

Academic year
2024 - 2025

Provide a title

Inte Vleminckx

Supervisors

prof. dr. H. Vangheluwe, UAntwerpen
R. Mittal, Doctoral Fellow, UAntwerpen



University of Antwerp
| Faculty of Science

Disclaimer

This document is an examination document that has not been corrected for any errors identified. Without prior written permission of both the supervisor(s) and the author(s), any copying, copying, using or realizing this publication or parts thereof is prohibited. For requests for information regarding the copying and/or use and/or realisation of parts of this publication, please contact to the university at which the author is registered.

Prior written permission from the supervisor(s) is also required for the use for industrial or commercial utility of the (original) methods, products, circuits and programs described in this thesis, and for the submission of this publication for participation in scientific prizes or competitions.

This document is in accordance with the faculty regulations related to this examination document and the Code of Conduct. The text has been reviewed by the supervisor and the attendant.

Contents

1	Introduction	1
2	Proposed Approach	2
2.1	The First Steps	2
2.1.1	A Simple Modelica Model	3
2.1.2	Interaction With The Control Loop	3
2.1.3	Complete Intergration	4
3	Modelica Library Overview	5
4	Results	6
5	Conclusion	7

1 Introduction

The development of products in the heating, ventilation, and air conditioning (HVAC) industry presents significant challenges in testing and validation. Building physical prototypes for every design iteration is often costly and time-consuming. A promising alternative is to model the most expensive or complex components in a virtual environment, enabling early testing without full-scale prototypes. This approach allows the evaluation of critical subsystems, particularly the control software that regulates HVAC systems.

In this study, we investigate how to test the control loop of a heating and ventilation system by modeling all physical elements—such as the valve, the actuator controlling the valve, the flow sensor, the pipe network, and the pressure pump that generates the fluid flow. The control loop, which determines the actuator setpoint based on the flow sensor measurements, will interact with the virtual model using co-simulation techniques. To assess the feasibility and performance of this approach, we compare two testing strategies: Software-in-the-Loop (SiL), and Hardware-in-the-Loop (HiL). In SiL testing, the model interacts with a compiled version of the control loop running on a separate system, with all connections established virtually. In HiL testing, the model runs on one system while the control loop is executed on the actual embedded hardware used in the real setup, with physical connections between the two.

Our methodology proceeds in stages. First, we develop a simple flow circuit in Modelica to demonstrate basic co-simulation capabilities. Using this model, we investigate how to integrate it with SiL and HiL environments. Once this foundation is established, we expand the Modelica component library with more detailed and realistic system elements. Finally, we construct an advanced flow circuit model and benchmark SiL results against HiL results to evaluate performance differences and validate the modeling approach.

Inte: add what can be expected in each upcoming section in one sentence.

2 Proposed Approach

2.1 The First Steps

To address the problem at hand, we must first establish a clear understanding of the overall problem statement. As outlined in the introduction, our goal is to test the control loop of a heating and ventilation system by modeling all physical elements and allowing the control loop to interact with this model. To achieve this, we need to define, as simply as possible, what the model requires as input from the control loop and what the control loop requires as input from the model, without yet considering detailed configuration parameters of the individual physical components. With this understanding in place, we can represent the interaction between the control loop and the model using the following diagram:

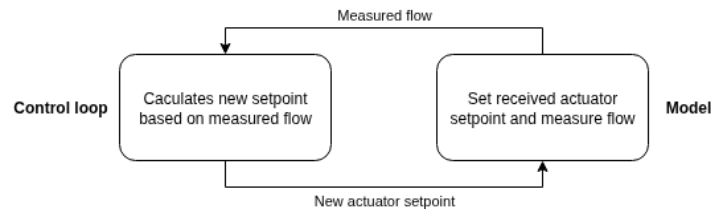


Figure 2.1: Simple representation of interaction between control loop and model

Next, we provide a more detailed explanation of the process illustrated in Figure 2.1. In this setup, the control loop receives a target flow setpoint. Although the source of this setpoint is not relevant to our study, it defines the desired flow rate in the circuit. To achieve this flow rate, the control loop calculates an actuator setpoint, which adjusts the valve opening to provide the required flow. In order to perform this calculation, the control loop must receive feedback from the model in the form of the measured flow in the circuit. By comparing the measured flow to the target flow, the control loop can determine the necessary actuator setpoint and adjust the valve position accordingly.

With this simplified setup, we can construct a Modelica model that accepts one real-valued input and produces one real-valued output—the actuator setpoint and the measured flow, respectively. For testing purposes, we can initially provide fixed output values from the model to the control loop to observe how the actuator setpoint evolves over time. To enable full co-simulation, we must also determine how to establish interaction between the Modelica model and the control loop. The workflow for this process is as follows:

1. Develop a simple Modelica model.
2. Investigate how the compiled control loop can be accessed and controlled using Python.
3. Integrate the Modelica model into the Python code so that the control loop can exchange data with the model in real time.

2.1.1 A Simple Modelica Model

A very simple Modelica model can be represented as shown in Figure 2.2.

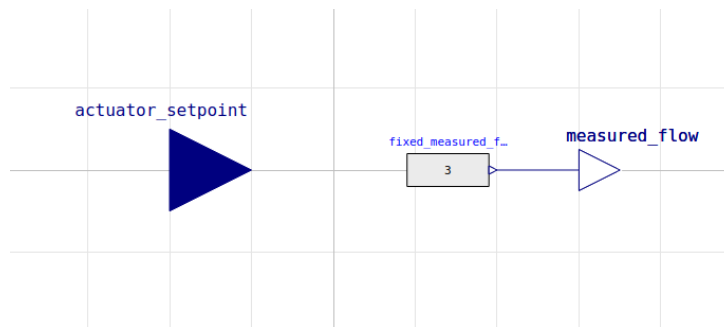


Figure 2.2: A simplified model representatino

This model accepts an actuator setpoint as input and produces a measured flow as output, which is currently fixed at a value of 3 for testing purposes.

2.1.2 Interaction With The Control Loop

The first step in enabling interaction with the control loop is to compile the common platform—containing the control loop—into an executable. This common platform is implemented in C and is compiled using CMake. Due to privacy regulations of the company related to this study, the source code of the common platform cannot be shared.

Once the executable is available, the next step is to establish a method for interacting with the common platform. There are two primary communication interfaces used by the platform:

1. A Flow Sensor Board (FSB), which emulates the behavior of a real flow sensor using a UART connection.
2. A Modbus interface, which allows reading registers from the embedded control board using a TCP connection.

In the case of Software-in-the-Loop (SiL) testing, the common platform is executed locally. A mock Modbus connection is created to replicate the behavior of a physical Modbus link to an embedded board. To keep the discussion focused, we first consider only the SiL setup. A schematic representation of this communication is shown in Figure 2.3.

To enable this interaction, we must first create virtual serial ports that allow communication with the common platform. This can be achieved using the following commands:

```
$ sudo socat -d -d pty,link=/dev/ttyV1,raw,echo=0 pty,link=/dev/ttyV2,raw,echo=0
$ sudo chmod 777 /dev/ttyV1 && sudo chmod 777 /dev/ttyV2 && \
sudo socat -d -d -u OPEN:/dev/ttyV1,raw tcp:localhost:8888,reuseaddr
```

Listing 2.1: Creating virtual ports for communication

This configuration uses socat to create a pair of virtual serial ports and forward data through a TCP socket. First, it establishes two pseudo-terminal devices—/dev/ttyV1 and /dev/ttyV2—which function

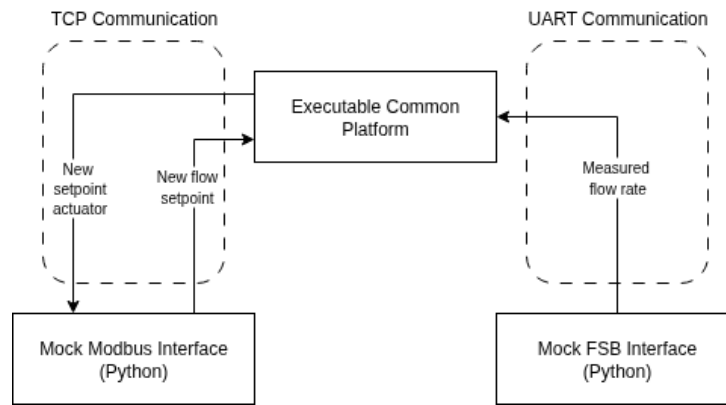


Figure 2.3: Overview communication with common platform

as the two ends of a virtual null-modem cable: any data written to one port immediately appears on the other. Next, their permissions are updated to allow unrestricted access, and a second socat process forwards all data received on `/dev/ttyV1` to a TCP connection on port 8888.

The last part is to those ports using python, such that we can interact with the common platform using python code.

2.1.3 Complete Intergration

Go futher from here, talk about how we connect to the ports using the python Uart and TCP code

3 Modelica Library Overview

4 Results

5 Conclusion

Bibliography