Enabling Hardware-in-the-Loop for Building Automation Networks: A Case Study for BACnet and PowerDEVS

Stefan Seifried, Franz Josef Preyser, Wolfgang Kastner
Automation Systems Group
Technische Universität Wien
{sseifried, fpreyser, k}@auto.tuwien.ac.at

***Abstract*—Hardware-in-the-Loop (HIL) is a well-established concept for developing and testing embedded systems. While it is widely used in industrial automation and the automotive area, it is rarely applied to Building Automation Systems (BAS). This work proposes the interconnection of a prominent building automation protocol, namely Building Automation and Control network (BACnet), and a simulator, PowerDEVS, to facilitate HIL testability of new and existing building automation networks. The *Discrete Event Systems Specification* formalism, used by PowerDEVS, is especially geared towards systems that show both discrete event and continuous state characteristics. Hence, the field of BAS is a prime candidate for such simulators. Further, the interconnection of HIL simulation and BAS via building automation networks allows for easy and location-independent testing of single field devices on one hand, and distributed BAS applications on the other hand. Therefore, design issues, and the pros and cons of a bridging solution between BAS and simulators are discussed throughout the course of this work. The paper introduces a modular software architecture for the interconnection of building automation networks and simulation. This allows for easy portability of the proposed design to other HIL simulators or building automation network protocols. The feasibility of the proposed approach is validated by means of a proof-of-concept.**

I. Introduction

Hardware-in-the-Loop (HIL) testing is a well-established technique used during the design and evaluation of (embedded) systems. The underlying idea is the interconnection of hardware devices with a simulated environment, instead of building up a complete testbed out of physical components. Despite reduced costs of testing, one of the benefits is a greater variety of system conditions that can be applied to the tested elements, some of which cannot be created without high costs or in a safe manner (e.g., extreme temperatures).

In case of Building Automation Systems (BAS), HIL is usually only performed on a per device basis. However, the distributed nature inherent to applications in the field of BAS boosts the need for testing of multiple field devices throughout a building automation network. A fitting solution in form of a simulation middleware has been briefly introduced in [1]. However, the focus of this work is on the co-simulation approach to integrate and interconnect different simulators and their corresponding models. While the co-simulation approach facilitates the reuse of existing simulation models, it drastically increases the complexity of a possible HIL approach.

Also, our proposed bridging approach is designed to interconnect a wide range of building automation network protocols and available simulators. Despite differing requirements on physical interconnection interfaces, each building automation network and simulator usually uses its very own semantic model. Hence, the communication endpoints of the proposed solution need to be specifically tailored to the respective information model of the interconnected entity. Hence, our software architecture and interconnection scheme was evaluated by the use of Building Automation and Control network (BACnet) and the Power DEVS (PowerDEVS) hybrid systems simulator. BACnet was chosen due to its wide-spread usage, numerous ready-to-use devices and its recognition as an international standard for BAS [2]. PowerDEVS ([3]) was chosen as a novel representative of the family of Discrete Event System Specification (DEVS) hybrid system simulators. The DEVS formalism allows for efficient simulation of discrete events and continuous state systems. In BAS systems, both discrete event systems, like light control, and continuous state systems, like airflow valves, appear equally often. Hence, PowerDEVS is an efficient and versatile solution for the simulation and modeling of physical relations that occur in BAS applications.

This paper discusses the design and architecture of a versatile interconnection solution between building automation networks and simulators while enabling HIL simulation and testability of distributed BAS by using the example of BACnet and PowerDEVS. Therefore, an introduction to the state-of-the-art of the utilized technologies is given for HIL in Section II, BACnet in Section III and PowerDEVS in Section IV. The design and architecture of the bridging solution is then discussed in Section V. Further, the proof-of-concept implementation is presented in Section VI. Finally, the paper concludes with a reflection on the work done and future matters in Section VII.

II. Hardware-in-the-Loop

HIL denotes the concept of interconnecting hardware devices with a simulation environment. Rather than validating a concept by a pure virtual simulation model, the interaction of a partly simulated and physical environment is advised. The hardware component evaluated using the HIL approach is further denoted as *hardware-under-test* to distinct it from other physical devices present in a HIL setup. The utilization of HIL is especially beneficial, when the resources needed to create a proper virtual simulation model exceed the necessary resources for functional development of a product.

Despite economical considerations, [4] describes the problem solved by HIL from the simulation fidelity point of

978-1-5386-1126-5/17/$31.00 ©2017 IEEE

view. First of all, there is an inherent uncertainty in the technical process, which cannot necessarily be precisely modeled. Second, the environmental coupling between different parts can not be modeled in an exact way. This effects the accuracy of the overall system model, when compared to the technical process. Furthermore, a holistic simulation model might not even be feasible due to the inherent complexity of the entangled physical relations.

Another criterion in favor of HIL is that components of a BAS are usually manufactured by different vendors. Hence, there is not necessarily detailed information available about the internal behaviour or control algorithm of an automation component. In this case, it might not be feasible to build a simulation model of such a component at all. Thus, the only option left is to interconnect such a component directly with the *hardware-under-test*.

Finally, another use case for HIL is pre-commercial development. This is especially useful, when the behaviour of the component needs to be validated under extreme conditions inside a protected testbed. Such conditions usually cannot be created easily or in a safe way using a physical test environment. Additionally, there is also the possibility of artificially injecting faults into the test setup to validate the correct function of the *hardware-under-test* in case of an error. This technique is exceptionally valuable when verifying safety-critical components.

While HIL testing is well-established in the domain of industrial automation and the automation of safety-critical systems, it is seldom considered for BAS. Nonetheless, the domain of buildings is a prime candidate of HIL simulation due to the underlying complex ecosystems where environmental influences and uncertainties introduced by the behaviour of their tenants have to be taken into account. Further, BAS applications are usually implemented in a decentralized and distributed fashion, involving several different devices. The decentralized nature of BAS simplifies the implementation of a HIL approach, since individual components can be easily replaced by a simulation model without affecting the remaining system.

In a conventional HIL setup, the simulator attaches to the *hardware-under-test* via its digital or analog inputs and outputs. While this type of interconnection is a valid approach for locally restricted automation systems, it becomes increasingly difficult for distributed approaches. Since BAS are mostly implemented utilizing building automation networks, a possible solution to this problem is the interconnection of the simulation via the network instead.

III. BACnet

BACnet is one of the predominant interconnection standards in the field of BAS and was designed to provide maximum interoperability between different manufacturers and trades. The scope of BACnet applications ranges from Heating, Ventilation and Air Conditioning (HVAC), lighting, shading to safety and security. While the development of BACnet was ongoing since 1987, it was only first released as an official standard in 1995 as an ANSI/American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) norm. In 2003, BACnet was also adopted as ISO standard, where the latest iteration has been published in 2014 [2].

The BACnet network stack is based on a four-layer model (See Figure 1) which is equivalent to the physical, data-link, network and application layer of the ISO/OSI model. The physical and data-link layer are comprised of a diverse set of existing communication standards ranging from wired EIA485 with Master Slave / Token Passing (MS/TP) over Ethernet and Internet Protocol (IP) to the wireless domain (i.e. ZigBee). Interconnection between the different communication standards is achieved with the help of the common BACnet network layer and BACnet router devices. The application layer defines the interface to the various applications, providing services to access data, and alarm and management services.

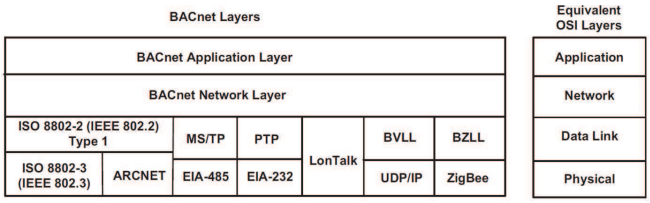

Fig. 1. BACnet collapsed architecture [2]

Additionally, an object-oriented information model is defined to standardize the data structures for the *network-visible* representation of information. Therefore, BACnet devices are always modeled as a collection of objects. While the latest iteration of the BACnet standard offers up to 60 different object types, only 25 object types were present in the first iteration of the standard. This is done to ensure backwards compatibility with earlier BACnet devices and is owed to the longevity of BAS installations.

The BACnet object model offers a set of generic object types and properties. It is then the responsibility of the application to map the BACnet objects and BACnet services to application-specific data structures. Examples of such generic BACnet object types are the *Binary Input Object Type*, *Binary Output Object Type*, *Analog Input Object Type* and *Analog Object Type* which are modeled after the four basic I/O categories found in typical embedded devices. The root node of the object hierarchy in BACnet is the *Device Object*.

Every object type in BACnet is composed of properties, which in turn consist of a *Property Identifier* and a *Property Datatype*. Furthermore, properties are classified along the *Conformance Code* into required and optional properties. Table I shows all the required properties for a *Binary Input Object Type*. BACnet *objects* are addressable by their *Object_Identifier* property. The value of the *Object_Identifier* property is unique throughout the BACnet device. Further, the *Object_Identifier* of an arbitrary object part of a device, combined with the *Object_Identifier* of the *Device* object itself, is unique within the whole BACnet network.

Property Identifier	Property Datatype
Object_Identifier	BACnetObjectIdentifier
Object_Name	CharacterString
Object_Type	BACnetObjecType
Present_Value	BACnetBinaryPV
Status_Flags	BACnetStatusFlags
Event_State	BACnetEventState
Out_Of_Service	BOOLEAN
Polarity	BACnetPolarity
Property_List	BACnetARRAY[N] of BACnet-PropertyIdentifier

TABLE I
REQUIRED PROPERTIES OF A BACNET BINARY INPUT OBJECT [2]

One of the particularly interesting BACnet services in conjunction with the BACnet object model is Change of Value (COV). The COV is part of the *alarm and event* services, and allows a COV-client to subscribe with a COV-server to be notified about changes in object properties. Furthermore, a trigger level for such a notification can be specified for certain properties of a BACnet object. While this feature frees a BACnet-client from constantly polling a monitored property, it is unfortunately only optional.

As described in Section V, the proposed approach especially utilizes the *Binary Input/Output Object Type* and the *Analog Input/Output Object Type*. Furthermore, the COV service is implemented for the *value* property of those object types. Hence, a possible *hardware-under-test* is able to choose between *event* or *polling* based mode of communication for incoming values from the PowerDEVS simulation.

IV. POWERDEVS

PowerDEVS[1] [5] is a hybrid[2] systems simulator. Its simulation engine is based on the classic DEVS formalism [6], [7]. For the description and numerical treatment of the continuous model parts, the family of Quantized State Systems (QSS)-methods [7], [8], [9], [10], [11], [12], [13] is used.

Classically used ODE solvers in combination with iterative event location mechanism (e.g., regular falsi) suffer from considerable performance breakdowns when facing simulation models with a moderate number of state events [14], [15], which are often accompanied by model discontinuities. State events though are immanent to hybrid models. DEVS in combination with QSS, as used in *PowerDEVS*, is a very promising pairing concerning performance of hybrid systems simulation and a meaningful alternative to co-simulation approaches.

V. BRIDGING BACNET AND POWERDEVS

The aim of this work is to design a bridging component between a simulation software module and arbitrary BACnet components to enable HIL simulation for BAS. Figure 2 shows a SDL [16] system level definition of the proposed interconnection solution. The system architecture of the proposed bridging solution is split according to the *separation-of-concerns* principle into two modules, the *Building Automation Front-end* and the *Simulation Back-end*. This design choice enables the easy replacement of either the targeted BAS or the used simulation software component. Thus, effortless adaption of the presented approach to other building automation technologies or simulators is assured. Further, it even allows for the facilitation of different computing devices due the use of standard Ethernet (*Front-end/Back-end protocol*) as link between the *front-end* and *back-end* software components. The *Front-end/Back-end protocol* is further described in Section V-B, whereas the *Building Automation Front-end* is described in Section V-A and the *Simulation Back-end* in Section V-C.

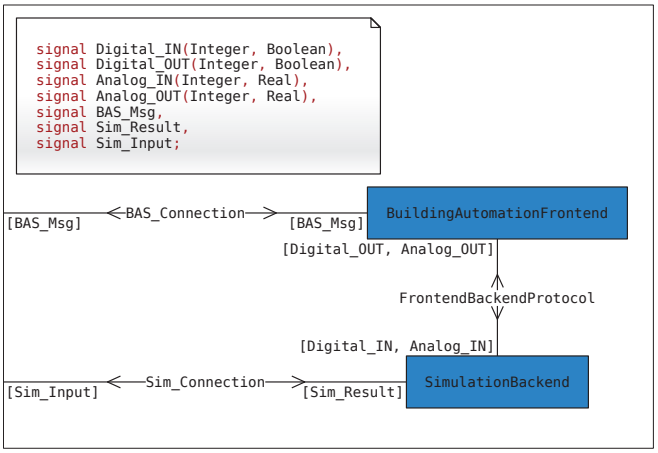

Fig. 2. Specification and Description Language (SDL) System Model

A. Building Automation Front-end

The *Building Automation Front-end* consists of three building blocks, the *BACnet/IP Server*, the *Translation Logic*, and the *Simulation Connector* (Figure 3). It is primarily designed to hide specific details of the utilized building automation network protocol from the simulator and effectively works as a *gateway* device.

According to the *gateway* characteristics of the *Building Automation Front-end*, a stateful communication translation process needs to be implemented [17]. Therefore, a local data storage and translation rules between the data representation formats of the attached building automation network and simulator are needed.

As already described in Section III, a BACnet device is modeled with the help of BACnet objects. While the newest iteration of the standard defines up to 60 different object types, many devices still reside on the 25 object types specified in earlier instances of the norm. Furthermore, typical BACnet devices usually model their ”network-visible” data according to physical interfaces of a hardware device. Hence, the *Binary Input Object Type* and *Binary Output Object Type* are used to depict digital values, and the *Analog Input Object Type* and *Analog Output Object Type* model analog values. Thus, it is sufficient to consider only those four object types to model a proper HIL simulator BACnet device.

[1] https://powerdevs.sourceforge.io/
[2] in the spirit of combined discrete and continuous dynamics

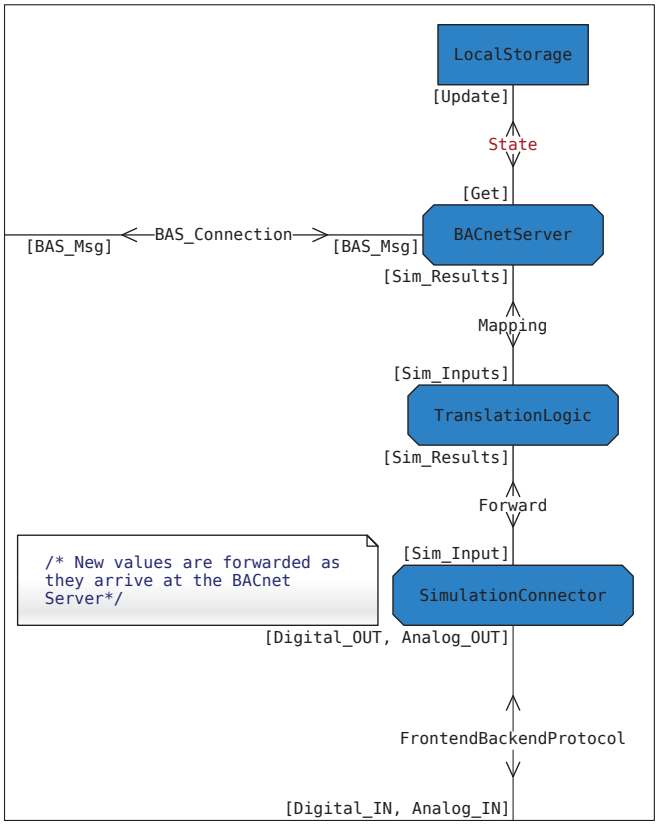

Fig. 3. SDL Model of Building Automation Front-end

A straightforward approach to design a simple, yet versatile interface to the simulator is to adapt the well-established interconnection approach from conventional HIL. As described in Section II, simulators are usually attached to the *hardware-under-test* by means of the available digital or analog inputs and outputs. Hence, the *Simulation Connector* mimics this behavior by also accepting either digital or analog values.

In the case of a BACnet based *Building Automation Front-end*, the *Translation Logic* can then be achieved by a simple one-to-one mapping between the corresponding digital and analog I/O data types of the simulator and building automation network. However, this is not necessarily the case for other communication protocols in the field of BAS. Consider, for instance, that even the recent BACnet standard contains other object types for modeling BAS application behaviour than the standard input and outputs (cf. *Lighting Output Object Type*).

Therefore, a dedicated *Translation Logic* is needed. The purpose of such a building block is twofold. On the one hand, its task is to map information from the analog and digital values provided by the *Simulation Connector* to the corresponding BACnet objects modeled at the *BACnet Server*. On the other hand, it is responsible to translate information between those building blocks.

The mapping relation between the *Simulation Connector* and the *BACnet Server* is a one-to-many relation. While a digital or analog value can be assigned to many properties of BACnet objects, a property can only be mapped to exactly one value. In case aggregated values are needed, those have to be created at the *Simulation Back-end*. Further, transformations of the digital and analog values are allowed in form of translation rules. The translation rules convert between the BACnet representation and the raw digital and analog values of the *Simulation Connector*. Every mapping relation is attached to exactly one translation rule. Further, rules are stateless and can therefore be reused for different mapping relations, allowing for a very flexible, yet compact translation process.

B. Front-end/Back-end Protocol

As described in Section V-A, the *Building Automation Front-end* interface is modeled according to traditional HIL like digital and analog I/O interface. Therefore, as depicted in Figure 4, the message frame format could be kept small and simple. It is basically modeled according to the terminal block of a programmable logic controller (PLC), where the individual connections are named by their type (i.e., digital or analog), direction (i.e., input or output) and consecutively numbered.

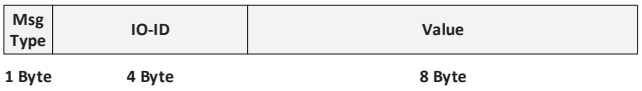

Fig. 4. Message Frame Format

The *MsgType* field is 1 byte wide and can assume one of the following enumeration values:

1 . . . Digital In
2 . . . Digital Out
3 . . . Analog In
4 . . . Analog Out

Next, the *IO ID* field is used to assign a unique ID within the respective *MsgType*. Hence, it is possible that there exists both, an *ID0* for *Analog In* and *Analog Out*. Finally, the *Value* field either holds an 8 byte double value for analog type I/Os, or a 1 byte boolean value for digital type I/Os. Unused space inside the fields is always padded with zeroes.

In order to design the protocol in a manner suitable for resource constrained devices, the User Datagram Protocol (UDP) network protocol was utilized. One of the shortcomings of UDP is its lack of reliable message transmission. This can result in lost messages and lead to wrong simulation results due to missing values. However, this issue is only presented when communicating from the *Building Automation Front-end* to the *Simulation Back-end* due to the event triggered mode of communication. In the opposite way, information is forwarded on a regular basis and missing messages only result in delayed value updates at the *Building Automation Front-end*. This flaw can be easily fixed by utilizing the Transmission Control Protocol (TCP) protocol instead of *UDP*, implementing a

custom protocol supporting reliable message transfer on top of UDP or run both components on a single device.

Further, data from the *Building Automation Front-end* is forwarded to the *Simulation Back-end* upon arrival of new values from the interconnected BAS. Contrary, simulation results are transmitted according to a regular timing interval. Hence, the *Building Automation Front-end* facilitates a COV like behaviour, but acts like a *Sample and Hold* element when viewed from the BAS. This behaviour is inefficient in terms of network utilization, particularly considering the message flow from the back-end to the front-end.

A possible solution for this issue is to implement a COV behavior pattern for the data exchange between the *Simulation Back-end* and the *Building Automation Front-end*. Therefore, an incoming transaction from the BAS at the front-end triggers a polling transaction to the back-end. During this polling transaction, the BAS transaction is kept in a pending state. As soon as the response to the polling transaction arrives from the back-end, it is processed and the BAS transaction can be answered and closed.

While this mechanism seems more sophisticated than sending data from the *Simulation Back-end* to the *Building Automation Front-end* on a scheduled timing interval, it impedes the implementation of BACnets COV feature (cf. Section BACnet). While a digital value can be simply forwarded in the event its value changes, a threshold has to be defined for an analog value. For this reason, the COV increment defined in a BACnet object has also to be announced to the back-end. Hence, the interdependence between the *Building Automation Front-end* and *Simulation Back-end* is drastically increased.

Furthermore, requests incoming from the actual building automation network need to be usually answered in a timely fashion. Otherwise, the pending request message transaction will timeout and will at least result in a re-transmit of the datagram. This problem becomes even more aggravating, when considering other physical layers of the BACnet stack than Ethernet and IP. A good example is the ZigBee low power wireless network protocol. In [18] it was shown, that it is possible to tunnel IEEE 802.15.4 telegrams, the basis of ZigBee, through an Ethernet link. However, this was only achieved by exploiting the re-transmit mechanism of the protocol. In the worst case, it took up to 3 packet re-transmissions to finalize a message transaction with a proper acknowledgement message. Hence, the design choice for a stateful implementation of the *Building Automation Front-end* was made.

C. Simulation Back-end

The *Simulation Back-end* is comprised of two components, the *UDP Sink* and *UDP Source* (Figure 5). The *UDP Sink/-Source* components translate messages between the information format of the *Front-end/Back-end Protocol* and the representation used by the *PowerDEVS* simulator. As described in Section V-B, the *IO ID* is used to assign incoming messages to signals used by the *Control Model*.

The *UDP Sink/Source* components are named from the simulators point of view. So the *UDP Sink* actually emits messages

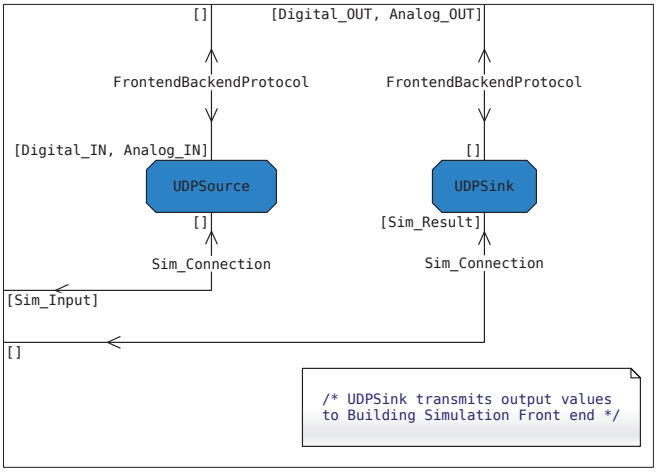

Fig. 5. SDL Model of Simulation Back-end

to the network and the *UDP Source* receives telegrams from the network. Further, the *PowerDEVS* simulator processes the input values from the *UDP Source* block in a stateful manner. This is done by storing and applying received values to the respective signals of the *Control Model*. The simulation results are then sent to the *UDP Sink* and in turn regularly forwarded to the *Building Automation Front-end*.

However, the simulation results need to be passed on to the building automation network in a timely manner. While BAS systems are concerned with rather slow physical processes, there are also use cases that call for immediate responses, like turning on the lights. Therefore, PowerDEVS supports soft real-time simulation when executed under a standard Operating System (OS) and is even able to provide hard real-time capabilities when operated on a proper real-time OS (e.g. Linux RTAI).

VI. Implementation

The feasibility of the presented approach was evaluated by creating a proof-of-concept implementation for both the *Building Automation Front-end* and the *Simulation Back-end*. The prototype software was then deployed to a single regular PC and connected to a small-scale BAS testbed.

The *Building Automation Front-end* was programmed in Java, due to the large number of available components and its support for numerous hardware platforms. The *BACnet4j*[3] open-source software component was utilized to serve as a connector to a BACnet/IP network. Furthermore, the *Translation Logic* was implemented with the help of *JAXB*[4], thus enabling the automatic creation of storage classes and a verifiable XML format for persistence. The one-to-one mapping (see Section V-A) between the BACnet objects and the corresponding message type entities used by the *Simulation Connector*

[3]https://github.com/infiniteautomation/BACnet4J
[4]https://jaxb.java.net/

was hardcoded. Finally, JLine3[5], a Java implemented terminal, provided an easy-to-use user interface for debugging, logging and examination of the ongoing processes at the *Building Automation Front-end*.

Listing 1. Google Protocol Buffers defintion

```
package proto;

option java_package = "at.ac.tuwien.auto.bacnetsrv";
option java_outer_classname = "HBASimProtos";

// digital in: front-end -> sim
message DIN {
    int32 id = 1;
    bool value = 2;
}
// digital out: front-end <- sim
message DOUT {
    int32 id = 1;
    bool value = 2;
}
// analogue in: front-end -> sim
message AIN {
    int32 id = 1;
    double value = 2;
}
// analogue out: front-end <- sim
message AOUT {
    int32 id = 1;
    double value = 2;
}
```

The *Front-end/Back-end Protocol* was implemented with the help of the *Google Protocol Buffers*[6] compiler toolchain. This allowed to generate the necessary C code for the *PowerDEVS* simulator and Java code from a single protocol definition file. Listing 1 shows a stripped down version of the *Protocol Buffers* definition file.

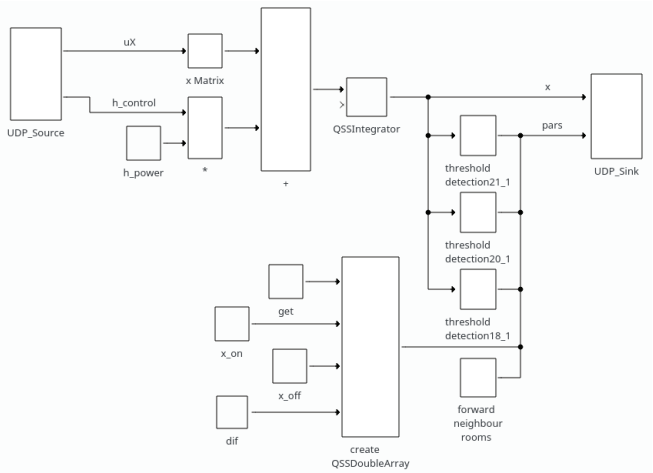

Fig. 6. Simulation model of a single room

The software components needed for the implementation of the *Simulation Back-end* could be entirely integrated into PowerDEVS. The *UDP Sink* and *UDP Source* were implemented as control model components. Hence, the *Simulation Back-end* can be easily configured within the *PowerDEVS* simulation editor. The data forwarded from the *Building Automation Front-end* is then used as an input vector for control models utilized within the *PowerDEVS* simulator. Figure 6 depicts a heating control model that was used in conjunction with physical components throughout the evaluation of the proposed bridging solution.

The chosen evaluation environment for the proposed bridging solution consists of a small-scale HVAC process model controlled by a *Beckhoff BACnet Programmable Logic Controller (PLC)*. The *Beckhoff BACnet PLC* serves as the *hardware-under-test* for the prototype of the proposed bridging concept. Connections at the terminal block from the PLC were removed as required and replaced by external signal values originating from the HIL simulation. The proof-of-concept implementation of both the *Building Automation Front-end* and the *Simulation Back-end* were deployed to a single workstation. Thus, the connection between the front-end and back-end was not achieved by using real Ethernet wiring. Apart from that, interconnection between the front-end and the PLC was achieved using BACnet/IP via an Ethernet network involving one routing device. The results were evaluated and verified by the means of the logging and console features of the *Building Automation Front-end*, an open-source *BACnet* client (*Yet Another BACnet Explorer*[7]) and the Human Machine Interface (HMI) of the PLC. A screenshot of the HMI is depicted by Figure 7, representing also a schematic overview of the evaluation environment. Further details about the HVAC process model can be found in [19].

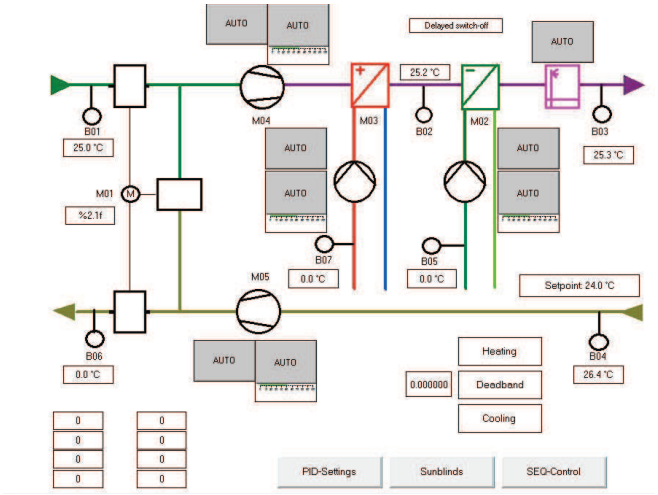

Fig. 7. Beckhoff HMI [19]

[5]https://github.com/jline/jline3
[6]https://developers.google.com/protocol-buffers/
[7]https://sourceforge.net/projects/yetanotherbacnetexplorer/

VII. Conclusion and Outlook

This paper presents an interconnection solution between BACnet and PowerDEVS to facilitate easy HIL testing of BAS components. While traditionally HIL testing is realized by direct interaction with the physical input and outputs of the *hardware-under-test*, this solution provides values from the simulation environment to the BACnet network. Hence, this solution is able to provide simulated datapoint values to an arbitrary number of interconnected BACnet devices and may significantly extend the range and versatility when compared to traditional HIL. Hence, a distributed BAS can be tested and evaluated by interacting with a single simulator.

However, it can be argued, that the same result can be achieved by directly tapping into the inputs and outputs of BACnet devices. The BACnet devices further provide the values from the simulation to the BACnet network and eradicate the need for the *Building Automation Front-end*. Additionally aggravating, such an approach would benefit from broad compatibility not only for BACnet, but throughout different BAS solutions. In fact, the needed complexity is only shifted from the proposed software solution to a hardware solution. Also, there is still at least the need for some hardware I/O device, along a software module connecting it to the PowerDEVS simulator. Additionally, characteristic graphs for different types of sensors and actuators have to be considered by the simulation. Furthermore, such an approach cannot be easily applied to an already deployed BACnet network due to its spatial extent.

While functionality in BAS is usually implemented in a distributed manner, there are also use cases for centralized applications that are contained on a single device. Hence, it is usually not foreseen to utilize datapoints from different *BACnet servers* on such devices and the presented approach is not feasible. Fortunately, the *Building Automation Front-end* can be adapted to interface with physical I/Os and therefore directly interact with the I/O of such a device. However, this reduces the discussed approach to a traditional HIL solution.

The current design is focused on the simulation of physical processes. A possible improvement to this approach is the simulation of Direct Digital Control (DDC) devices or even entire network segments. Especially, existing networks may benefit from integration testing with complete sets of simulated devices before their actual deployment.

Finally, the main feature of the presented approach resides in its modular design. This enables the easy adaption to other BAS communication solutions, like KNX or ZigBee. The same holds true for the simulation endpoint of the proposed bridging solution. The only requirement for a simulator to interact with the *Simulation Back-end* is to provide means for accepting input and emit output values during runtime.

Future work will be be focused on adding additional *Simulation Back-end* and *Building Automation Front-end* implementation. This will enable the necessary means to provide not only performance data (e.g., network load, CPU load) for the discussed bridging solution, but will also enable to benchmark connected simulators.

References

[1] M. Wetter, "Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed," *Journal of Building Performance Simulation*, vol. 4, no. 3, pp. 185–203, 2010.

[2] ISO 16484-5:2014, "Building Automation and Control Systems (BACS)Part 5: Data Communication Protocol," International Organization for Standardization, ISO Standard, 2014.

[3] F. Preyser, B. Heinzl, P. Raich, and W. Kastner, "Towards Extending the Parallel-DEVS Formalism to Improve Component Modularity," in *Tagungsband des Workshops der ASIM/GI-Fachgruppen STS und GMMS 2016*, 2016, pp. 83–89.

[4] M. Bacic, "On hardware-in-the-loop simulation," in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 3194–3198.

[5] F. Bergero and E. Kofman, "PowerDEVS: A tool for hybrid system modeling and real-time simulation," *Simulation*, vol. 87, no. 1-2, pp. 113–132, 2011.

[6] B. P. Zeigler, *Theory of Modeling and Simulation*. Krieger Pub Co, 1976.

[7] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000, no. April.

[8] E. Kofman and S. Junco, "Quantized-state systems: a DEVS Approach for continuous system simulation," *Trans. Soc. Model. Simul. Int.*, vol. 18, pp. 123–132, 2001.

[9] E. Kofman, "A Second Order Approximation for DEVS Simulation of Continuous Systems," *Simul. Trans. Soc. Model. Simul. Int.*, vol. 78, no. 2, pp. 76–89, 2002.

[10] ——, "Quantization-Based Simulation of Differential Algebraic Equation Systems," *Simul. Trans. Soc. Model. Simul.*, vol. 79, no. 7, pp. 363–376, 2003.

[11] ——, "Discrete Event Simulation of Hybrid Systems," *SIAM J. Sci. Comput.*, vol. 25, no. 5, pp. 1771–1797, 2004.

[12] ——, "A third order discrete event method for continuous system simulation," *Lat. Am. Appl. Res.*, vol. 36, no. 2, pp. 101–108, 2006.

[13] G. Migoni, E. Kofman, and F. E. Cellier, "Quantization-based new integration methods for stiff ordinary differential equations," *Simulation*, vol. 88, no. 4, pp. 387–407, 2012.

[14] J. Fernández and E. Kofman, "A StandAlone Quantized State System Solver for Continuous System Simulation," *Simul. Trans. Soc. Model. Simul. Int.*, pp. 1–36, 2014.

[15] G. Migoni, P. Rullo, F. Bergero, and E. Kofman, "Efficient simulation of Hybrid Renewable Energy Systems," *Int. J. Hydrogen Energy*, 2016.

[16] F. Babich and L. Deotto, "Formal methods for specification and analysis of communication protocols," *IEEE Communications Surveys Tutorials*, vol. 4, no. 1, pp. 2–20, First 2002.

[17] S. Seifried and W. Kastner, "Reliable control network gateways for building automation networks," in *Proceedings of the Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP*, 2016, pp. 1–8.

[18] L. Krammer, S. Seifried, and W. Kastner, "A fault-tolerant Backbone for IEEE 802.15.4 based Networks," in *Proceedings of the Conference on Industrial Technology (ICIT)*, 2014, pp. 736–742.

[19] B. Bezeczky, "Integration of a BECKHOFF PLC into an HVAC model plant," Technical University of Vienna, Tech. Rep., 2015. [Online]. Available: https://www.auto.tuwien.ac.at/bib/pdf_TR/TR0177.pdf