // register thread belonging to this instance of Simulator
                    rootS->thread_creating_blocks_map[rootS->UDP_thread_ptr->
                        ↪ get_id()] = (Simulator*)this;
                    if (LOG_LEVEL==_DEBUG_ROUGH) {
                      std::ostringstream ostrstr;
                      ostrstr<<rootS->UDP_thread_ptr->get_id();
                      printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: delta,
                          ↪ created thread with id: %s \n",name,myself,t,ostrstr.str().
                          ↪ c_str());
                    }
                }
            }

            sigma = INF; // internal events are only triggered by arriving UDP messages
        }
    }

    /**
    @brief lambda function is responsible for reception and interpretation of UDP
        ↪ messages
```

```
Messages have to be extracted from a global buffer and the information has to be
    ↪ decoded
from the protobuf message format.

Edited for the BACnet HIL Simulation bachelor thesis
@author Stefan Adelmann <e01633044@student.tuwien.ac.at>
**/
void udp_source::lambda(double t) {
  //This function returns an Event:
  //      Event(%&Value%, %NroPort%)
  //where:
  //      %&Value% points to the variable which contains the value.
  //      %NroPort% is the port number (from 0 to n-1)

  printLogAtLevel(_INFO, "%s, %d, t=%G: lambda \n",name,myself,t);

  /* This function corresponds to the RPDEVS-function lambda and
  is supposed to produce output (or not). It may be called several
  times at same simulation time, before a state transition is
  conducted. Therefore, do not change state variables inside here!
  Calculate output values due to the received input values (x_b) and
  due to the current values of the state variables.

  Output Messages have to be instances of classes which are derived
  from DEVSMessage, as for example QSSDoubleArray, which is used for
  QSS-signals. Add an output message at output port 'i' by typing for
  example:

    QSSDoubleArray output;

    add_output(&output, i);

  To see how to access input messages, see comments in 'Internal transition '.

   */
  if(t + (tn-t) > t) { // if pure external transition
    // DEVSMessage *in_msg;
    // if(pop_input(0,&in_msg)) // if input message at input port 0
  } else if(!input_bag_empty()) { // if confluent transition
    // DEVSMessage *in_msg;
    // if(pop_input(0,&in_msg)) // if input message at input port 0

  } else { // if pure internal transition
```

```
std::string UDP_buffer_cp(UDP_buffer);
printLogAtLevel(_DEBUG_ROUGH,"%s, %d, t=%G: lambda: UDP message
    ↪ received! \n",name, myself, t);

  proto::UDPMSG udp_msg;

  //convert the string buffer to a Input stream
  ArrayInputStream rawIn(UDP_buffer_cp.c_str(), 1024);

  //stream has to be encoded to parse protobuf messages
CodedInputStream codedIn(&rawIn);

uint32_t size = 0;
bool end = false;

  while(!end)
  {
   //read the size of the next message
   codedIn.ReadVarint32(&size);

    if(size != 0)
    {
      //set the limit for the next read operation according to the message
          ↪ size
      CodedInputStream::Limit limit = codedIn.PushLimit(size);

      //parse the message from the stream
      udp_msg.ParseFromCodedStream(&codedIn);

      //pop the number of bytes that are occupied by the message
      codedIn.PopLimit(limit);

      switch(udp_msg.HMI_case())
      {
        case proto::UDPMSG::kCtrl:
          printLogAtLevel(_DEBUG_ROUGH,"%s, %d, t=%G: lambda:
              ↪ received control message! \n",name, myself, t);
          break;

        case proto::UDPMSG::kDigMsg:
          //binary message
          printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: lambda:
              ↪ digital: id: %d, value: %d \n",name,myself,t,udp_msg.digmsg
```

```
                    ↪ ().id(), udp_msg.digmsg().value());

                output = udp_msg.digmsg().value();

                output.t_exp = t;
                add_output(&output,(unsigned int)1, udp_msg.digmsg().id());

                break;

            case proto::UDPMSG::kAnMsg:
                //analog message
                printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: lambda:
                    ↪ analog: id: %d, value: %f \n",name,myself,t,udp_msg.anmsg
                    ↪ ().id(), udp_msg.anmsg().value());

                output = udp_msg.anmsg().value();
                output.t_exp = t;

                add_output(&output,(unsigned int)0, udp_msg.anmsg().id());

                break;

            default:
                printLogAtLevel(_ERROR,"%s, %d, t=%G: lambda: received
                    ↪ wrong message! \n",name, myself, t);
                end = true;
                break;
            }
        }
        else
        {
            end = true;
        }
      }
    }
}

void udp_source::exit() {
    //Code executed at the end of the simulation.
    printLogAtLevel(_INFO, "%s, %d: exit \n", name,myself);
    io_serv.stop();

    if(NULL != rootS->UDP_thread_ptr) {
```

```
        printLogAtLevel(_DEBUG_ROUGH,"%s: exit(): UDP_thread_ptr!=NULL\n",
            ↪ name);
        if(rootS->UDP_thread_ptr->joinable()) {
            printLogAtLevel(_DEBUG_ROUGH,"%s: exit(): UDP_thread joinable!\n",
                ↪ name);
            rootS->UDP_thread_ptr->join();
            rootS->UDP_thread_ptr = NULL;
            printLogAtLevel(_DEBUG_ROUGH,"%s: exit(): UDP_thread joined!\n",
                ↪ name);
        }
    }
}
```

9.2.2 udp_sink.cpp

```
#include "udp_sink.h"
using namespace google::protobuf::io;

//namespace google::protobuf::io

// custom code:
void udp_sink::init(double t ,...)  {
    //The 'parameters' variable contains the parameters transferred from the editor.
    va_list parameters;
    va_start(parameters,t);
    //To get a parameter: %Name% = va_arg(parameters,%Type%)
    //where:
    //       %Name% is the parameter name
    // %Type% is the parameter type
    printLogAtLevel(_INFO, "%s, %d, t=%G: Init \n",name,myself,t);

    ////////////////////////////////////////////////////////////
    /*** Read parameters and initialize state variables here ***/
    ////////////////////////////////////////////////////////////

    /* Parameters are extracted from 'parameters' by
    %Name% = va_arg(parameters,%Type%) in the order of their
    occurence in the parameters dialogue.
    For example: to extract a parameter of type int, type:
    int par = va_arg(parameters, int); (does not realy work properly)
    If the next parameter is of type String, it can be fetched
    by:
```

```
    std::istringstream strstr;
    strstr.str(std::string(va_arg(parameters,char*)));
    strstr.str() returns the String containing the parameter.
    Any other of type coded in the string can be extracted with ">>":
    for example:
        double d;
        int i;
        strstr >> d >> i;
    ... if the string contains a floating point value followed by an
    integer, separated by a space. With floating point values, the '.'
    is used as comma-separation.

    When reusing 'strstr' to retriev further string parameters, it has
    to be cleared first. For example:
        strstr.clear();
        strstr.str(std::string(va_arg(parameters,char*)));
        double x;
        strstr >> x;
     */
    std::istringstream strstr;
    strstr.str(va_arg(parameters,char*));
    strstr >> dest_IP;
    strstr.str(va_arg(parameters,char*));
    strstr.seekg(0);
    strstr >> UDP_port;

    printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: Init: dest_IP=%s,
        ↪ UDP_port=%s \n",name,myself,t,dest_IP.c_str(), UDP_port.c_str());
     // always end the init-function with:
     va_end(parameters);
}
double udp_sink::ta(double t) {
    //This function returns a double.
    printLogAtLevel(_INFO, "%s, %d, t=%G: ta \n", name,myself,t);
    /* return double value with the time to the next internal event here. */
    return(INF);
}

/**
@brief delta function sends UDP messages to the Simulation-Frontend

Input values of the block are encoded as protobuf messages and sent to the
Frontend as delimited messages.
```

```
Edited for the BACnet HIL Simulation bachelor thesis
@author Stefan Adelmann <e01633044@student.tuwien.ac.at>
**/
void udp_sink::delta(double t) {
    printLogAtLevel(_INFO, "%s, %d, t=%G: delta \n", name,myself,t);
    /* All messages to be sent have to be instances of a class which is
    derived from class DEVSMessage. For QSS-Signals, there is the
    class QSSDoubleArray. For Entities, there is the class 'Entity'.
    Further classes can be derived if needed.
    Since signals are in general vectorial, every input message has
    an attribute 'index', which tells the index inside the vector.
    The default case is 'index=0'.
    Therefore, 'x_b[port][index]' is a 'std::vector<DEVSMessage>',
    containing a list of messages arrived at port 'port' with index
    'index'. However, 'x_b[port][index]' does only exist, if there
    arrived at least one message at port 'port' with index 'index'.
    Otherwise, 'x_b[port].empty() == true'.
    'x_b[port]' does only exist, if input port 'port' is connected
    to some other port within the corresponding coupled model.

    Thus, the easiest way to access input messages at input port
    'port' is to type:

        DEVSMessage *in_msg;
        if(pop_input(port,&in_msg)) {
            DerivedClass *input = (DerivedClass*)in_msg;
            ...
        }

    The index of the input message can then be determined via:
        'input->index'

    The function 'pop_input' can be called repeatedly and returns
    true, as long as there is an input message at the given port,
    which has not been popped so far. After calling 'pop_input'
    succesfully, 'in_msg' points to the popped input message.
    Otherwise, 'in_msg=NULL'.

    The input bag will be cleared after the call of 'delta'
    (= this function). Therefore, if the input message shall be
    stored beyond this function call (e.g. in the systems' state),
    a deep copy has to be made. This is done by:
```

```
    // inside the state - section (left):
       DerivedClass *input;
    //

    input = (DerivedClass*)in_msg->getCopy();

However, in this case you have to remember to deallocate
the corresponding memory, before you reasign 'input' the
next time. For example:

    if(NULL!=input) {
       delete input;
       input = NULL,
    }
    input = (DerivedClass*)in_msg->getCopy();

 */
if(t + (tn-t) > t) { // if pure external transition
    // open UPD socket:
    printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: delta: external
        ↪ transition \n", name,myself,t);
    using boost::asio::ip::udp;
    udp::endpoint receiver_endpoint;
    udp::socket socket(io_serv);

    //try to resolve a route to the recipient of the messages
    try {
       udp::resolver resolver(io_serv);
          udp::resolver::query query(udp::v4(), dest_IP, UDP_port, boost::asio::ip
              ↪ ::resolver_query_base::flags());
       receiver_endpoint = *resolver.resolve(query);

       //open a socket to communicate directly to the recipient
       socket.open(udp::v4());
    } catch (std::exception& e) {
       printLogAtLevel(_ERROR, "open socked failed: %s\n", e.what());
       receiver_endpoint.address(boost::asio::ip::address::from_string(dest_IP));
       std::istringstream istrstr (UDP_port);
       unsigned short int port;
       istrstr >>port;
       receiver_endpoint.port(port);
       printf("%s\n",e.what());
```

```
        fflush (stdout);

    std::stringstream error_msg;
        error_msg << name << " t=" << t << ": Could not open socket (
            ↪ destination IP: "<< dest_IP << ", destination port: "<<
            ↪ UDP_port <<")! Returned error message is: "<< e.what() <<"\n"
            ↪ ;
    throwException(error_msg.str());
}
printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: delta: UDP socket open
    ↪ \n", name,myself,t);

// check received input messages, encode them and send them via UDP
DEVSMessage *in_msg;
unsigned int in_port;

boost::asio::streambuf stream_buffer;
std::ostream output_stream(&stream_buffer);

unsigned char buffer[1024];
//create a raw stream as input for the coded stream that is needed to send
    ↪ protobuf msg
ZeroCopyOutputStream* rawOut = new ArrayOutputStream(buffer, sizeof(
    ↪ buffer));
CodedOutputStream* codedOut = new CodedOutputStream(rawOut);

proto::UDPMSG msg;

while(pop_input(&in_msg,in_port)) {
    printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: delta: popped input
        ↪ message \n", name,myself,t);

    if(0==in_port) { // if msg arrived at analog input port
        proto::Analog analogMsg;

        QSSDoubleArray output = (*dynamic_cast<QSSDoubleArray*>(in_msg)
            ↪ );

        //set the id and the value of the analog message
        analogMsg.set_id(in_msg->index);
        analogMsg.set_value(output[0]);
```

```
        printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: delta: analog: id:
            ↪ %d, value: %d \n",name,myself,t,in_msg->index, analogMsg.value
            ↪ ());

        //insert the analog message in the UDP message capsule
        msg.set_allocated_anmsg(&analogMsg);

        //write the size of the message followed by the message to the stream
        codedOut->WriteVarint32(msg.ByteSize());
        msg.SerializeToCodedStream(codedOut);

        msg.release_anmsg();

    } else { // if msg arrived at digital input port

        proto::Digital digitalMsg;

        QSSDoubleArray output = (*dynamic_cast<QSSDoubleArray*>(in_msg)
            ↪ );

        //set the id and the value of the digital message
        digitalMsg.set_value(output[0]);
        digitalMsg.set_id(in_msg->index);

        printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: delta: digital: id:
            ↪ %d, value: %d \n",name,myself,t,in_msg->index, digitalMsg.value
            ↪ ());

        //insert the digital message into the UDPMSG capsule
        msg.set_allocated_digmsg(&digitalMsg);

        //write the size of the message followed by the message to the stream
        codedOut->WriteVarint32(msg.ByteSize());
        msg.SerializeToCodedStream(codedOut);

        msg.release_digmsg();
    }
}

if(codedOut != NULL)
    delete codedOut;

if(rawOut != NULL)
```

```
            delete rawOut;

        if(socket.is_open()) {

            socket.send_to(boost::asio::buffer(buffer), receiver_endpoint);

            printLogAtLevel(_DEBUG_ROUGH, "%s, %d, t=%G: delta: message sent
                ↪ via UDP! %s \n", name,myself,t);
        } else {
            printLogAtLevel(_WARNING, "%s, %d, t=%G: delta: message: %s not sent
                ↪ via UDP, because socket not open!\n", name,myself,t,UDP_buffer.str
                ↪ ().c_str());
        }

        // DEVSMessage *in_msg;
        // if(pop_input(0,&in_msg)) // if input message at input port 0
        // phase = newphase
        // sigma = ...
    } else if(!input_bag_empty()) { // if confluent transition
        // DEVSMessage *in_msg;
        // if(pop_input(0,&in_msg)) // if input message at input port 0
        // phase = newphase
        // sigma = ...
    } else { // if pure internal transition
        // phase = newphase
        // sigma = ...
    }
}
void udp_sink::lambda(double t) {
    //This function returns an Event:
    //      Event(%&Value%, %NroPort%)
    //where:
    //      %&Value% points to the variable which contains the value.
    //      %NroPort% is the port number (from 0 to n-1)

    printLogAtLevel(_INFO, "%s, %d, t=%G: lambda \n",name,myself,t);

    /* This function corresponds to the RPDEVS-function lambda and
    is supposed to produce output (or not). It may be called several
    times at same simulation time, before a state transition is
    conducted. Therefore, do not change state variables inside here!
    Calculate output values due to the received input values (x_b) and
    due to the current values of the state variables.
```

```
    Output Messages have to be instances of classes which are derived
    from DEVSMessage, as for example QSSDoubleArray, which is used for
    QSS_signals. Add an output message at output port 'i' by typing for
    example:

        QSSDoubleArray output;

        add_output(&output, i);

    To see how to access input messages, see comments in 'Internal transition '.

     */
    if(t + (tn_t) > t) { // if pure external transition
        // DEVSMessage *in_msg;
        // if(pop_input(0,&in_msg)) // if input message at input port 0
    } else if(!input_bag_empty()) { // if confluent transition
        // DEVSMessage *in_msg;
        // if(pop_input(0,&in_msg)) // if input message at input port 0

    } else { // if pure internal transition

    }

}
void udp_sink::exit() {
    //Code executed at the end of the simulation.
    printLogAtLevel(_INFO, "%s, %d: exit \n", name,myself);
}
```

9.2.3 hbasim.proto

```
// See README.txt for information and build instructions.
//
// Note: START and END tags are used in comments to define sections used in
// tutorials. They are not part of the syntax for Protocol Buffers.
//
// To get an in_depth walkthrough of this file and the related examples, see:
// https://developers.google.com/protocol_buffers/docs/tutorials

// [START declaration]
syntax = "proto2";
```

```
package proto;
// [END declaration]

// [START java_declaration]
option java_package = "at.ac.tuwien.auto.bacnetsrv";
option java_outer_classname = "HBASimProtos";
// [END java_declaration]

// C# is not supported at this time
// [START csharp_declaration]
// option csharp_namespace = "";
// [END csharp_declaration]

// [START messages]

// ctrl command for rpdevs simulator
message CtrlCmd {
    enum CmdType {
        START = 0;
        STOP = 1;
    }

    optional CmdType cmd = 1;
}

// digital message format
message Digital {
    optional int32 id = 1;
    optional bool value = 2;
}

// analog message format
message Analog {
    optional int32 id = 1;
    optional double value = 2;
}

message UDPMSG {
    oneof HMI {
        CtrlCmd ctrl      = 1;
        Digital digMsg    = 2;
        Analog anMsg      = 3;
    }
```

```
}
// [END messages]
```

9.2.4 pdevslib.cpp

```
/*                ***************************************************************************
 **
 **   Copyright (C) 2018 Franz Preyser - Automation Systems Group, TU Wien
 **   Contact: PowerRPDEVS Information (Franz.Preyser@tuwien.ac.at)
 **   Copyright (C) 2009 Facultad de Ciencia Exactas Ingenieria y Agrimensura
 **                 Universidad Nacional de Rosario - Argentina.
 **   Contact: PowerDEVS Information (kofman@fceia.unr.edu.ar,
 **                                   fbergero@fceia.unr.edu.ar)
 **
 **   This file is part of PowerRPDEVS.
 **
 **   PowerRPDEVS is forked from PowerDEVS and is free software: you can
 **   redistribute it and/or modify it under the terms of the GNU General Public
 **   License as published by the Free Software Foundation, either version 3 of
 **   the License, or (at your option) any later version.
 **
 **   PowerRPDEVS is distributed in the hope that it will be useful,
 **   but WITHOUT ANY WARRANTY; without even the implied warranty of
 **   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 **   GNU General Public License for more details.
 **
 **   You should have received a copy of the GNU General Public License
 **   along with PowerRPDEVS. If not, see <http://www.gnu.org/licenses/>.
 **
 *****************************************************************************/

// doxygen - command:
/*! \file
 *  \ingroup AuxiliaryFunctions
 */
#include "pdevslib.h"
#include "root_simulator.h"
#include "root_coupling.h"

bool debug=true;
void DEBUG(DebugEvent de,Simulator *s, const char *fmt, ...)
{
```

```
  if (!debug) return;
  va_list listPtr;
  va_start( listPtr, fmt );
  vprintf( fmt, listPtr );
  va_end( listPtr );
}


RootSimulator* getRootSimulator(Simulator *sim) {
    RootCoupling* rootC;
    Coupling* fath = sim->father;
    while(NULL != fath->father) {
        fath = fath->father;
    }
    rootC = (RootCoupling*)fath;
    return (rootC->rootSim);
}

Time get_tf(Simulator *sim) {
    RootSimulator* rootC = getRootSimulator(sim);
    return(rootC->tf);
}

//extended for the BACnet HIL Simulation bachelor thesis
//@author Stefan Adelmann <e01633044@student.tuwien.ac.at>
#ifdef ___WINDOWS___
  typedef short _stdcall (*inpfuncPtr)(short portaddr);
  typedef void _stdcall (*oupfuncPtr)(short portaddr, short datum);
  #include "pdevslib.win.cpp"
#else
  #ifdef RTAIOS
  #include "pdevslib.rtai.cpp"
#else
  #ifdef ___ARM___
  #include "pdevslib.arm.cpp"
#else
#include "pdevslib.linux.cpp"
#endif
#endif
#endif
```

9.2.5 pdevslib.common.cpp

```
/*          ***************************************************************************
 **
 **   Copyright (C) 2018 Franz Preyser - Automation Systems Group, TU Wien
 **   Contact: PowerRPDEVS Information (Franz.Preyser@tuwien.ac.at)
 **   Copyright (C) 2009 Facultad de Ciencia Exactas Ingenieria y Agrimensura
 **                 Universidad Nacional de Rosario - Argentina.
 **   Contact: PowerDEVS Information (kofman@fceia.unr.edu.ar,
 **                                   fbergero@fceia.unr.edu.ar)
 **
 **   This file is part of PowerRPDEVS.
 **
 **   PowerRPDEVS is forked from PowerDEVS and is free software: you can
 **   redistribute it and/or modify it under the terms of the GNU General Public
 **   License as published by the Free Software Foundation, either version 3 of
 **   the License, or (at your option) any later version.
 **
 **   PowerRPDEVS is distributed in the hope that it will be useful,
 **   but WITHOUT ANY WARRANTY; without even the implied warranty of
 **   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 **   GNU General Public License for more details.
 **
 **   You should have received a copy of the GNU General Public License
 **   along with PowerRPDEVS. If not, see <http://www.gnu.org/licenses/>.
 **
          ***************************************************************************          */

// doxygen - command:
/*! \file
 *   \ingroup AuxiliaryFunctions
 */

#ifdef ___linux___
#define SOCKET_ERROR -1
int SendSocket;
#else
#define H5_SIZEOF_SSIZE_T 1
SOCKET SendSocket;
#endif
struct sockaddr_in service, serviceUDP;
bool activeScilab;

//Edited for the BACnet HIL Simulation bachelor thesis
```

```
//@author Stefan Adelmann <e01633044@student.tuwien.ac.at>
#ifdef __UBUNTU1404__
    #include <hdf5.h> // use this line in kubuntu 14.04
#elif __UBUNTU1804__
    #include <hdf5/serial/hdf5.h> // use this line in mint 19
#elif __UBUNTU1604__
    #include <hdf5/serial/hdf5.h> // use this line in kubuntu 16.04
#elif __ARM__
  //no scilab/matlab integration possible
#else
    #include <hdf5/serial/hdf5.h>
#endif


#include <string>
#include <boost/chrono.hpp>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/shared_ptr.hpp>
#include <mutex>
#include "types.h"

 void UDP_send_handle(const boost::system::error_code& error,
      std::size_t len_ptr)
  {
    printLogAtLevel(_INFO,"thread: UDP_send_handle\n");
  }


void UDP_receive_handle(const boost::system::error_code& error,std::size_t len, std
    ↪ ::size_t* len_ptr){
   printLogAtLevel(_INFO,"thread: UDP_receive_handle\n");
    if (!error || error == boost::asio::error::message_size) {
      *len_ptr = len;
    }
  else
    throw boost::system::system_error(error);

}

/**
@brief Thread handles asynchounus reception of UDP messages for the UDP_Source
    ↪ block
```

```
Thread synchronizes with a potential sender by the means of sending a ready message.
    ↪ After that the thread
is active until a message is received, at this point the execution is stopped and
    ↪ the message stored in the
global buffer.

Edited for the BACnet HIL Simulation bachelor thesis
@author Stefan Adelmann <e01633044@student.tuwien.ac.at>
**/
void UDP_Server_Thread(std::string* msg, unsigned int port, boost::chrono::
    ↪ steady_clock::time_point *t_start_ptr, Time *tn_new, boost::asio::io_service *
    ↪ io_serv_ptr, std::mutex* mtx, bool* udpReceived) {
   printLogAtLevel(_INFO,"thread: enter UDP thread\n");
   try
   {
      using boost::asio::ip::udp;
    udp::socket skt((*io_serv_ptr), udp::endpoint(udp::v4(), port));

    //create a end point for the synchonization
    udp::endpoint receiver_endpoint;
    std::string dest_IP = "127.0.0.1";
    std::string UDP_port = "2013";
    boost::asio::io_service io_serv;

    //try to resolve the server to synchronize with
    try {
      udp::resolver resolver(io_serv);
      udp::resolver::query query(udp::v4(), dest_IP, UDP_port, boost::asio::ip::
          ↪ resolver_query_base::flags());
      receiver_endpoint = *resolver.resolve(query);
    } catch (std::exception& e) {
      printLogAtLevel(_ERROR, "open socked failed: %s\n", e.what());
    }

      udp::endpoint remote_endpoint;
      boost::system::error_code error;

      std::array<char, 256> recv_buf;
      std::size_t len=0;
      printLogAtLevel(_INFO,"thread: started reception\n");

    //create a ready message to send
```

```
        boost::shared_ptr<std::string> message(
                new std::string("ready"));

        //sync send of the ready message to the remote server
        skt.send_to(boost::asio::buffer(*message), receiver_endpoint);

        //start a asynchrounus reception process to receive UDP messages
        skt.async_receive_from(boost::asio::buffer(recv_buf), remote_endpoint, boost::
            ↪ bind(&UDP_receive_handle,boost::asio::placeholders::error,boost::asio::
            ↪ placeholders::bytes_transferred, &len) );

        try {
            printLogAtLevel(_INFO,"thread: io_serv_ptr->run()\n");
            mtx->unlock();
            io_serv_ptr->run();
          //at this point a message was received
            printLogAtLevel(_INFO,"thread: exited io_serv_ptr->run()\n");
          }
          catch (const std::exception& e) {
            std::cerr << "Server: Network exception: "<< e.what()<< std::endl;
          }
          catch (...) {
            std::cerr << "Server: Network exception: unknown"<< std::endl;
          }
          io_serv_ptr->reset();

        //write the message and length to the global buffer
          *msg = std::string(recv_buf.data(),len);
          *tn_new = (boost::chrono::duration_cast<boost::chrono::duration<double> >(
              ↪ boost::chrono::steady_clock::now()-(*t_start_ptr))).count();
          printLogAtLevel(_INFO,"thread: tn=%G\n", *tn_new);
        } catch (std::exception& e)
        {
          std::cerr << e.what() << std::endl;
        }

      //exit the UDP Thread to wait for msg usage
       mtx->unlock();
      *udpReceived = true;
       printLogAtLevel(_INFO,"thread: exit UDP thread\n");
    }

    void throwException(const std::string &error_msg) {
```

```
    MsgException myException(error_msg.c_str());
    printLogAtLevel(_ERROR, error_msg.c_str());
    throw myException;
}

#ifndef ___ARM___
    double executeVoidScilabJob(char *cmd,bool blocking) {
      char buff[1024];
      sprintf(buff,"\\%s",cmd); // If the command starts with a slash the command
          ↪ is not written to the out var
      executeScilabJob(buff,blocking);
      return 0.0;
    }

    double executeScilabJob(char *cmd,bool blocking) {
      double ans=0;
      char buff[1024];
      int result;

      initScilab();
      strcpy(buff,"");
      if (blocking)
        strcpy(buff,"!");
      strcat(buff,cmd);
      strcat(buff,"\n");
      if (!activeScilab) {
        exitStatus = -1;
        printLog("There's not an instance of Scilab running. Returing zero\n");
        return 0.0;
      }
    #ifdef ___linux___
      if (write(SendSocket,buff,strlen(buff))<(signed)strlen(buff))
    #else
      if ((send(SendSocket,buff,strlen(buff),0))<(signed)strlen(buff))
    #endif
        printLog("Incomplete TCP message\n");
      if (blocking) {
    #ifdef ___linux___
          result = read(SendSocket, buff , 1024);
    #else
          result = recv(SendSocket, buff , 1024,0);
    #endif
      }
```

```
  return ans;
}

void initScilab() {
  static int init=0;
  if (init++ == 0) {
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = inet_addr("127.0.0.1");
#ifdef ___linux___
    int actual_port = 27020+(getuid() % 10000);
#else
    int actual_port = 27020;
    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
#endif
    SendSocket = socket(AF_INET, SOCK_STREAM,IPPROTO_TCP);
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = inet_addr("127.0.0.1");
    service.sin_port = htons(actual_port);
    printLog("Openening connection with scilab to TPC port %d\n",actual_port);
    //ioctlsocket(SendSocket,FIONBIO,&bio);
      if (connect(SendSocket,(struct sockaddr*)&service,sizeof(service)) ==
        ↪ SOCKET_ERROR)
       {
        activeScilab=false;
        exitStatus = -1;
     printLog("Scilab not found\n");
        return;
       }
       activeScilab=true;
  }
}
void cleanScilab() {
 static int clean=0;
 if (clean++ == 0) {
#ifdef ___linux___
    close(SendSocket);
#else
    closesocket(SendSocket);
#endif
 }
}
```

```
void putScilabVar(char *varname, double var) {
  initScilab();
   char buf[1024];
   sprintf(buf,"%s=%g",varname,var);
   executeVoidScilabJob(buf,false);
}

double getScilabVar(char *varname) {
  // Try to solve it locally
  char *s;
  double d=strtod(varname,&s);
  if (varname+strlen(varname)==s)
    return d;
  initScilab();
  char buf[1024];
  double f;
  sprintf(buf,"anss=%s",varname);
  executeVoidScilabJob(buf,true);
  executeScilabJob((char*)"exists('anss')",false);
  getAns(&f,1,1);
  if (!activeScilab) {
    exitStatus = -1;
    printLog("There's not an instance of Scilab running. Returning atof(%s)\n",
        ↪ varname);
    return atof(varname);
  }
  if (f==0) {
    exitStatus = -1;
    printLog("Variable %s does not exists! Returning zero\n",varname);
    return 0.0;
  }
  executeScilabJob((char*)"anss",true);
  getAns(&f,1,1);
  executeVoidScilabJob((char*)"clear anss",true);
  return f;
}


void getAns(double *ans, int rows, int cols) {
  initScilab();
  if (!activeScilab) {
    exitStatus = -1;
```

```
    printLog("There's not an instance of Scilab running. Returing zero\n");
    ans[0]=0.0;
  }
  char cmd[124] = "@";
  int iResult;
  executeScilabJob(cmd,false);
  ans[0]=0;
  if (!activeScilab)
    return;
#ifdef ___linux___
  iResult = read(SendSocket, (char*)ans, sizeof(double)*rows*cols);
#else
  iResult = recv(SendSocket, (char*)ans, sizeof(double)*rows*cols,0);
#endif
  if (iResult<0) {
    exitStatus = -1;
    printLog("There's not an instance of Scilab running. Returing zero\n");
    activeScilab=false;
  }
}


void getScilabMatrix(char* varname, int *rows, int *cols, double **data) {
  char buf[1024];
  hid_t file_id, dataset_id;
  sprintf(buf,"tempvar=%s",varname);
  executeVoidScilabJob(buf,true);
  sprintf(buf,"save('%s/../output/temp.dat','tempvar')",getenv("PWD"));
  executeVoidScilabJob(buf,true);
  file_id = H5Fopen("temp.dat",H5F_ACC_RDONLY, H5P_DEFAULT);
  dataset_id = H5Dopen2(file_id, "/tempvar", H5P_DEFAULT);
  int dimension = H5Sget_simple_extent_ndims(H5Dget_space(dataset_id));
  if (dimension!=2) {
    printLog("Incomplete read in getScilabMatrix\n");
    return;
  }
  hsize_t size[2];
  H5Sget_simple_extent_dims(H5Dget_space(dataset_id), size, NULL);
  rows[0]=(int)size[0];
  cols[0]=(int)size[1];
  H5Dread(dataset_id, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL,
      ↪ H5P_DEFAULT, data);
  H5Fclose(file_id);
```

```
  H5Dclose(dataset_id);
  unlink("temp.dat");
}

void getScilabVector(char* varname, int *length, double *data) {
  char *s;
  double d=strtod(varname,&s);
  hid_t       file_id, dataset_id;
  hsize_t size[2];
  if (varname+strlen(varname)==s)
  {
    *length=1;
    data[0]=d;
    return;
  }
  char buf[1024];
  sprintf(buf,"tempvar=%s",varname);
  executeVoidScilabJob(buf,true);
  sprintf(buf,"save('temp.dat','tempvar')");
  executeVoidScilabJob(buf,true);
  sprintf(buf,"temp.dat");
  file_id = H5Fopen(buf,H5F_ACC_RDONLY, H5P_DEFAULT);
  dataset_id = H5Dopen2(file_id, "/tempvar", H5P_DEFAULT);
  int dimension = H5Sget_simple_extent_ndims(H5Dget_space(dataset_id));
  H5Sget_simple_extent_dims(H5Dget_space(dataset_id), size, NULL);
  int rows=(int)size[0];
  int cols=(int)size[1];
  if (dimension!=2 || rows!=1) {
    printLog("Incomplete read in getScilabVector\n");
    return;
  }
  *length=cols;
  H5Dread(dataset_id, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL,
      ↪ H5P_DEFAULT, data);
  H5Dclose(dataset_id);
  H5Fclose(file_id);
  unlink("temp.dat");

  /*sprintf(buf," deletefile ('C:\\powerdevs\\output\\temp.dat')");
  printLog(buf);
  executeVoidScilabJob(buf,true);
  */
}
```

```
#endif

extern double tf;
double getFinalTime()
{
   return tf;
}
```

9.2.6 pdevslib_arm.cpp

```
/*                ***************************************************************************
 **
 **   Copyright (C) 2018 Franz Preyser - Automation Systems Group, TU Wien
 **   Contact: PowerRPDEVS Information (Franz.Preyser@tuwien.ac.at)
 **   Copyright (C) 2009 Facultad de Ciencia Exactas Ingenieria y Agrimensura
 **                 Universidad Nacional de Rosario - Argentina.
 **   Contact: PowerDEVS Information (kofman@fceia.unr.edu.ar,
 **                                   fbergero@fceia.unr.edu.ar)
 **
 **   This file is part of PowerRPDEVS.
 **
 **   PowerRPDEVS is forked from PowerDEVS and is free software: you can
 **   redistribute it and/or modify it under the terms of the GNU General Public
 **   License as published by the Free Software Foundation, either version 3 of
 **   the License, or (at your option) any later version.
 **
 **   PowerRPDEVS is distributed in the hope that it will be useful,
 **   but WITHOUT ANY WARRANTY; without even the implied warranty of
 **   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 **   GNU General Public License for more details.
 **
 **   You should have received a copy of the GNU General Public License
 **   along with PowerRPDEVS. If not, see <http://www.gnu.org/licenses/>.
 **
 **   Edited for the BACnet HIL Simulation bachelor thesis
 **   Removed all non ARM compatiple function
 **   @author Stefan Adelmann <e01633044@student.tuwien.ac.at>
 *****************************************************************************          */

// doxygen - command:
/*! \file
 *  \ingroup AuxiliaryFunctions
```

```
 */

#include <sys/poll.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include "types.h"
#include <sys/time.h>
#include <math.h>
#include <sys/io.h>
#include <root_simulator.h>

OS getOs()
{
   return LINUX;
}

double getTime()
{
   return 1.0*clock()/CLOCKS_PER_SEC;
}


double getRealSimulationTime()
{
  struct timeval tv;
  gettimeofday(&tv,NULL);
  return (tv.tv_sec + tv.tv_usec*1.0e-6) - realTiSimulation;
}

int waitFor(Time t, RealTimeMode m)
{
   double ti=getRealSimulationTime();
   if (t<=0)
```

```
    return 0;
   while ((getRealSimulationTime()-ti)<t) ;
  return 0;
}

void initLib()
{
  struct timeval tv;
  gettimeofday(&tv,NULL);
  realTiSimulation = tv.tv_sec + tv.tv_usec*1.0e-6;
}

void cleanLib()
{

}

void parseCommandLine(char *program, char* cmdLineTxt, char*** argv, int* argc){
    int count = 1;

    char *cmdLineCopy = strdupa(cmdLineTxt);
    char* match = strtok(cmdLineCopy, " ");
// First, count the number of arguments
    while(match != NULL){
        count++;
        match = strtok(NULL, " ");
    }

    *argv = (char**) malloc(sizeof(char*) * (count+1));
    (*argv)[count] = 0;
    **argv = strdup(program); // The program name would normally go in here

    if (count > 1){
        int i=1;
        cmdLineCopy = strdupa(cmdLineTxt);
        match = strtok(cmdLineCopy, " ");
        do{
            (*argv)[i++] = strdup(match);
            match = strtok(NULL, " ");
        } while(match != NULL);
    }

    *argc = count;
```

```
}

std::vector<pid_t> spawned_children;

void spawnProcess(const char *path, char *arg) {
    char **argv;
    int argc;
    parseCommandLine((char*)path,arg,&argv,&argc);

    pid_t p = fork();
    assert(p != -1);

    if(p == 0) { // Child process
        /*
        printLog("Running program %s with args:", path);
        for (int i=0;i<argc;i++)
            printLog("%d=%s, ",i,argv[i]);
        printLog("\n");
        */
        execv(argv[0],argv);
        throw std::runtime_error("ERROR: "+std::string(argv[0])+" not found!\n");
    }else{
        // FIXME: memory leak in argv
        spawned_children.push_back(p);
    }
}

void cleanupSpawnedProcesses(void){
    for(auto& pid : spawned_children){
        int status;
        if(waitpid(pid, &status, WNOHANG) == pid){
            pid = -1;
        }
    }
    for(auto& pid : spawned_children){
        int status;
        if(pid != -1){
            kill(pid, SIGTERM);
            waitpid(pid, &status, 0);
            pid = -1;
        }
    }
}
```

```
long int PDFileOpen(char* name,char mode) {
    char strMode[2];
    strMode[0]=mode;
    strMode[1]='\0';
    return (long int)fopen(name,strMode);
};
long int PDFileWrite(long int file ,char* buf,int size) {
    long int r=fwrite(buf,1,size,(FILE*)file);
   fflush((FILE*)file);
  return r;
};
long int PDFileRead(long int file ,char* buf ,int size){
  int r;
    if ((r=fread(buf,1,size,(FILE*)file))<size)
     printLog("Incomplete read from file\n");
  return r;
}
void PDFileClose(long int file ){
    fclose((FILE*)file);
};

void printLog(const char *fmt,...) {
  static int init=0;
  if (!init) {
    init=1;
    fclose(fopen("pdevs.log","w"));
  }
    va_list va;
    va_start(va,fmt);
    FILE *fd=fopen("pdevs.log","a");
    vfprintf(fd,fmt,va);
    fclose(fd);
    va_end(va);
}

void RequestIRQ(unsigned irq, void *a){

}

#include "pdevslib.common.cpp"
```

9.2.7 Makefile.include

```
UNAME = $(shell uname)
ifeq ($(UNAME), Windows)
OPTFLAGS = -O3
CXXFLAGS = -Wall $(OPTFLAGS) -std=c++11 -D___WINDOWS___
ATOMICS = ..\atomics
SRCENGINE = ..\engine
BUILD = ..\build
BUILDOBJ = ..\build\objs
BUILDLIB = ..\build\lib
OUTPUT = ..\output
MODEL = $(OUTPUT)\model.exe
RM = del
MV = copy
OBJS = $(BUILDOBJ)\*.o
LIBS = -lsimpd -lws2_32 -lhdf5 -lpthread -lboost_system -lboost_thread -
    ↪ lboost_chrono
INCLUDES = -I$(SRCENGINE) -I$(SRCENGINE)\protobuf -I$(ATOMICS) -I$(
    ↪ BUILD) -L$(BUILDLIB) -I$(SRCENGINE)/hdf5/include -L$(SRCENGINE)/
    ↪ hdf5/bin
all : $(BUILDLIB)/libsimpd.a $(MODEL)
clean:
$(RM) /q /s $(OBJS) $(MODEL)
endif
ifeq ($(UNAME), Linux)
OSVERSION = $(shell lsb_release -sr)
ifeq ($(COMPILER), g++)
OPTFLAGS = -march=native -O3
else
OSVERSIONFLAG = -D___ARM___
OPTFLAGS = -march=armv7 -O3
endif
ifeq ($(OSVERSION), 16.04)
OSVERSIONFLAG = -D___UBUNTU1604___
endif
ifeq ($(OSVERSION), 14.04)
OSVERSIONFLAG = -D___UBUNTU1404___
endif
CXXFLAGS = -Wall $(OPTFLAGS) -std=c++11 $(OSVERSIONFLAG)
ATOMICS = ../atomics
SRCENGINE = ../engine
BUILD = ../build
BUILDOBJ = ../build/objs
```

```
BUILDLIB = ../build/lib
OUTPUT = ../output
ifeq ($(COMPILER), g++)
LIBS = -L/usr/lib/x86_64-linux-gnu/hdf5/serial/ -lsimpd -lhdf5 -lpthread -
    ↪ lboost_system -lboost_thread -lboost_chrono -lprotobuf
else
LIBS = -L$(BUILD)/ARM -lsimpd -lpthread -lboost_system -lboost_thread -
    ↪ lboost_chrono -lprotobuf
endif
RM = rm
MV = mv
OBJS = $(BUILDOBJ)/*.o
MODEL = $(OUTPUT)/model
INCLUDES = -I$(SRCENGINE) -I$(SRCENGINE)/protobuf -I$(ATOMICS) -I$(
    ↪ BUILD) -L$(BUILDLIB)
all :
    if test -f $(OUTPUT)/debug; then make --makefile=Makefile meisterproper;fi
    make --makefile=Makefile all2

all2 : $(BUILDLIB)/libsimpd.a $(MODEL)

meisterproper:
    echo MeisterProper!
    $(RM) -rf $(OBJS) $(BUILDLIB)/libsimpd.a $(MODEL) $(OUTPUT)/debug
    make --directory=$(SRCENGINE)/protobuf clean

clean:
    $(RM) -rf $(OBJS) $(MODEL) $(OUTPUT)/debug

SIMPD=/usr/include/simpd
install : $(BUILDLIB)/libsimpd.a
    -mkdir $(SIMPD)
    install -m 644 -p $(BUILDLIB)/libsimpd.a /usr/lib
    install -m 644 -p ../engine/protobuf/hbasim.pb.h $(SIMPD)
    install -m 644 -p ../engine/QSSDoubleArray.h $(SIMPD)
    install -m 644 -p ../engine/connection.h $(SIMPD)
    install -m 644 -p ../engine/coupling.h $(SIMPD)
    install -m 644 -p ../engine/engine.h $(SIMPD)
    install -m 644 -p ../atomics/bond/bond_connection.h $(SIMPD)
    install -m 644 -p ../engine/types.h $(SIMPD)
    install -m 644 -p ../engine/pdevslib.h $(SIMPD)
endif
```

```
engine_header: $(SRCENGINE)/connection.h $(SRCENGINE)/coupling.h $(
    ↪ SRCENGINE)/DEVSMessage.h $(SRCENGINE)/engine.h $(SRCENGINE)/
    ↪ Entity.h $(SRCENGINE)/pdevslib.h $(SRCENGINE)/QSSDoubleArray.h $(
    ↪ SRCENGINE)/root_coupling.h $(SRCENGINE)/root_simulator.h $(
    ↪ SRCENGINE)/simulator.h $(SRCENGINE)/types.h

$(BUILDLIB)/libsimpd.a: $(BUILDOBJ)/connection.o $(BUILDOBJ)/coupling.o $(
    ↪ SRCENGINE)/DEVSMessage.h $(SRCENGINE)/engine.h $(SRCENGINE)/
    ↪ Entity.h $(BUILDOBJ)/pdevslib.o $(BUILDOBJ)/QSSDoubleArray.o $(
    ↪ BUILDOBJ)/bond_connection.o $(BUILDOBJ)/root_coupling.o $(
    ↪ BUILDOBJ)/root_simulator.o $(BUILDOBJ)/simulator.o $(SRCENGINE)/
    ↪ types.h $(BUILDOBJ)/hbasim.o
  $(AR) rvs $(BUILDLIB)/libsimpd.a $(BUILDOBJ)/connection.o $(BUILDOBJ)/
      ↪ coupling.o $(BUILDOBJ)/pdevslib.o $(BUILDOBJ)/QSSDoubleArray.o $(
      ↪ BUILDOBJ)/bond_connection.o $(BUILDOBJ)/root_coupling.o $(
      ↪ BUILDOBJ)/root_simulator.o $(BUILDOBJ)/simulator.o $(BUILDOBJ)/
      ↪ hbasim.o

$(BUILDOBJ)/connection.o: $(SRCENGINE)/connection.cpp $(SRCENGINE)/
    ↪ connection.h $(SRCENGINE)/types.h
  $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/connection.cpp
      ↪  -o $(BUILDOBJ)/connection.o

$(BUILDOBJ)/coupling.o: $(SRCENGINE)/coupling.cpp $(SRCENGINE)/coupling.h
    ↪  $(SRCENGINE)/simulator.h $(SRCENGINE)/connection.h $(SRCENGINE)/
    ↪ pdevslib.h
  $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/coupling.cpp -
      ↪ o $(BUILDOBJ)/coupling.o

$(BUILDOBJ)/pdevslib.o: $(SRCENGINE)/pdevslib.h $(SRCENGINE)/pdevslib.cpp
    ↪ $(SRCENGINE)/pdevslib.linux.cpp $(SRCENGINE)/pdevslib.win.cpp $(
    ↪ SRCENGINE)/pdevslib.rtai.cpp $(SRCENGINE)/pdevslib.common.cpp $(
    ↪ SRCENGINE)/types.h $(SRCENGINE)/root_simulator.h
  $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/pdevslib.cpp -
      ↪ o $(BUILDOBJ)/pdevslib.o

$(BUILDOBJ)/QSSDoubleArray.o: $(SRCENGINE)/QSSDoubleArray.cpp $(
    ↪ SRCENGINE)/QSSDoubleArray.h $(SRCENGINE)/types.h $(SRCENGINE)/
    ↪ DEVSMessage.h
  $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/
      ↪ QSSDoubleArray.cpp -o $(BUILDOBJ)/QSSDoubleArray.o
```

```
$(BUILDOBJ)/bond_connection.o: $(SRCENGINE)/../atomics/bond/
    ↪ bond_connection.cpp $(SRCENGINE)/../atomics/bond/bond_connection.h
   $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/../atomics/
       ↪ bond/bond_connection.cpp -o $(BUILDOBJ)/bond_connection.o

$(BUILDOBJ)/root_coupling.o: $(SRCENGINE)/root_coupling.cpp $(SRCENGINE)
    ↪ /root_coupling.h $(SRCENGINE)/coupling.h $(SRCENGINE)/root_simulator.
    ↪ h $(SRCENGINE)/connection.h $(SRCENGINE)/pdevslib.h
   $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/root_coupling.
       ↪ cpp -o $(BUILDOBJ)/root_coupling.o

$(BUILDOBJ)/root_simulator.o: $(SRCENGINE)/root_simulator.cpp $(
    ↪ SRCENGINE)/root_simulator.h $(SRCENGINE)/root_coupling.h $(
    ↪ SRCENGINE)/types.h $(SRCENGINE)/pdevslib.h $(SRCENGINE)/coupling.h
   $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/root_simulator
       ↪ .cpp -o $(BUILDOBJ)/root_simulator.o

$(BUILDOBJ)/simulator.o: $(SRCENGINE)/simulator.cpp $(SRCENGINE)/
    ↪ simulator.h $(SRCENGINE)/types.h $(SRCENGINE)/pdevslib.h $(
    ↪ SRCENGINE)/coupling.h
   $(COMPILER) $(CXXFLAGS) $(INCLUDES) -c $(SRCENGINE)/simulator.cpp
       ↪ -o $(BUILDOBJ)/simulator.o

$(BUILDOBJ)/hbasim.o: $(SRCENGINE)/protobuf/hbasim.pb.cc
   pkg-config --cflags  protobuf
   $(COMPILER) $(CXXFLAGS) -c $(SRCENGINE)/protobuf/hbasim.pb.cc -o $(
       ↪ BUILDOBJ)/hbasim.o `pkg-config --cflags --libs protobuf`

$(SRCENGINE)/protobuf/hbasim.pb.cc: $(SRCENGINE)/protobuf/hbasim.proto
   protoc --cpp_out=$(SRCENGINE)/protobuf/ --proto_path=$(SRCENGINE)/
       ↪ protobuf/ $(SRCENGINE)/protobuf/hbasim.proto
   @touch $(SRCENGINE)/protobuf/protoc

-include $(BUILD)/Makefile.include
```

9.2.8 pdppt

main.cpp

```
/*            ***************************************************************************
**
**  Copyright (C) 2009 Facultad de Ciencia Exactas Ingenieria y Agrimensura
**  Universidad Nacional de Rosario - Argentina.
```

```
**  Contact: PowerDEVS Information (kofman@fceia.unr.edu.ar, fbergero@fceia.unr.
    ↪ edu.ar)
**
**  This file is part of PowerDEVS.
**
**  PowerDEVS is free software: you can redistribute it and/or modify
**  it under the terms of the GNU General Public License as published by
**  the Free Software Foundation, either version 3 of the License, or
**  (at your option) any later version.
**
**  PowerDEVS is distributed in the hope that it will be useful,
**  but WITHOUT ANY WARRANTY; without even the implied warranty of
**  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
**  GNU General Public License for more details.
**
**  You should have received a copy of the GNU General Public License
**  along with PowerDEVS. If not, see <http://www.gnu.org/licenses/>.
**
        ************************************************************************ */

#include <stdio.h>
#include <stdlib.h>
#include <QProcess>
#include <QThread>
#include <QFile>
#include <QDir>
#include <QMessageBox>
#include <QDebug>
#include <QPushButton>

#include "parser.h"
#include "modelcoupled.h"
#include "codegenerator.h"

void showLog();
int main(int argc, char **argv)
{

    QApplication app(argc, argv);
  QStringList args=QCoreApplication::arguments();
    if (args.size()==1) {
        printf("Usage: pdppt [options] file \n");
        printf(" Options = \n");
```

```
        printf("                -r Only generate structure file \n");
        printf("                -m Generate structure and generate code\n");
        printf("                -x Generate structure, code and run\n");
        printf("                -s Generate structure, code and run silent\n");
        printf("                -f file Use the supplied Makefile\n");
      printf("                -d debug mode, don't use multiple cores\n");
      printf("               -arm cross-compile model for ARM cores");
        printf("                -pdif binary Use the supplied interface for running\n");
        printf("  file can be a model (pdm) or a model structure(pds)\n");
       return 0;
    }
    // Parse command line args
    QString filename;
    QString makefilename("");
    QString pdif_command("../bin/pdif");
    bool generateCCode = true;
    bool silent = false;
    bool runSimulation = false;
   bool debug = false;
   bool crossCompile = false;

   for (int i=1;i<args.size();i++)
   {
        if (args.at(i) == "-x")
            runSimulation = true;
        else if (args.at(i) == "-m")
            runSimulation = false;
        else if (args.at(i)=="-arm")
          crossCompile = true;
        else if (args.at(i) == "-s") {
             silent = true;
            runSimulation = true;
        } else if (args.at(i) == "-d") {
            debug = true;
        } else if (args.at(i)=="-f") {
        if (i+1<args.size())
            makefilename=args.at(++i);
        } else if (args.at(i)=="-pdif") {
        if (i+1<args.size())
            pdif_command=args.at(++i);
        } else filename = args.at(i);
   }
    // Hack to run under wine
```

```
 if (filename.contains(":\\powerdevs")) {
    filename = filename.mid(filename.indexOf("powerdevs") + 9).replace("\\", "/");
    filename = ".." + filename;
 }
 modelCoupled *c = NULL;
 if (filename.endsWith(".pdm", Qt::CaseInsensitive)) {
    c = parseModel(filename);
    generateCode(c, filename.replace(".pdm", ".pds", Qt::CaseInsensitive),
        ↪ generateCCode, true);
 } else if (filename.endsWith(".pds", Qt::CaseInsensitive)) {
    c = parsePDS(filename);
    generateCode(c, filename.replace(".pdm", ".pds", Qt::CaseInsensitive),
        ↪ generateCCode, false);
 } else {
    printf("File must be either a .pdm or .pds\n");
    return -1;
 }
QString path = QCoreApplication::applicationDirPath();
qDebug()<< "ApplicationDirPath before setCurrent: " << path;
QDir buildDir (path + "/../build");
QDir binDir (path + "/../bin");
QString absBuildPath = buildDir.absolutePath();
QString absBinPath = binDir.absolutePath();
QDir::setCurrent(absBuildPath);
path = QCoreApplication::applicationDirPath();
qDebug() << "absolute path after setCurrent: " << path;
QProcess make;
//make.setProcessChannelMode(QProcess::ForwardedChannels);
make.setWorkingDirectory(absBuildPath);
QStringList argsmake;
argsmake << "--directory="+absBuildPath;
argsmake << "--print-directory";
argsmake << "BUILD_DIR="+absBuildPath;
argsmake << "BIN_DIR="+absBinPath;

if(crossCompile)
  argsmake << "COMPILER=arm-linux-gnueabihf-g++";
else
  argsmake << "COMPILER=g++";

if (makefilename != "") {
  argsmake << "-f";
```

```
    argsmake << makefilename;
  }
  if( false ==debug) {
    auto cores = QThread::idealThreadCount();
    if(cores > 1){
        argsmake << "-j";
        argsmake << QString::number(cores);
        qDebug()<<"Using " << cores << "cores\n";
    }
  }

  qDebug()<< "Calling make with " << argsmake;
#ifdef Q_OS_LINUX
  qDebug()<< "make working directory:" << make.workingDirectory();
  make.start("/usr/bin/make",argsmake);
#else
  QStringList env = QProcess::systemEnvironment();
  env << "PATH=" + path + "/gcc/bin" ;
  make.setEnvironment(env);
  make.start("../bin/gcc/bin/make.exe",argsmake);
#endif
  if( false == make.waitForStarted()) {
      qDebug()<< "ERROR! - starting make failed! ";
      qDebug()<< make.errorString();
    }
  make.waitForFinished(-1);
  qDebug()<< make.readAll();

    QByteArray log(make.readAllStandardError());
      if (make.exitCode() == 0) {
       if (runSimulation) {
          QDir::setCurrent(path + "/../output");
          QProcess pdif;
#ifdef RTAIOS
          pdif.startDetached("/usr/bin/kdesudo", QStringList() << pdif_command
              ↪ << filename.left(filename. lastIndexOf(".")) + ".stm");
#else
          if (! silent ) {
#ifdef Q_OS_LINUX
          bool pdifRunning = system("ps -e | grep pdif") == 0;
          qDebug() << "PDIF running = " << pdifRunning;
          if (pdifRunning) {
            system(" killall  pdif");
```

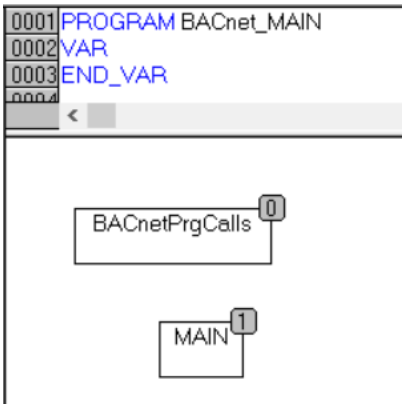
```
        }
#endif
        if (pdif_command != "null")
            pdif.startDetached(pdif_command, QStringList() << filename.left(
                ↪ filename.lastIndexOf(".")) + ".stm");
         } else {
            QFile stm(filename.left(filename.lastIndexOf(".")) + ".stm");
            //qDebug() << (filename.left(filename.lastIndexOf(".")) + ".stm");
            double tf, ti;
            bool ok=false;
            if (stm.open(QIODevice::ReadOnly | QIODevice::Text)) {
               QString s(stm.readAll());
               QStringList ls = s.split('\n');
               ti = ls.first().trimmed().toDouble();
               tf = ls.at(1).trimmed().toDouble(&ok);
            }
            if (ok)
             pdif.startDetached("./model", QStringList() << "-tf" << QString("%1
                 ↪ ").arg(tf));
            else
               pdif.startDetached(pdif_command, QStringList() << filename.left(
                   ↪ filename.lastIndexOf(".")) + ".stm");
        }
        } else if (crossCompile)
    {
        QMessageBox msgBox;
        msgBox.setWindowState(Qt::WindowNoState);

        //msgBox.setDetailedText(QString:: fromUtf8(log));
        msgBox.addButton(QMessageBox::Ok);
        msgBox.setText ("Success: Cross-Compilation finished");
        msgBox.setIcon(QMessageBox::Information);
        msgBox.setWindowTitle("PowerDEVS");
        msgBox.show();
        msgBox.showNormal();
        msgBox.exec();
    }
#endif
    } else {
        QMessageBox msgBox;
        msgBox.setWindowState(Qt::WindowNoState);

        //msgBox.setDetailedText(QString:: fromUtf8(log));
```

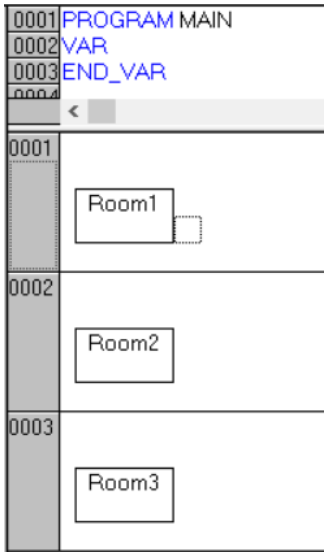
```
        QPushButton viewLog("View Compilation Log");

        msgBox.addButton(&viewLog,QMessageBox::HelpRole);
        msgBox.addButton(QMessageBox::Ok);
            msgBox.setText ("Error: The compilation process has reported an error.");
            msgBox.setIcon(QMessageBox::Critical);
            msgBox.setWindowTitle("PowerDEVS");
            msgBox.show();
            msgBox.showNormal();

        QFile logfile (path + "/../output/compile.log");
         logfile .open(QIODevice::WriteOnly);
         logfile .write(log);
         logfile .close();
        msgBox.exec();
            if (msgBox.clickedButton()==&viewLog) {
#ifndef Q_OS_WIN32
          QProcess::startDetached("/usr/bin/xdg-open", QStringList() << "../output/
              ↪ compile.log");
#else
          QProcess::startDetached("notepad", QStringList() << "../output/compile.log")
              ↪ ;
#endif
        }
            return -1;
        }
     QFile logfile (path + "/../output/compile.log");
      logfile .open(QIODevice::WriteOnly);
      logfile .write(log);
      logfile .close();
    return 0;
}

void showLog() {

}
```

9.3 PLC

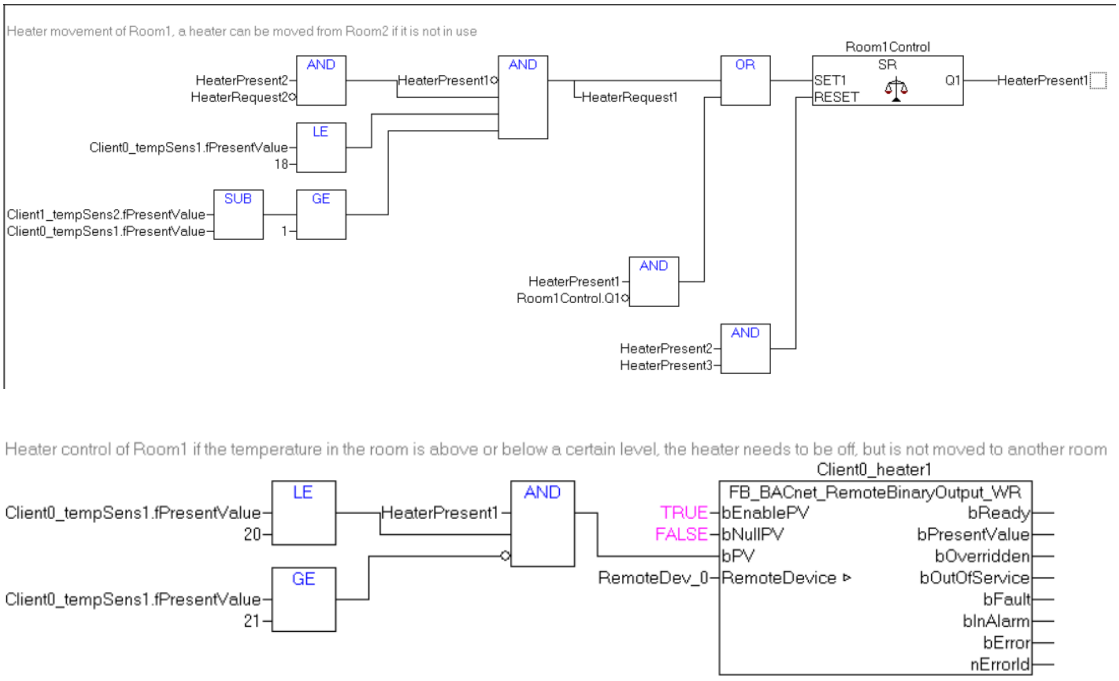
9.3.1 BACNET_MAIN

9.3.2 MAIN

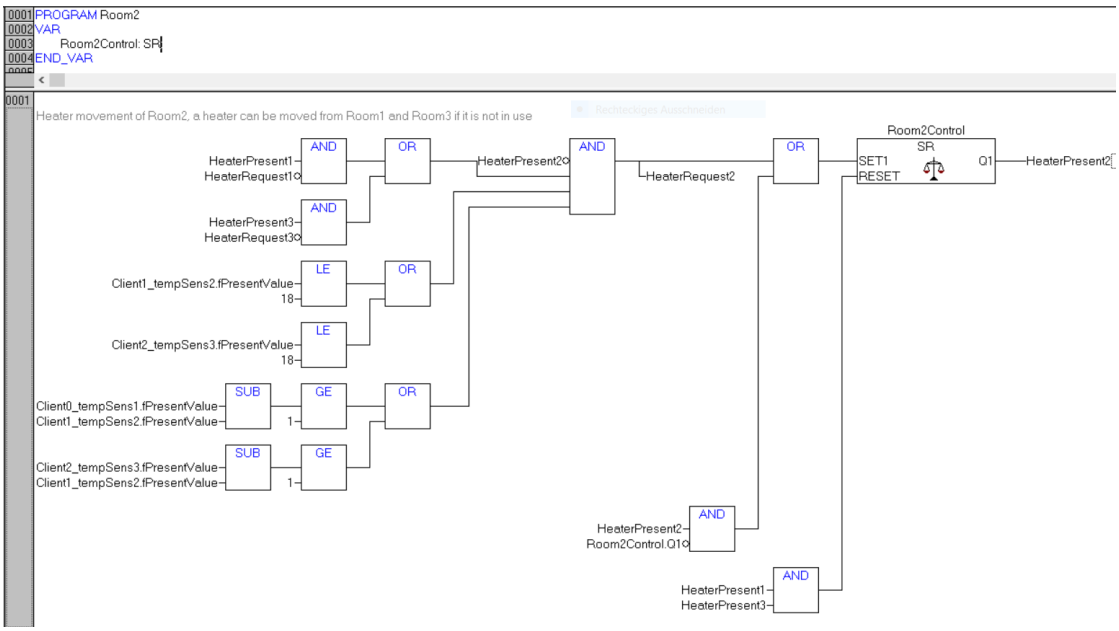

0001 PROGRAM BACnet_MAIN
0002 VAR
0003 END_VAR
BACnetPrgCalls
0
MAIN
1

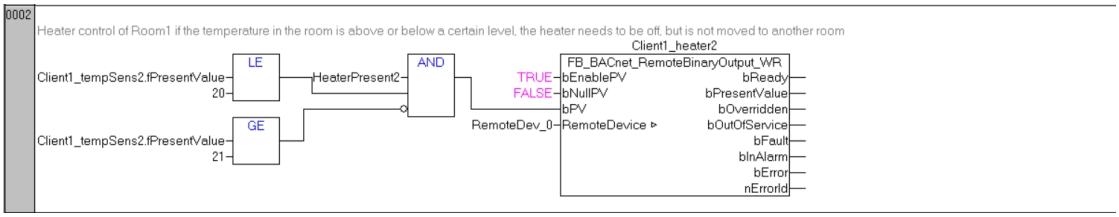

0001 PROGRAM MAIN
0002 VAR
0003 END_VAR
0001
Room1
0002
Room2
0003
Room3

9.3.3 Room1

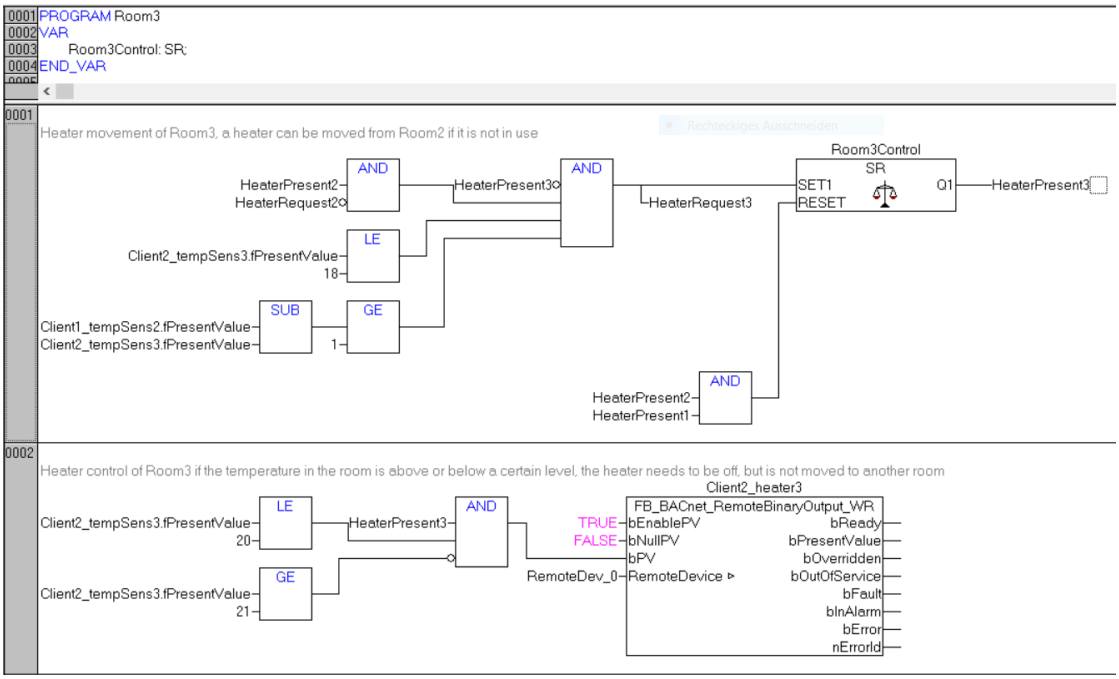

Heater movement of Room1, a heater can be moved from Room2 if it is not in use
HeaterPresent2
HeaterRequest2
AND
HeaterPresent1
AND
HeaterRequest1
OR
Room1Control
SR
SET1
RESET
Q1
HeaterPresent1
LE
Client0_tempSens1.fPresentValue
18
SUB
GE
Client1_tempSens2.fPresentValue
Client0_tempSens1.fPresentValue
1
AND
HeaterPresent1
Room1Control.Q1
AND
HeaterPresent2
HeaterPresent3
Heater control of Room1 if the temperature in the room is above or below a certain level, the heater needs to be off, but is not moved to another room
Client0_tempSens1.fPresentValue
20
LE
HeaterPresent1
AND
Client0_tempSens1.fPresentValue
21
GE
Client0_heater1
FB_BACnet_RemoteBinaryOutput_WR
TRUE
bEnablePV
bReady
FALSE
bNullPV
bPresentValue
bPV
bOverridden
RemoteDev_0
RemoteDevice
bOutOfService
bFault
bInAlarm
bError
nErrorId

9.3.4 Room2


0001 PROGRAM Room2
0002 VAR
0003 Room2Control: SR
0004 END_VAR
0001
Heater movement of Room2, a heater can be moved from Room1 and Room3 if it is not in use
HeaterPresent1
HeaterRequest1
AND
OR
HeaterPresent2
AND
HeaterRequest2
OR
Room2Control
SR
SET1
RESET
Q1
HeaterPresent2
AND
HeaterPresent3
HeaterRequest3
LE
Client1_tempSens2.fPresentValue
18
OR
LE
Client2_tempSens3.fPresentValue
18
SUB
GE
OR
Client0_tempSens1.fPresentValue
Client1_tempSens2.fPresentValue
1
SUB
GE
Client2_tempSens3.fPresentValue
Client1_tempSens2.fPresentValue
1
AND
HeaterPresent2
Room2Control.Q1
AND
HeaterPresent1
HeaterPresent3


0002
Heater control of Room1 if the temperature in the room is above or below a certain level, the heater needs to be off, but is not moved to another room
Client1_heater2
Client1_tempSens2.fPresentValue
20
LE
HeaterPresent2
AND
Client1_tempSens2.fPresentValue
21
GE
TRUE
FALSE
RemoteDev_0
FB_BACnet_RemoteBinaryOutput_WR
bEnablePV
bNullPV
bPV
RemoteDevice
bReady
bPresentValue
bOverridden
bOutOfService
bFault
bInAlarm
bError
nErrorId

9.3.5 Room3


0001 PROGRAM Room3
0002 VAR
0003 Room3Control: SR;
0004 END_VAR
0001
Heater movement of Room3, a heater can be moved from Room2 if it is not in use
HeaterPresent2
HeaterRequest2
AND
HeaterPresent3
AND
HeaterRequest3
Room3Control
SR
SET1
RESET
Q1
HeaterPresent3
Client2_tempSens3.fPresentValue
18
LE
Client1_tempSens2.fPresentValue
Client2_tempSens3.fPresentValue
SUB
1
GE
HeaterPresent2
HeaterPresent1
AND
0002
Heater control of Room3 if the temperature in the room is above or below a certain level, the heater needs to be off, but is not moved to another room
Client2_heater3
Client2_tempSens3.fPresentValue
20
LE
HeaterPresent3
AND
Client2_tempSens3.fPresentValue
21
GE
TRUE
FALSE
RemoteDev_0
FB_BACnet_RemoteBinaryOutput_WR
bEnablePV
bNullPV
bPV
RemoteDevice
bReady
bPresentValue
bOverridden
bOutOfService
bFault
bInAlarm
bError
nErrorId

List of Figures

List of Tables

Bibliography

[Aug04] Godfried Augenbroe. Trends in building simulation. In *Advanced building simulation*, pages 18–38. Routledge, 2004.

[BAC] BAC. `https://github.com/infiniteautomation/BACnet4J`. [Online; accessed 5-July-2019].

[BBB] BBB. `https://beagleboard.org/black`. [Online; accessed 5-July-2019].

[BEA] BEAGLE. `https://beagleboard.org/latest-images`. [Online; accessed 5-July-2019].

[BEC] BECKHOFF. `https://www.beckhoff.com/default.asp?twincat/lizensi.htm?id=159876032042887`. [Online; accessed 5-July-2019].

[BOO] BOOST. `https://www.boost.org/doc/libs/1_66_0/doc/html/boost_asio.html`. [Online; accessed 5-July-2019].

[CX] CX. `https://www.beckhoff.de/CX5010/`. [Online; accessed 5-July-2019].

[ETC] ETCHER. `https://www.balena.io/etcher/`. [Online; accessed 5-July-2019].

[FI04] Ansgar Fehnker and Franjo Ivančić. Benchmarks for hybrid systems verification. In *International Workshop on Hybrid Systems: Computation and Control*, pages 326–341. Springer, 2004.

[INT] INTELLIJ. `https://account.jetbrains.com/login`. [Online; accessed 5-July-2019].

[ISS99] Rolf Isermann, Jochen Schaffnit, and Stefan Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7(5):643–653, 1999.

[JAV] JAVA. `https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`. [Online; accessed 5-July-2019].

[PHK] Franz Josef Preyser, Bernhard Heinzl, and Wolfgang Kastner. Rpdevs abstract simulator. In *Proc. of ASIM-Workshop Simulation technischer Systeme/Grundlagen und Methoden in Modellbildung und Simulation*, page 6.

[pow] POWERRP. `https://sourceforge.net/projects/powerrpdevs/`. [Online; accessed 5-July-2019].

[PRO] PROTO. `https://developers.google.com/protocol-buffers/`. [Online; accessed 5-July-2019].

[PUT] PUT. `https://www.putty.org/`. [Online; accessed 16-July-2019].

[PWBH12] Xiufeng Pang, Michael Wetter, Prajesh Bhattacharya, and Philip Haves. A framework for simulation-based real-time whole building performance assessment. *Building and Environment*, 54:100–108, 2012.

[SCP] SCP. https://winscp.net/eng/index.php. [Online; accessed 16-July-2019].

[SPK17] Stefan Seifried, Franz Josef Preyser, and Wolfgang Kastner. Enabling hardware-in-the-loop for building automation networks: A case study for bacnet and powerdevs. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 8119–8125. IEEE, 2017.

[SRC] SRC. https://sourceforge.net/u/fpreyser/powerrpdevs/ci/BakStefan/tree/. [Online; accessed 21-July-2019].

[TC2] TC2. https://www.beckhoff.com/default.asp?download/tcatdow.htm?id=159876032042844. [Online; accessed 5-July-2019].

[Wet11] Michael Wetter. Co-simulation of building energy and control systems with the building controls virtual test bed. *Journal of Building Performance Simulation*, 4:185–203, 09 2011.

[WIR] WIRE. https://www.wireshark.org/. [Online; accessed 16-July-2019].

[XMI] XMI. https://sourceforge.net/projects/xming/. [Online; accessed 16-July-2019].