Simulation Environment for Modular Automation Systems

Björn Leander*†, Tijana Marković*, Aida Čaušević*, Tomas Lindström†, Hans Hansson*, Sasikumar Punnekkat*
**Mälardalen University*,Västerås, Sweden,
{bjorn.leander, tijana.markovic, aida.causevic, hans.hansson, sasikumar.punnekkat}@mdu.se
†*ABB AB, Process Control Platform*,Västerås, Sweden, {bjorn.leander, tomas.lindstrom}@se.abb.com

***Abstract*—When developing products or performing experimental research studies, the simulation of physical or logical systems is of great importance for evaluation and verification purposes. For research-, and development-related distributed control systems, there is a need to simulate common physical environments with separate interconnected modules independently controlled, and orchestrated using standardized network communication protocols.**

The simulation environment presented in this paper is a bespoke solution precisely for these conditions, based on the Modular Automation design strategy. It allows easy configuration and combination of simple modules into complex production processes, with support for individual low-level control of modules, as well as recipe-orchestration for high-level coordination. The use of the environment is exemplified in a configuration of a modular ice-cream factory, used for cybersecurity-related research.

I. Introduction

Industrial control systems are undergoing a transformation driven by the Industry 4.0 evolution, characterized by increased connectivity and flexibility. There is a trend towards utilizing a network-centric architecture, as, e.g., described by the Open Process Automation Standard (O-PAS) [1]. Interoperability is of great importance in such systems, and using standarized, ethernet-based communication protocols, is one way of achieving such interoperability. Two such protocols are the Open Process Communication Unified Automation (OPC UA) [2], which is increasingly used in Operational Technology environments, and Message Queuing Telemetry Transport (MQTT) [3], a lightweight messaging protocol, suitable for constrained environments, e.g., for machine-to-machine communication.

Modular Automation (MA) [4], is one of the design strategies for Industry 4.0, adapted for process industries [5], [6], with the main application areas within the energy, pharmaceutical, chemical, food and beverage sectors. According to ZVEI [7], in 2030 MA is expected to contribute to a market volume of 3.5 billion euros. MA is a distributed control design strategy, which relies upon usage of interoperable protocols, using a services-based architecture. In MA, the physical process consists of a set of modules specialized in autonomously performing low-level tasks with standardized logical and physical interfaces, such that modules can be combined to fulfill high-level production tasks [8]. The execution of the high-level tasks is coordinated by an orchestrator [9] managing a high-level workflow described by a recipe. This creates a production system that is easy to retrofit and scale, as well as adapt to both changing business demands and short innovation cycles.

When conducting experiments or developing products aiming for production systems, simulation of the physical processes is a common way of understanding the benefit of the product or the validity of the conducted experiment [10]. Typically, a math-based simulation engine such as MATLAB is used to model the physical process. However, to do experiments on systems that include interconnected and individually controlled physical modules, common in e.g., modular automation, it is difficult to find a simulation engine capable to act as the physical counterpart of multiple controllers, while maintaining the physical integrity of the overall process, e.g., with regard to mass and temperature flow between modules.

Physical system simulation [11] is a wide research area, with many commercial as well as open source tools available, focusing on different objectives. However, there are few tools available that allow integration of the simulation with a controller. MATLAB contains an Industrial Communication Toolbox[1] to support integration with, e.g., the OPC UA and MQTT protocols, with the focus on enabling MATLAB functions to work as clients, thereby enabling access to industrial data using these protocols. There are examples of academic works with physics simulators having controllable I/O, e.g., Latif *et al.* [12] wraps an OPC UA server around a Modelica[2] model, in order to connect it to an industrial controller. However, we are not aware of any examples of MA system-wide simulation environments allowing control of separate modules, together with module orchestration.

- **Physical modules** - Defining the behavior and Input/Output (I/O) data for the modules.
- **Module Interconnections** - Defining how material flow between modules, such that, e.g., the output of one module can be input to another module.
- **Sensors/Actuators** - Exposing, for each module, a set of I/O that can be used to control the module behavior from an external controller.
- **System reconfiguration** - Adding and removing simulated modules, to adapt the process to changing production requirements.

[1]se.mathworks.com/products/industrial-communication.html
[2]modelica.org/index.html

- **System orchestration** - Automated high level synchronization of the process execution, by orchestration of the module controllers.

MQTT is used for detailed control, i.e. communication between simulated I/O and controller, while OPC UA is used for high-level orchestration and supervision.

The main contributions presented in this paper are:

- A simulation environment for modular production systems, following the MA design strategy.
- A description of an implementation of the simulation environment.
- An example of the use of the simulation environment for a modular ice-cream factory.

The remainder of this paper is organized as follows. Section II provides details on the implementation, and Section III contains an example of software in the loop for a modular ice-cream factory that is created using the presented simulation environment. In Section IV we discuss the impact, shortcomings, and potential future extensions. Section V concludes the paper and outlines future plans.

II. Implementation

The main goal of this work is to provide a simulation environment that can be used to demonstrate and simulate different features of a system built using the MA design strategy. The aim is to use standardized protocols and extendable techniques to allow continuous improvement and development of the simulation environment for different research purposes.

The simulation engine is not intended to provide high-fidelity simulation of a specific physical process, since there are several much more capable solutions available on the market. Instead, the purpose of the implementation is to create a re-configurable environment, able to concurrently simulate the behavior and interactions of several high-level modules that are individually controlled.

The simulation engine is developed using C# targeting the .NET Core platform. MQTT connectivity has been implemented using the M2Mqtt[3] library for .NET core, while the OPC UA support for the simplified module controllers and the orchestrator has been implemented using the .NET core OPC UA stack from the OPC foundation[4].

A. Functional overview

Fig. 1 shows an overview of the simulation engine architecture. The first step is to provide a configuration file that describes the modules to be simulated, including their physical properties, I/O, and details on how signals are interconnected (e.g., the output signal of one module may be the input signal of another one). This file is used to construct instances of all the specified modules and to populate the internal database with initial values of the simulation signals. Several tasks are started to allow concurrent execution of the physical process simulation, message handling, and the user interface.

[3]github.com/eclipse/paho.mqtt.m2mqtt
[4]github.com/OPCFoundation/UA-.NETStandard

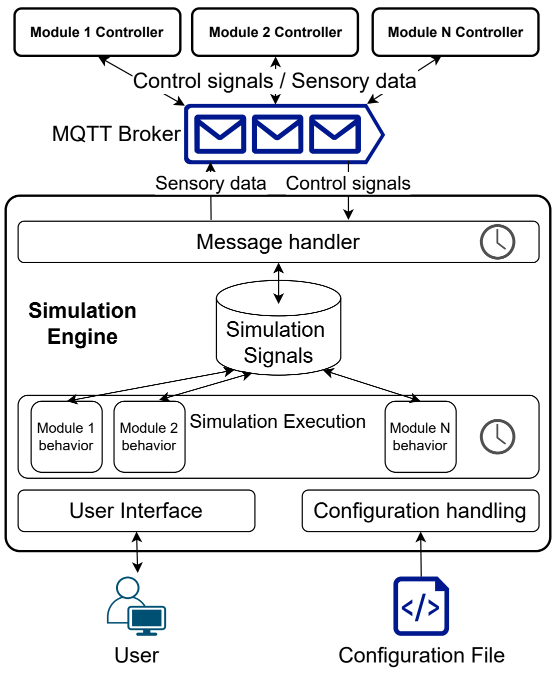


Fig. 1: Architecture overview of the simulation engine.

The simulation execution task contains a periodically executing loop. During each run, the physical behavior of each module is simulated, executing the internal module logic which reads and updates the simulated signals accordingly. Interactions between the modules are modeled by connections between the simulation signals.

The message handler is responsible for MQTT broker interactions. Signals representing sensor outputs are published and control signals are subscribed to, allowing interactions between the simulated modules and individual module controllers. Publishing to the MQTT broker is done periodically.

Modules are implemented as separate classes that simulates module behavior and contain the logic for tying signals to module properties. In principle, the simulated module behavior is implemented using discrete physics formulas, e.g., the level l' of a tank which is drained from an output aperture by gravity pressure is calculated according to:

$$l' = l + \Delta T(V_{in} - V_{out})/A, \tag{1}$$

where l is the previous level, V_{in} and V_{out} are the volume input and output flows respectively, ΔT is the length of the sample time in seconds, and A is the base-area of the tank. The volume outflow V_{out} is calculated according to [13][5]:

$$V_{out} = c_c c_v A_o \sqrt{2gl}, \tag{2}$$

where c_c is the contraction coefficient for the outlet aperture of the tank, c_v is the medium specific velocity coefficient, A_o is the outlet area, and g is the gravity acceleration constant.

[5]www.engineeringtoolbox.com/flow-liquid-water-tank-d_1753.html

B. Signals and Parameters

We distinguish between five major types of signals: Digital Input (DI), Digital Output (DO), Analog Input (AI), Analog Output (AO) and Calculated Signal (CO). The DI, DO, AI, DO types correspond to the traditional I/O signals in an automation system, representing sensors and actuators. The CO describes interconnections (e.g., when the outflow of one tank is connected to the inflow of another one), or a predefined relationship (e.g., the signal representing the level of a tank expressed in meters can be used to directly calculate the fill percentage, based on the height of the tank). We model the deviation of the signals that represent sensor readings by adding randomized noise to the actual ground-truth sensor values (sv), where the amplitude of noise for a specific sensor is available as an attribute in the configuration file. This attribute defines the sensor precision (p). Sensor readings are generated according to the uniform distribution defined on the range from $(sv - p)$ to $(sv + p)$.

The basic physical properties of the modules are described as parameters. For example, for a simple tank, the parameters are base area, height, and outlet area. These static values are part of the simulation engine configuration file.

C. User interface

Visualization of the current state of the simulated set of modules is provided as a simple graphical user interface (see Fig. 2). It contains visual representations of modules, their interconnections, and current values of the parameters. Modules are represented with rectangles, pipes with lines, and valves with circles that can be white colored (opened valve) or black colored (closed valve). The dark gray color indicates the level of material in each module. By default, only the basic parameters for each module are visible (i.e., module name, temperature, and level), but when the user selects a specific module, all parameters are shown (e.g., T1 in Fig. 2). Moreover, there is an option to view raw data in real time and to download it for the desired period (Fig. 3).

The graphical user interface is created as a Windows desktop application, using Windows Presentation Foundation (WPF) and Extensible Application Markup Language (XAML). It does not allow any direct interactions with the simulation engine, but there is a command-line interface that can be used to manually modify the values of internal simulation signals, which can be used to drive the simulation without connected controllers.

D. High-level and low-level Control

In order to allow the individual control of the modules, the simulation environment also contains implementations of a few variants of control engines.

They enable low-level control logic for the modules based on I/O signals using MQTT connectivity, and expose high-level control functions using OPC UA. These controllers run as individual processes and may be deployed on any node, as long as they can access the MQTT broker. A simple example is a tank level controller, which reads the level of the tank and controls the in-flow by a pump. The low-level control may in this case be in the form of a PID-loop, and the high-level control function may be changing the set-point for the level. This separation of low-level and high-level control allows recipe orchestration according to the MA system design strategy.

The control-part of the simulation environment is completely independent of the simulation engine, any controller that can communicate using the MQTT protocol could be used for the low-level module control, and any controller using OPC UA can be used for the high-level orchestrator control.

E. Data extraction

Logging of signal values in the simulation engine can be turned on using one of the parameters in the configuration file. If this option is selected, all the signal values will be recorded in a single CSV file. One row in the CSV file represents the current status of the simulation engine at a specific time point, including the exact date and time of each record. The logs are recorded with a configurable periodicity. Additionally, data can be extracted using different logging options for the MQTT broker, and network data can be recorded using software solutions for network monitoring (e.g. Wireshark[6]).

The collected data can be used for machine learning applications, e.g., anomaly and intrusion detection research areas.

III. An Example: The Modular Ice-cream Factory

To illustrate how the simulation engine can be configured and integrated into a larger simulator environment, this section provides an example of a simulation of a modular ice-cream factory. The factory is split into five main modules, each individually controlled and physically interconnected. An overview of the environment is depicted in Fig. 4. The simulation engine is configured to simulate behavior of five separate modules, a mixer, a pasteurizer, a homogenizator, a module handling dynamic freezing that whips air into the ice-cream mixture while refrigerating it, and a packaging module. For each of the modules, a separate module controller is implemented containing the low-level control logic for each module, as well as high-level commands, e.g., StartPasteurize, EmptyModule, etc.

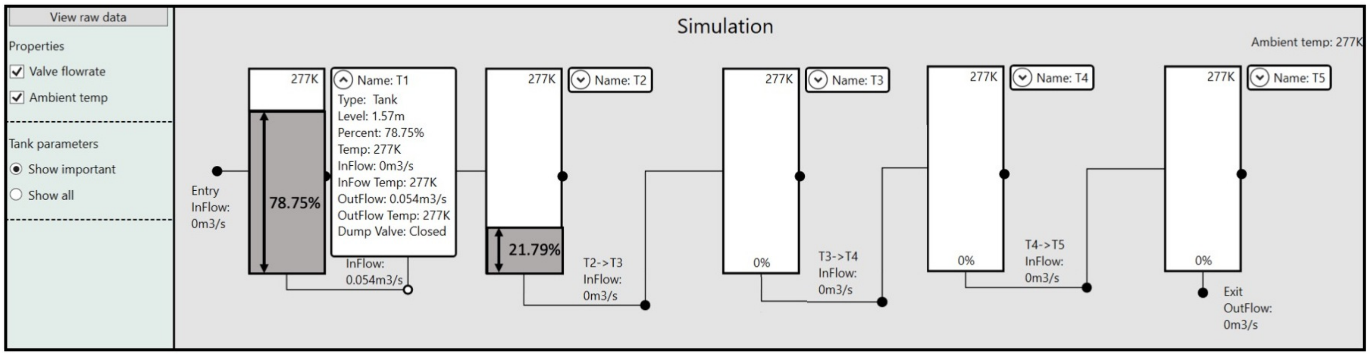
The controllers are running on separate physical nodes in the system, and have separate network connections for process orchestration, and sensor network connectivity, respectively.

The simulation environment contains one orchestrator that uses recipes defined as Sequential Function Charts (SFC) [14] to perform the high-level synchronization of production. Furthermore, there are control room functionalities (operator workplace), that allow human operators to monitor the system, and engineering tools (engineer workplace), that enable functionality for engineering of the production recipes.

The hardware (Fig. 5) and software used with the experimental setup of the simulation environment contain:

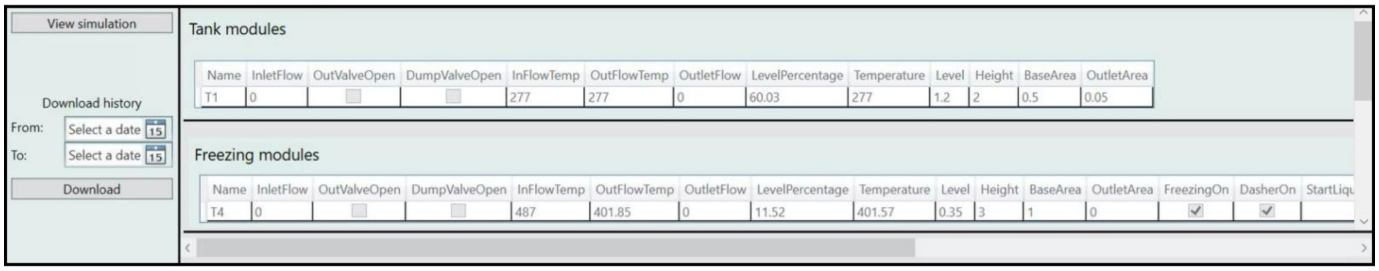
- SNL (Sensor Network Layer) - a network switch used for field communication, i.e., low level control of the process.

[6]www.wireshark.org/



Name: T1
Type: Tank
Level: 1.57m
Percent: 78.75%
Temp: 277K
InFlow: 0m3/s
InFow Temp: 277K
OutFlow: 0.054m3/s
OutFlow Temp: 277K
Dump Valve: Closed

Fig. 2: User interface that visualizes five interconnected modules of different types with all parameters shown.



Tank modules

Name	InletFlow	OutValveOpen	DumpValveOpen	InFlowTemp	OutFlowTemp	OutletFlow	LevelPercentage	Temperature	Level	Height	BaseArea	OutletArea
T1	0	☐	☐	277	277	0	60.03	277	1.2	2	0.5	0.05

Freezing modules

Name	InletFlow	OutValveOpen	DumpValveOpen	InFlowTemp	OutFlowTemp	OutletFlow	LevelPercentage	Temperature	Level	Height	BaseArea	OutletArea	FreezingOn	DasherOn	StartLiqu
T4	0	☐	☐	487	401.85	0	11.52	401.57	0.35	3	1	0	✓	✓	

Fig. 3: User interface used for raw data view and download.

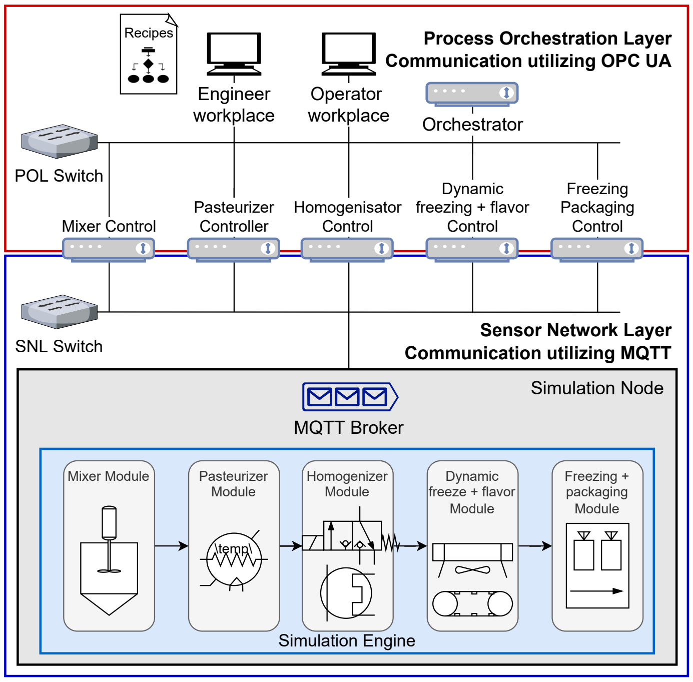


Fig. 4: Ice-cream factory overview.

- POL (Process Orchestration Layer) - a network switch used for the high level control and process supervision.
- Simulation node - a Windows 10 server connected to the SNL switch, for running the simulation engine and the Mosquitto[7] MQTT broker.
- Module controllers: A set of Raspberry PI devices, model 4, running Ubuntu server ver.16, for the module controllers, equipped with double network interfaces, with the lower side connected to the SNL switch, and the upper interface connected to the POL switch.
- The Orchestrator, executed on a Raspberry PI model 4 device, connected to the POL switch.
- Engineering and Operations, executed as a separate virtual machines running on a Windows 10 server machine, connected to the POL switch.

The purpose of this specific system is to evaluate how different types of disturbance on components of the system may influence the physical process, or even the produced product. The disturbances may be emulated by the simulator, in the case of sensor faults or noise, or injected on the network using various fuzzing tools. One example of data-collection from a sub-section of the simulation environment is visualized in Fig. 6, which shows the module levels and in-flows during two cycles of mixing and pasteurization.

IV. Discussion

In the previous sections, we have described the simulation environment and ways to instantiate the principal components of a modular automation process, using the simulation engine combined with controllers and the orchestrator. In this section, we discuss the advantages and current limitations of the developed simulation environment, present new functionalities

[7]mosquitto.org/

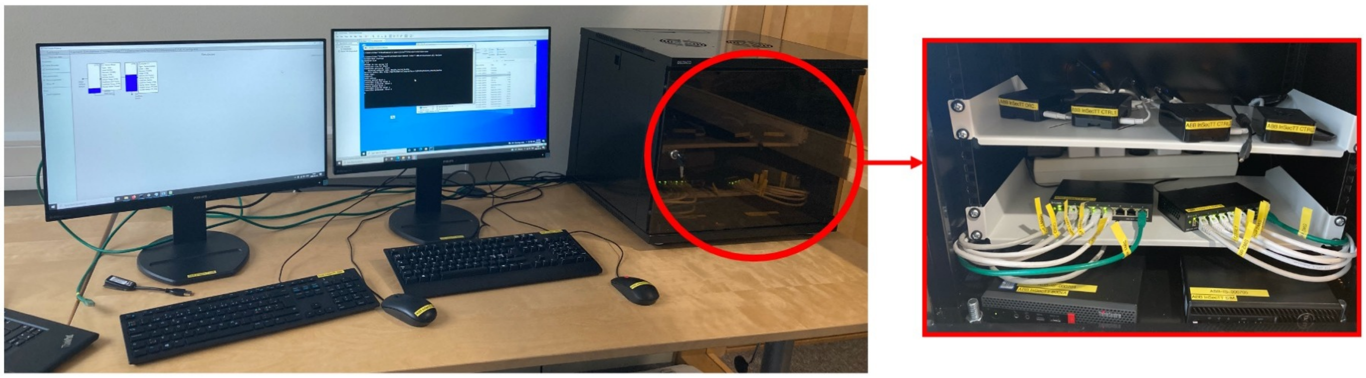


Fig. 5: Lab environment setup which includes hardware of the ice-cream factory example, with two modules simulated.

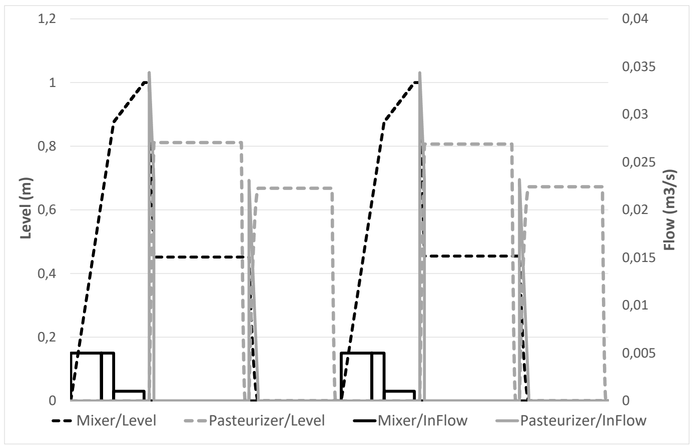


Fig. 6: Data plot from simulation of the ice-cream factory example, covering levels and flows over two cycles of mixing and pasteurization.

that will be developed in the future, and suggest the potential use of our solution.

The developed simulation engine can mimic the behavior of a fixed set of physical module types, limited by what is described in the code. Although some physical properties, as well as the interconnections between simulated modules, can be configured, the fidelity and granularity can be improved. More advanced interactions and physical behaviors, as well as additional module types, cannot be simulated without expanding the code-base of the engine, which is a clear limitation. One of the ideas for future development is to integrate the developed solution with a math-based simulator, for example Modelica, which could be used for high-fidelity and extendable module definitions.

The signals-based approach for module interactions, using only analog and digital signals, provides a loosely coupled design, which can easily be for example extended with additional module types. When the material propagates between modules, the features of the produced material require separate signals for different aspects (e.g., density, temperature, viscosity). This propagation brings compatibility issues when adding new modules, or realizing new features that must be handled in the simulation. One future improvement of the simulation engine is, therefore, to change the modeling of the material flow from analog/digital signals into complex-type signals, with a common functionality for handling propagation of different features, when mixing materials.

The simulation engine is developed using .NET Core, allowing it to be executed on different platforms. However, the user interface is limited to the Windows operating system, but we plan to explore alternatives for a cross-platform solution.

In addition, we plan to extend the simulation environment with functionalities related to different internal and external threats that can affect the normal behavior of the system. For this purpose, we plan to implement the following:

- **Anomaly simulation**, which enables users to simulate lower-level anomalies that can occur during the production process itself (e.g., anomalies in the signals of different sensors) and observe how these anomalies can affect system behavior. Data collected during the simulation with anomalies is used to create a suitable dataset for the development of a reliable anomaly detection.
- **Anomaly detection**, [15], [16] which enables generation of alerts when an anomaly occurs within the environment. Different machine learning algorithms will be evaluated on the created dataset and the ones with the best performances can be used to develop this functionality.
- **Intrusion detection** [17], [18], which enables the generation of alerts when a threat is detected at network level (i.e., different network attacks). Similar to the previous functionality, the machine learning algorithms for intrusion detection that we find most suitable shall be used.

When this type of environment is developed, it is important to have a clear vision of how it can be used and which target groups can benefit from it. To some extent, the potential use of the simulation environment overlaps with use-cases for traditional physics simulators. The major difference is that this environment is developed with a focus on the MA design strategy and network connectivity. On the other hand, easy integration of external controllers is typically not part of physics simulators. The developed simulation environment can be used for several different purposes, including the following:

- **Research** - The simulation of a realistic system can be used to conduct research in different research areas, including cybersecurity for manufacturing systems, such as the development and evaluation of access control mechanisms [19] and AI-based anomaly detection. For example, in the system described in Section III, anomalies can be injected into different layers, and different cyberattacks can be simulated to assess the impact on the physical process and to evaluate the effectiveness of the implemented mechanisms.
- **Education** - We see education as one of the main areas that can benefit from the developed simulation environment. Students in computer science can use it in areas ranging from control theory to artificial intelligence.
- **Product Development** - When developing products or creating prototypes for components and services for control systems, easy integration with a test environment is important for verification and evaluation purposes. The simulation environment can be used for simulation at system level, with a small footprint and cost. This makes it a useful tool for prototyping and product development related to control, operations, engineering, specifically for software products in the modular automation area.
- **Other industrial use cases** - Virtual commissioning [20] and operator training are two areas for which the simulation environment can be useful, thanks to its ability to interact and integrate with controllers and orchestrators.

V. Conclusions

In this paper, we present a simulation environment that can be used to simulate the physical process of a modular automation system with the ability to individually control the physical modules. The implementation uses the MQTT protocol for interactions between controllers and the simulation engine, and the connectivity in the orchestration layer is using OPC UA. The described solution is following the distributed control design strategy mandated by modular automation, and is highly configurable, allowing for any combination of implemented modules being simulated. It can be used in product development, research, and education.

The feasibility of the simulation environment for use as a part of an automation system test-bed is illustrated by the implementation of a modular ice-cream factory simulator.

As the future work, we aim to integrate the developed environment with a math-based simulator, such as MATLAB, and provide a user interface that can be used on different operating systems. We are also currently developing a set of functionalities related to anomaly and intrusion detection to enable cybersecurity related simulations and research.

Acknowledgements

Andreas Ununger, Abhinav Sasikumar and Gloria Ninsiima made technical contributions to the implementation, together with students in the Distributed Software Development course at Mälardalen University, fall semester 2021.

This work is supported by ABB AB; the industrial postgraduate school Automation Region Research Academy (ARRAY), funded by The Knowledge Foundation; and the Horizon 2020 project InSecTT. InSecTT (www.insectt.eu) has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876038. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey.[8]

References

[1] "O-PAS Standard, Version 2.0: Part 1 – Technical Architecture Overview," Open Group Preliminary Standard (P201-1), The Open Group, February 2020.

[2] "IEC 62541 OPC unified architecture, rev 1.05," standard, International Electrotechnical Commission, Geneva, CH.

[3] "MQTT Version 5.0," OASIS Standard, March 2019. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta.

[4] NAMUR Working Group 1.12, "NE 148 Automation Requirements relating to Modularisation of Process Plants," NAMUR-recommendation, 2013.

[5] J. Ladiges *et al.*, "Integration of modular process units into process control systems," *IEEE Transactions on Industry Applications*, vol. 54, pp. 1870–1880, March 2018.

[6] ZVEI—German Electrical and Electronic Manufacturers' Association, "Module-based production in the process industry—effects on automation in the "Industrie 4.0" environment," White Paper, March 2015.

[7] ZVEI—German Electrical and Electronic Manufacturers' Association, "Process industrie 4.0: The age of modular production," White Paper, August 2019.

[8] M. Hoernicke *et al.*, "Automation architecture and engineering for modular process plants - Approach and industrial pilot application," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 8255–8260, 2020.

[9] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, pp. 46–52, Oct 2003.

[10] S. M. Sanchez and H. Wan, "Work smarter, not harder: A tutorial on designing and conducting simulation experiments," *Proceedings - Winter Simulation Conference*, vol. 2016-February, pp. 1795–1809, 2016.

[11] L. G. Birta and G. Arbez, *Modelling and Simulation*. Simulation Foundations, Methods and Applications, Cham: Springer International Publishing, 3rd ed., 2019.

[12] H. Latif, G. Shao, and B. Starly, "Integrating a dynamic simulator and advanced process control using the opc-ua standard," *Procedia Manufacturing*, vol. 34, pp. 813–819, 2019.

[13] A. L. Gerhart, J. I. Hochstein, and P. M. Gerhart, *Munson, Young and Okiishi's Fundamentals of Fluid Mechanics*. John Wiley & Sons, 2020.

[14] "IEC 61131-3:2013 Programmable Controllers - Part 3: Programming Languages," standard, IEC, 2013.

[15] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[16] K. Zope *et al.*, "Anomaly detection and diagnosis in manufacturing systems: A comparative study of statistical, machine learning and deep learning techniques," in *Annu. Conf. PHM Soc*, vol. 11, 2019.

[17] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[18] M. Leon, T. Markovic, and S. Punnekkat, "Comparative Evaluation of Machine Learning Algorithms for Network Intrusion Detection and Attack Classification," in *2022 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022.

[19] B. Leander, A. Čaušević, H. Hansson, and T. Lindström, "Toward an ideal access control strategy for industry 4.0 manufacturing systems," *IEEE Access*, vol. 9, pp. 114037–114050, 2021.

[20] T. Lechler, E. Fischer, M. Metzner, A. Mayr, and J. Franke, "Virtual commissioning – Scientific review and exploratory use cases in advanced production systems," *Procedia CIRP*, vol. 81, pp. 1125–1130, 2019.

[8]The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.