SoftwareX 34 (2026) 102560

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

FMUiL: An open-source package for in-the-loop simulations with functional mock-up units

K. Klemets [a,*], D. Bouzoulas [b,c], M. Manngård [b], J.M. Böling [a]

[a] Automation Engineering, Mechanical and Materials Engineering, Department of Technology, University of Turku, Turku, Finland
[b] Automation and Maritime Simulation, Faculty of Technology and Seafaring, Novia University of Applied Sciences, Turku, Finland
[c] Department of Energy and Mechanical Engineering, Aalto University, Espoo, Finland

ARTICLE INFO

Keywords:
FMI
FMU
OPC UA
HiL
MiL
SiL

ABSTRACT

Functional Mock-up Units in the Loop (FMUiL) is an open-source Python package designed to support virtual commissioning and design phases of projects. It combines the FMI and OPC UA standards to enable the co-simulation between simulation models, software, and hardware. Since the FMI standard does not specify how co-simulation between FMUs and external systems should be handled, `FMUiL` uses the OPC UA communication protocol, which is widely adopted in the process automation industry, to manage the data exchange between devices and systems. Simulation models are wrapped as OPC UA servers that can be connected to external servers. To simplify experiment setup and ensure reproducibility, `FMUiL` uses YAML configuration files to define simulation scenarios and experiments, which can then be executed through the provided command-line interface. This paper documents the architecture and functionality of the `FMUiL` package, describes how experiments are configured, and demonstrates its use through practical examples.

Metadata

Code metadata.

Nr.	Code metadata description	Metadata
C1	Current code version	0.2.1
C2	Permanent link to code/repository used for this code version	https://github.com/Novia-RDI-Seafaring/FMUiL
C3	Permanent link to Reproducible Capsule	https://github.com/Novia-RDI-Seafaring/FMUiL-SoftwareX/
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, FMI, OPC UA
C7	Compilation requirements, operating environments & dependencies	asyncua >= 1.1.8, colorama >= 0.4.6, fmpy >= 0.3.26, numpy >= 2.3.3, pydantic >= 2.11.7, pywin32 >= 311, pyyaml >= 6.0.2, typer >= 0.19.2, typing-extensions >= 4.15.0
C8	If available Link to developer documentation/manual	https://github.com/Novia-RDI-Seafaring/FMUiL/blob/main/README.md
C9	Support email for questions	kristian.klemets@utu.fi

* Corresponding author.
Email address: kristian.klemets@utu.fi (K. Klemets).

https://doi.org/10.1016/j.softx.2026.102560
Received 31 October 2025; Received in revised form 19 January 2026; Accepted 9 February 2026
Available online 19 February 2026

1. Motivation and significance

The FMUiL Python package [1] is designed to support the use of simulation tools in the design and commissioning of systems, enabling earlier validation and facilitating integration testing across hardware and software components. It achieves this by combining two open standards: the Functional Mock-up Interface (FMI) for model exchange and co-simulation [2], and the Open Platform Communications Unified Architecture (OPC UA) for reliable data exchange in industrial automation systems [3].

The commissioning phase of a project is the final stage, where all systems and components are tested, verified, and validated to ensure they operate according to design specifications and meet the project's performance requirements before being handed over to the client or end-user. Since commissioning occurs late in the project lifecycle, any issues or faults identified during this stage can lead to costly delays and disruptions. In large-scale systems integration projects, where numerous subsystems and organizations are involved, the commissioning process becomes even more complex. Virtual commissioning (VC) techniques [4–6] have been proposed to address these challenges. VC uses simulation models of plants and automation systems to perform parts of the testing earlier in the project, thereby reducing the risks, time, and costs associated with physical commissioning [4].

However, VC of large-scale systems introduces its own set of challenges, particularly interoperability issues caused by the use of different simulation tools, data formats, and communication interfaces across suppliers (see [7] and references therein). The FMI standard [2] offers a partial solution by encapsulating simulation models into Functional Mock-up Units (FMUs). These self-contained files can be exchanged and co-simulated across different tools while protecting intellectual property by exposing only the inputs, outputs, and tunable model parameters [2,8]. The Open Simulation Platform (OSP) is one example of a platform developed for co-simulating ship systems using FMUs [9]. Nevertheless, the FMI standard alone is limited to co-simulation with components that can be packaged as FMUs.

In practice, VC requires co-simulation between models and components that cannot be encapsulated within FMUs. These simulation configurations are collectively referred to as X-in-the-Loop (XiL) and include Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), and Hardware-in-the-Loop (HiL) simulations. Reliable data exchange in XiL simulations requires standardized communication protocols [7]. With FMI 3.0, the layered standards (FMI-LS) were introduced to support specific co-simulation use cases, initially focusing on automotive protocols such as CAN and LIN [10]. However, the layered standards could be extended to other protocols used in the automation industry in the future.

In this work, we use OPC UA as the communication protocol to bridge FMUs, hardware, and software. OPC UA [3] is an open standard for secure, reliable, and platform-independent data exchange between devices and systems [3,11]. Its widespread adoption in the process automation industry and alignment with Industry 4.0 principles [11] make it well-suited as the communication backbone for the FMUiL package.

Several frameworks for co-simulation leveraging either OPC UA, FMI, or both have been introduced in recent years. Studies such as [12,13] focus on OPC UA-based communication between different simulators, whereas [14] extends this approach to HiL and SiL simulations with Programmable Logic Controllers (PLCs). In [15], Liu et al. present a HiL co-simulation environment for multi-modal energy systems based on the FMI and the MQTT protocol, while [16] explores the use of FMI for conducting full-system simulations in the maritime sector. Further contributions [17–19] demonstrate the feasibility of combining FMI and OPC UA, highlighting their applicability across various domains. Recent research emphasizes the importance of open software frameworks that can interact with off-the-shelf platforms and simplify co-simulation by making FMIs easy to configure and use. Notable examples of such frameworks are MultiCoSim [20] and Ecos [21].

FMUiL is an open-source software package designed to facilitate XiL simulations using FMUs. Unlike previous approaches that combine FMU simulation with OPC UA devices for specific use cases, FMUiL was developed as a general-purpose, reusable, and open-source solution for bridging FMUs, hardware, and software via OPC UA. It can be used to validate control systems or parameters in various industrial applications, perform rapid prototyping of system designs by combining components developed with different modeling tools, or support hardware-in-the-loop and virtual commissioning workflows, allowing engineers and researchers to quickly assess system performance, identify integration issues, and test control strategies without the need for full-scale physical prototypes. The main benefits of FMUiL are:

- Easy integration with external models and hardware platforms that support the OPC UA protocol.
- Human-readable configuration of servers, simulation scenarios, and experiments using YAML files.
- A command-line interface (CLI) for executing simulation workflows.

This flexibility lowers the barrier to adopting FMI-based co-simulation in complex industrial environments, enabling efficient testing of interconnected systems and control strategies before deployment.

2. Software description

The FMUiL package allows users to connect FMU simulation models to OPC UA servers and define simulation scenarios and experiments. It allows users to define how data flows between components by specifying input–output connections, configure initial values and parameters, define evaluation criteria, record output data, and execute simulations. The simulation setup is specified in a human-readable format using experiment files written in the YAML data serialization language (`.yaml`).

FMUiL is implemented in Python, using the FMPy [22] and asyncua [23] packages as its main dependencies. FMPy offers a standardized and efficient way to run FMUs according to the FMI specification, and asyncua enables asynchronous and standards-compliant OPC UA communication services.

The software has been released under the MIT license, and the version presented in this article has been tested and validated on Windows 10 and Windows 11. The current version supports FMI 2.0, but support for FMI 3.0 is planned for future development.

2.1. Software functionalities

The main purpose of FMUiL is to set up and orchestrate XiL simulations. It does this by wrapping user-provided FMUs in OPC UA servers and creating the necessary clients to communicate between these servers and any external OPC UA systems. In this context, an internal or simulation server refers to an OPC UA server created by FMUiL to run an FMU instance. An external OPC UA server refers to an existing server that functions independently without FMUiL's control.

During simulation initialization, FMUiL sets up the servers and establishes OPC UA client connections to facilitate communication between them. For each FMU, an OPC UA server is instantiated based on a standardized template. This template defines the interface for advancing FMU simulations, synchronizing nodes and FMU variables within the server, and resetting FMUs to their initial states. For external OPC UA servers, FMUiL establishes client connections capable of reading from and writing to the specified server nodes.

A new experiment begins with the initialization of the specified initial conditions, which are propagated to the servers and subsequently to the FMUs. This is achieved by mapping the FMU inputs, outputs, and parameters to corresponding OPC UA nodes. Once this mapping is complete, the predefined connections between variables are established, and the simulation environment is ready for execution. The interaction between FMUiL, OPC UA servers, and user-provided configuration files and models is illustrated in Fig. 1.

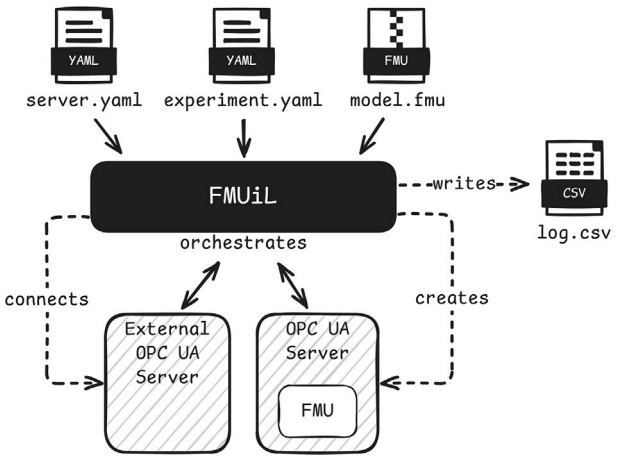


Fig. 1. Interaction between the user and the FMUiL system. The user defines experiment configurations, which the system then executes by creating internal OPC UA servers, connecting to external OPC UA servers, and orchestrating the simulation workflow.

The simulation progresses in cycles. During each cycle, all FMUs are stepped forward according to the defined timestep and simulated until the next communication point is reached. Each FMU may use its own internal timestep, but the communication timestep must be an integer multiple of each FMU's timestep to maintain accurate time progression. At the communication point, data are exchanged between connected servers, which then update the FMUs accordingly. Fig. 2 shows the main sequence of a simulation cycle. During the first cycle, the initial values from the configuration are written to the server. The simulation cycle begins by stepping all FMUs until the communication timestep is reached. After the simulation step, all variables defined in the configuration connections are updated accordingly. The global simulation time is then advanced to synchronize with the FMUs, and finally, values are logged or evaluated before the next cycle begins.

Evaluation and logging are optional, separate features in the FMUiL. When evaluation is enabled, it checks whether the defined conditions are satisfied, while logging captures the current values of variables. These operations produce their own logs and are performed before the simulation is stepped forward.

2.2. Software architecture

FMUiL is the main package and is organized into five subpackages as illustrated in Fig. 3. The functionality of the subpackages can be summarized as follows:

- `handlers`: Contains three modules that together form the coordination layer of FMUiL. The `config_handler` module manages experiment definitions; the `fmu_handler` defines mechanisms for setting up and initializing FMU models (including the arrangement of their inputs, outputs, and parameters as specified in the FMU's model description file); and the `simulation_handler` serves as the central orchestration component of the architecture, coordinating the overall simulation workflow and managing data exchange between instantiated servers.
- `communications`: Manages OPC UA–based data exchange within FMUiL, creating managers for servers and clients and establishing their relationships. The `server_setup` module defines a standard OPC UA server template for internal servers, providing essential methods to simulate and update FMUs.
- `logger`: Manages functions related to logging simulation results. The `experiment_logger` module within this subpackage is responsible for data recording, including both system evaluation results and variable logging.
- `schemas`: Contains `Pydantic` models [24] used for validating configuration files.
- `utils`: Provides auxiliary functions that support internal operations.

In addition, a command-line interface (CLI) is provided for executing experiments. Alongside the FMUiL package, users define an `experiments` folder that contains the experiment configuration files and the FMUs describing the simulation setups.

2.3. Configuration

The simulation configuration in FMUiL is defined using YAML [25] experiment files. Each experiment configuration file specifies parameters for setting up the simulation scenario, experiments, and logging of results. The experiment configuration file contains the following sections:

- `fmu_files`: List of paths to the FMU models used in the experiment.
- `external_servers`: List of paths to the external server configuration files referenced in the experiment.
- `experiment`: This section defines the simulation, evaluation, and logging parameters and contains the following fields:
  - `experiment_name`: Defines the name of the experiment.
  - `timestep`: Defines the communication interval for the experiment, determining how often information is exchanged.
  - `timing`: Defines the simulation timing mode. `real_time` synchronizes the simulations against a real-world clock, whereas `simulation_time`-mode simulates as fast as possible.
  - `stop_time`: Defines the simulation stop time (in seconds).
  - `initial_system_state`: Specifies the FMU's internal `timestep`, initial input values, and specific parameter settings.
  - `system_loop`: Connections and data flow between internal (FMU) and external servers.
  - `evaluation`: Defines evaluation criteria for the experiment.
  - `logging`: Defines which signals are to be logged.

More details on how to configure an experiment are provided in the example configuration file in Appendix A, Listing 1.

In addition, if the user intends to integrate external OPC UA servers into the simulation, a separate configuration file must be provided. This file specifies the server URL and, for each variable to be accessed, either the numeric node identifier (`id`) with its namespace (`ns`), or the string-based node identifier (`name`). The configuration is defined in a separate YAML file. An example configuration file for an OPC UA server is provided in Appendix A, Listing 2.

When configuration files are loaded, they are validated using `Pydantic` models [24], based on predefined schemas for experiments and servers, which are illustrated in Fig. 4a and b, respectively.

3. Illustrative examples

To demonstrate the capabilities of the FMUiL package, two simulation scenarios are presented: (i) a MiL experiment involving two interconnected FMUs, and (ii) a HiL experiment that co-simulates an FMU with a Siemens S7-1500 PLC. Both scenarios are based on a water-tank process where the water level is controlled. The plant model is adapted from the MATLAB/Simulink water tank example [26], and has been divided into two FMUs: `TankLevelPI.fmu`, which implements the PI controller, and `WaterTankSystem.fmu`, which represents the tank dynamics. The system is illustrated in Fig. 5. The water tank is described by

$$A\frac{dH}{dt} = bV - a\sqrt{H}, \tag{1}$$

where H is the tank water level, A is the cross-sectional area of the tank, V is the pump voltage, and a and b are parameters related to the flows.

The water level is controlled with a PI controller. The controller receives the desired water level `SP_WaterLevel` (setpoint) and the

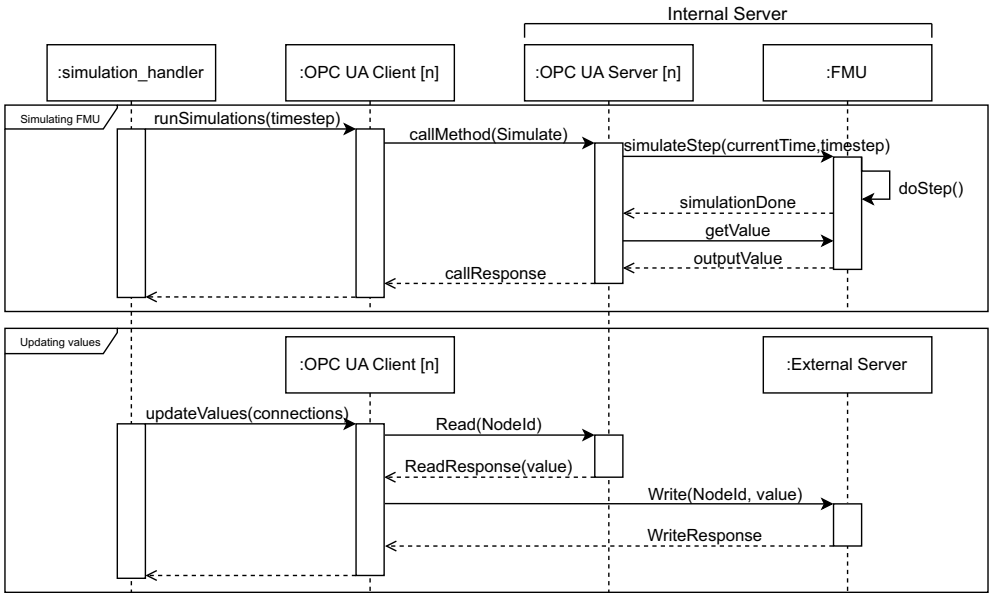


Fig. 2. System sequence diagram illustrating both a single FMU simulation and the update of values from an internal server to an external server as separate sequences. Updates on the internal server follow the same mechanism. Each FMU in the configured simulation is executed in turn, and values are updated for every configured variable connection.

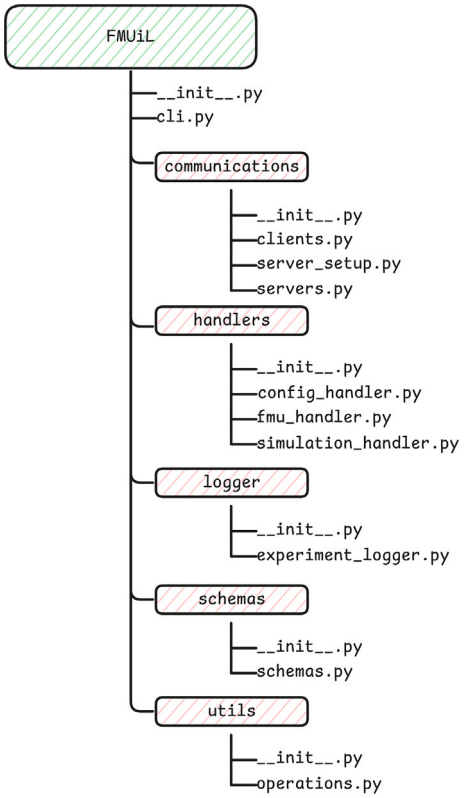


Fig. 3. Subpackages and modules of the `FMUiL` package.

measured water level `PV_WaterLevel` (process value) as inputs, and it outputs the control signal `CV_PumpCtrl`, which indirectly adjusts the inlet water flow rate by changing the input voltage to the pump.

The outcomes of both in-the-loop experiments are compared against the baseline results obtained from the standalone MATLAB/Simulink simulation of the complete system.

3.1. MiL setup: coupled FMUs

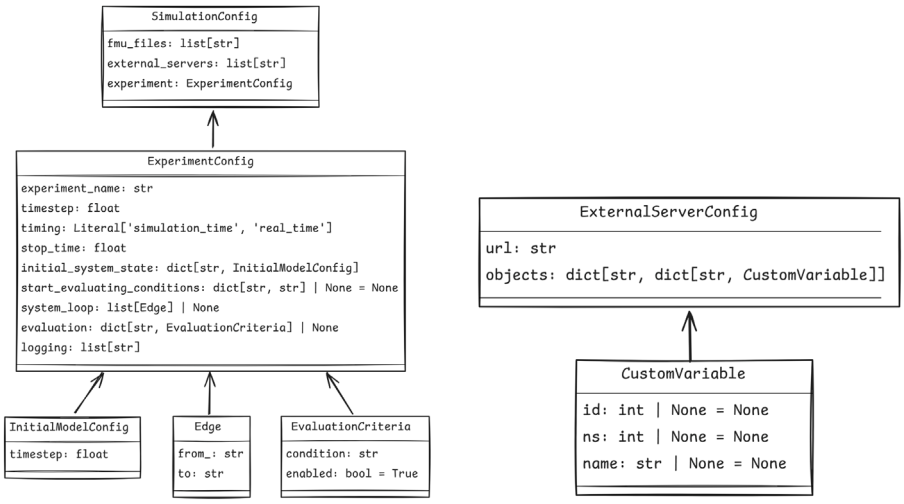
To set up the MiL experiment with the two interconnected FMUs, an experiment configuration file was created. Paths to the `TankLevel_PI.fmu` and `WaterTankSystem.fmu` FMUs were provided. No external servers were used in this experiment. The FMU time steps were set to 0.1 s for the `WaterTankSystem` and 0.1 s for the `TankLevel_PI` controller. For the `WaterTankSystem`, the default model parameters $A = 20$, $b = 5$, and $a = 2$ were used, while the PI-controller was configured with $K_p = 1.5$ and $K_i = 0.5$ (corresponding to an integration time $T_i = 3$ seconds). The controller setpoint (desired water level) was set to 10 meters. The input–output connections between the FMUs were specified as illustrated in the block diagram in Fig. 5.

Evaluation criteria for monitoring the water level and control signal limits were defined. Evaluations were set to be performed as long as there was water in the tank, and an evaluation criterion was set to raise an alarm whenever the water level exceeded 11.0 meters or the control signal became negative. If an output surpassed a defined limit, the evaluation status was recorded as `True` in the log files. The complete experiment configuration is presented in Appendix B, Listing 3.

3.2. HiL setup: FMU controlled with hardware PLC

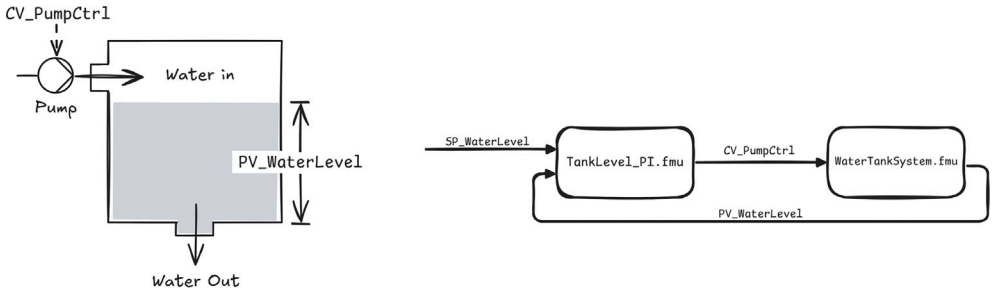
The HiL experiment is identical to the MiL setup, except that the control logic is executed on a physical PLC rather than as an FMU. This introduces more realistic hardware and communication constraints to the co-simulation. For PLC integration, the PLC must first be set up as an OPC UA server, and its interface needs to be properly defined. For the Siemens S7-1500 series, this setup is simple using the Siemens TIA Portal platform. After configuration, the PLC can be connected with `FMUiL` through a server description file that specifies the server's address, port, and the nodes involved in the simulation (see Section 2.3). An example of this configuration is shown in Appendix C, Listing 4.

To enable the PLC to operate correctly alongside the FMU, several additional modifications are required to the experiment file. Since the controller functions as a real-time system, the `timing` parameter must be set to `real_time`. Furthermore, the `system_loop` configuration needs to be updated to reference the external OPC UA nodes defined in Listing 4, replacing the variables of the controller FMU. The complete configuration is presented in Appendix C, Listing 5.



(a) Experiment configuration schema. (b) External server configuration schema.

Fig. 4. Class diagrams of `FMUiL` configuration schemas. (a) shows the experiment-configuration schema, and (b) shows the external server configuration schema.



(a) Water tank system. (b) Block diagram showing FMU connections.

Fig. 5. Illustration of the controlled water tank system. The water level, `PV_WaterLevel`, is controlled by a PI-controller that adjusts the voltage supplied to the pump. Water enters the tank proportionally to the control signal `CV_PumpCtrl`. (a) shows the physical system, and (b) presents the block diagram of how the FMUs are connected.

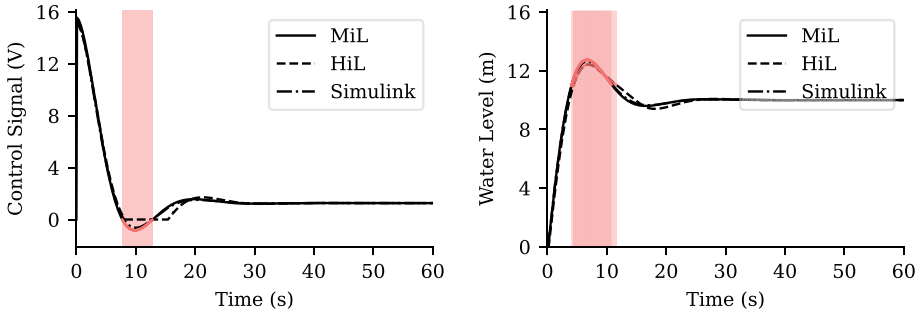


Fig. 6. Simulation results for the MiL and HiL experiments. Simulation results from the reference MATLAB/Simulink model are also included as a reference. The red-shaded regions indicate periods when the evaluation criteria were violated. In this case, negative control signals and water levels exceeding 11.0 meters triggered alarms. *Left:* Control signals (V) for the experiments. *Right:* Water levels (m) for the experiments. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.3. Simulation results

Simulation results for the MiL and HiL experiments, including the reference simulation performed in MATLAB/Simulink [26], are presented in Fig. 6. The figures were generated from the log files corresponding to each experiment. Time intervals during which evaluation criteria were satisfied—such as when the control signal became negative or the water level exceeded 11.0 meters—are also highlighted.

As the MiL experiment conducted with `FMUiL` closely replicates the original MATLAB/Simulink model, only minor deviations were expected. These differences primarily arise from variations in communication handling and initialization procedures between the two simulation platforms. Quantitative metrics used to evaluate the accuracy of the `FMUiL` MiL simulation are summarized in Table 1, including the root mean square error (RMSE), mean absolute error (MAE), and coefficient of determination (R^2) between the MiL results and the MATLAB/Simulink reference.

In contrast, the HiL experiment shows a more noticeable deviation due to hardware constraints imposed by the PLC controller. In particular,

Table 1
Quantitative comparison between MiL simulations in FMUiL and the source system in MATLAB/Simulink.

Variable	RMSE	MAE	R^2
Water level (m)	0.0704	0.1168	0.9996
Control signal (V)	0.1953	0.2830	0.9988

Table 2
Timing comparison of MiL and HiL simulations in FMUiL. Rows show average, standard deviation, maximum, and 95th percentile (ms) for FMU execution, variable updates (two per cycle), and full simulation cycles. HiL durations are higher due to interactions with external hardware.

Simulation	Event	Average (ms)	Standard deviation (ms)	Max (ms)	P95 (ms)
MiL	Simulate FMU (Watertank)	0.882	0.375	2.491	1.713
MiL	Simulate FMU (PI Control)	0.624	0.250	1.599	1.270
MiL	Update variable	0.882	0.326	2.499	1.583
MiL	Full Simulation cycle	6.002	1.955	14.819	11.104
HiL	Simulate FMU (Watertank)	1.616	0.734	3.817	3.030
HiL	Update variable	2.359	1.405	15.035	3.844
HiL	Full Simulation cycle	9.771	3.369	32.907	15.522

the control signal (voltage) was limited to positive values, which affected overall control performance. These examples clearly demonstrate the value of performing virtual commissioning using both MiL and HiL configurations to uncover such implementation-specific limitations early in the development process.

The simulations were performed on a Windows 10 laptop with a 13th Gen Intel® Core™ i7-1355U processor (1.70 GHz base frequency). Latency and cycle time measurements are reported in milliseconds, with average, standard deviation, maximum value, and the 95th percentile (P95) included to illustrate variability and outlier behavior. The results for both simulations are summarized in Table 2.

In this context, an “update variable” refers to a single variable update call. Both simulations involved two connections, resulting in two variable update calls per cycle. The observed differences between the MiL and HiL simulations are primarily due to the HiL setup being physically connected to external hardware, whereas the MiL simulation runs entirely on the local machine.

4. Impact

When conducting system-level simulations, it is common for individual subsystem models to be developed using domain-specific tools, many of which support exporting models as FMUs. To enable the simulation of complete systems, these FMUs must be interconnected and orchestrated. Although the FMI and System Structure and Parameterization (SSP) standards provide formal specifications for defining hierarchical system structures and parameter configurations, there is currently limited availability of open-source tools that fully support the simulation of SSP-defined or interconnected FMU systems. Furthermore, integrating external hardware and software into the simulation loop is not readily possible with existing standards and tools.

To address these gaps, FMUiL was developed as a lightweight, open-source framework designed to connect and execute simulations involving multiple interacting FMUs efficiently. Unlike many existing FMI-based frameworks, which are often complex, heavyweight, tightly coupled to specific simulation environments, or tailored to a particular domain, FMUiL is intentionally lightweight, easy to use, and general-purpose. The user only needs to provide the FMUs and possibly hardware, and then configure the simulation setup. This enables rapid prototyping and testing of system designs, interconnections, and control algorithms, while supporting interconnected system components from different tools or suppliers.

FMUiL uses the OPC UA standard to communicate and facilitate seamless integration of both hardware and software components within the simulation environment. Many modern simulation tools and industrial-grade hardware systems natively support OPC UA communication, enabling straightforward interoperability with FMUiL. This design choice enables the framework to incorporate real hardware-in-the-loop (HiL) components and external control systems, thereby supporting co-simulation scenarios that better reflect real-world system behavior.

By combining interoperable FMU-based co-simulation with standardized OPC UA communication, FMUiL provides a flexible foundation for research in cyber-physical system integration, digital twin development, and virtual commissioning. The framework enables rapid prototyping and evaluation of hybrid systems that include simulated and physical components, allowing researchers to investigate real-time interactions, assess control robustness, and explore system-level behaviors under realistic communication constraints. Furthermore, the open and modular architecture of FMUiL facilitates reproducible experimentation and the extension of the framework to emerging domains such as energy systems, industrial automation, and autonomous systems.

5. Conclusions

This work presents FMUiL, an open-source Python package, designed to facilitate FMU based in-the-loop and co-simulations through standardized OPC UA communication. The paper details the software functionalities and architecture, key components, and execution workflow. Two illustrative examples demonstrate its capability to connect and orchestrate both simulated and real control systems, highlighting the ease of coupling FMUs and integrating industrial hardware. By bridging FMI-based models and OPC UA-enabled devices, FMUiL supports reproducible research on cyber-physical systems, facilitates digital twin prototyping, and offers a flexible environment for testing across virtual and physical domains.

The software repository for FMUiL is jointly maintained by Novia University of Applied Sciences and the University of Turku and is expected to be continuously updated throughout the Virtual Sea Trial project. Future releases of the package are planned to include Python APIs for configuring and executing simulations, bridge servers for integrating hardware and software components, and a graphical user interface. Long-term development will focus on extending FMUiL into an intelligent simulation framework that incorporates AI agents and automated workflows for configuring simulation scenarios and experiments, as well as generating reports.

CRediT authorship contribution statement

K. Klemets: Writing – original draft, Validation, Software, Methodology, Conceptualization. **D. Bouzoulas:** Software, Methodology, Conceptualization. **M. Manngård:** Writing – review & editing, Supervision, Software, Project administration, Conceptualization. **J.M. Böling:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research was funded by Business Finland under Grant 7316/31/2023 “Virtual Sea Trial”.

Appendix A. Example configuration files

```yaml
fmu_files:  [
    "path01 to your fmu",
    "path02 to your fmu"
    ]

external_servers: [
                  "path01 to server description",
                  "path02 to server description"
                  ]

experiment:
  experiment_name: "a unique test name"
  timestep: 0.5
  timing:  "simulation_time" or "real_time"
  stop_time: 100.0
  test_description: "a description of the test"

  initial_system_state:
    FMU_Model_Name:
      FMU_input_name1: "desired initial value"
      FMU_input_name2: "desired initial value"
      FMU_parameter_name1: "desired initial value"
      FMU_parameter_name2: "desired initial value"

    example_server:
      opcua_object01_variable1: "desired initial value"

  start_readings_conditions:
    condition_name: "FMU_Model_Name.FMU_input_name1 > 10"

  system_loop:
    - from: source_object.source_variable
      to:   target_object.target_variable

    - from: source_object.source_variable
      to:   target_object.target_variable

  evaluation:
    eval_1:
      condition: "fmu1.value < 11.1"
      enabled: true
    eval_2:
      condition: "fmu2.value < 11.1"
      enabled: false

  logging:
    ["fmu.variable", "server.variable"]
```

Listing 1. Example of an experiment configuration file.

```yaml
  url: opc.tcp://localhost:4840/opcua/server/ # server url

  # object definition
  objects:
    object_name_01:
      variable_name_01:
        id: 4
        ns: 5

      variable_name_02:
        name: "variable_02"
```

Listing 2. Example of a server configuration file.

Appendix B. Configuration file for the MiL example

```yaml
fmu_files: ["experiments/fmus/WaterTankSystemV2.fmu",
            "experiments/fmus/TankLevel_PI.fmu"]

external_servers: []

experiment:
  experiment_name: Water Level Control
  timestep: 0.1
  timing: "simulation_time"
  stop_time: 60

  initial_system_state:
    WaterTankSystem:
      timestep: 0.1
      A: 20
      b: 5
      a: 2
      CV_PumpCtrl_in: 15

    TankLevel_PI:
      timestep: 0.1
      SP_WaterLevel_in: 10
      Ki: 0.5
      Kp: 1.5

  # The system loop is made according to the block diagram
  system_loop:
    - from: TankLevel_PI.CV_PumpCtrl_out
      to:   WaterTankSystem.CV_PumpCtrl_in

    - from: WaterTankSystem.PV_WaterLevel_out
      to:   TankLevel_PI.PV_WaterLevel_in

  evaluation:
    eval_1:
      condition: "WaterTankSystem.PV_WaterLevel_out > 11.0"
      enabled: true
    eval_2:
      condition: "TankLevel_PI.CV_PumpCtrl_out < 0.0"
      enabled: true

  logging:
    ["WaterTankSystem.PV_WaterLevel_out", "TankLevel_PI.
    CV_PumpCtrl_out"]
```

Listing 3. Configuration file for the MiL experiment.

Appendix C. Configuration files for the HiL example

```yaml
# File: S7-1500.yaml
url: opc.tcp://192.168.0.1:4840

objects:
  sample_opc_object:
    PV_WaterLevel_in:
      ns: 4
      id: 5

    CV_PumpCtrl_out:
      ns: 4
      id: 6
```

Listing 4. Configuration file for the external OPC UA server used in the HiL experiment.

```yaml
fmu_files: ["experiments/fmus/WaterTankSystemV2.fmu"]

external_servers: ["experiments/servers/S7-1500.yaml"]

experiment:
  experiment_name: Water Level Control
  timestep: 0.1
  timing: "real_time"
  stop_time: 60

  initial_system_state:
    WaterTankSystem:
      timestep: 0.1
      A: 20
      b: 5
      a: 2

  # The system loop is made according to the block diagram
  system_loop:
    - from: S7-1500.CV_PumpCtrl_out
      to:   WaterTankSystem.CV_PumpCtrl_in

    - from: WaterTankSystem.PV_WaterLevel_out
      to:   S7-1500.PV_WaterLevel_in

  evaluation:
    eval_1:
      condition: "WaterTankSystem.PV_WaterLevel_out > 11.0"
      enabled: true

  logging:
    ["WaterTankSystem.PV_WaterLevel_out", "S7-1500.CV_PumpCtrl_out"]
```

Listing 5. Configuration file for the HiL experiment with PLC integration.

References

[1] Klemets K, Bouzoulas D, Manngård M. FMUiL: functional mock-up units in the loop; 2025. https://github.com/Novia-RDI-Seafaring/FMUiL.

[2] Modelica Association. FMI 2.0.5 standard documentation; 2021. https://fmi-standard.org/docs/2.0.5/ [accessed 29 Aug 2025].

[3] OPC Foundation. Unified architecture; 2025. https://opcfoundation.org/about/opc-technologies/opc-ua/ [accessed 29 Aug 2025].

[4] Lee CG, Park SC. Survey on the virtual commissioning of manufacturing systems. J Comput Des Eng 2014;1(3):213–22.

[5] Berndt O, von Lukas U, Kuijper A. Functional modelling and simulation of overall system ship - virtual methods for engineering and commissioning in shipbuilding. In: European conference on modelling and simulation; 2015.

[6] Scheifele C, Verl A, Riedel O. Real-time co-simulation for the virtual commissioning of production systems. Procedia CIRP 2019;79:397–402.

[7] Jonsson A. Virtual commissioning as a service: standardizing modern co-simulation, [Ph.D. dissertation], Chalmers University of Technology Gothenburg, Sweden; 2024.

[8] Hansen ST, Thule C, Gomes C, Lausdahl KG, Madsen FP, Abbiati G, Larsen PG. Co-simulation at different levels of expertise with maestro2. J Syst Softw 2024;209:111905.

[9] Perabo F, Park D, Zadeh MK, Smogeli Ø, Jamt L. Digital twin modelling of ship power and propulsion systems: application of the open simulation platform (osp). In: 2020 IEEE 29th international symposium on industrial electronics (ISIE). IEEE; 2020. p. 1265–70.

[10] Bertsch C, Süvern M, Sommer T, Schuch K, Reim K, Menne B, Junghanns A, Blochwitz T, Blesken M, Täuber P, et al. Beyond fmi-towards new applications with layered standards. In: Modelica conferences; 2023. p. 381–8.

[11] Palm F, Grüner S, Pfrommer J, Graube M, Urbas L. Open source as enabler for OPC UA in industrial Automation. In: Proceedings of the 2015 IEEE 20th conference on emerging technologies and factory Automation (ETFA). IEEE; 2015. p. 1–6.

[12] Pelkola A, Saarela V, Ojala M, Miettinen T, Savolainen J. Co-simulation with rigorous dynamic simulators and optimizing apc via OPC UA communication. In: Automaatio XXI proceedings. Suomen Automaatioseura; 2015.

[13] Hensel S, Graube M, Urbas L, Heinzerling T, Oppelt M. Co-simulation with OPC UA. In: 2016 IEEE 14th international conference on industrial informatics (INDIN). Poitiers, France; 2016. p. 20–5.

[14] Liu D, Hering D, Carta D, Xhonneux A, Müller D, Benigni A. A hardware-in-the-loop co-simulation of multi-modal energy system for control validation. In: IECON 2021 – 47th annual conference of the IEEE industrial electronics society. Toronto, ON, Canada; 2021. p. 1–6.

[15] Lan J, Zou W, Xu Q, Lai Y, Zhu S. Model-based co-simulation method for PLC programming: interaction design and optimization. In: 2024 IEEE 33rd international symposium on industrial electronics (ISIE). Ulsan, Korea; 2024. p. 1–6.

[16] Severin S, Kyllingstad L, Rindarøy M, Skjong S, Æsøy V, Fathi D, Hassani V, Johnsen T, Nielsen J, Pedersen E. Distributed co-simulation of maritime systems and operations. J Offshore Mech Arctic Eng 2019;141(1):011302.

[17] Centomo S. Modeling and simulation methodologies for digital twin in industry 4.0, [Ph.D. dissertation], University of Verona; 2021.

[18] Kurtović A, Steindl G, Kastner W. Seamlessly interfacing Automation systems with simulation models. In: Proceedings of the 2022 IEEE 5th international conference on industrial Cyber-Physical systems (ICPS); 2022.

[19] Laine T. Digital twin implementation with functional mock-up interface for Automation systems, [Master's thesis], Aalto University; 2024.

[20] Thibeault Q, Pedrielli G. Multicosim: A python-based multi-fidelity co-simulation framework. arxiv:2506.10869, 2025.

[21] Hatledal LI. Ecos: an accessible and intuitive co-simulation framework. J Open Source Softw 2025;10(110):8182.

[22] CATIA-Systems. FMPy; https://github.com/CATIA-Systems/FMPy [accessed 29 Aug 2025].

[23] FreeOpcUa. Opcua-asyncio; https://github.com/FreeOpcUa/opcua-asyncio [accessed 29 Aug 2025].

[24] Pydantic. Validation; https://docs.pydantic.dev/latest/ [accessed 2 Sep 2025].

[25] YAML. YAML ain't markup language; 2025. https://yaml.org/ [accessed 2 Sep].

[26] MathWorks. Watertank simulink model; 2025, mATLAB Documentation, https://se.mathworks.com/help/slcontrol/ug/watertank-simulink-model.html [accessed 2 Sep 2025].