Study on Co-simulation of Heterogeneous Simulators Using Functional Mock-up Interface and Open-Source Toolchains

Jeongsik Kim
Ulsan Intelligent Convergence Research Section
AI Robot Research Division
Electronics and Telecommunications Research Institute
Republic of Korea
j.s.kim@etri.re.kr

Homin Park
Ulsan Intelligent Convergence Research Section
AI Robot Research Division
Electronics and Telecommunications Research Institute
Republic of Korea
hominpark@etri.re.kr

Woo-Sung Jung
Ulsan Intelligent Convergence Research Section
AI Robot Research Division
Electronics and Telecommunications Research Institute
Republic of Korea
woosung@etri.re.kr

Dae Seung Yoo
Ulsan Intelligent Convergence Research Section
AI Robot Research Division
Electronics and Telecommunications Research Institute
Republic of Korea
ooseyds@etri.re.kr

***Abstract*— Computer simulations have been widely utilized within the paradigm of artificial intelligence. However, many existing simulations have been developed in an ad hoc manner, using heterogeneous techniques, tools, and algorithms, which hinders their reusability and interoperability. To address these challenges, this study explores a co-simulation approach through an illustrative case that integrates simulators from physical and network domains by employing a standardized interface and open-source tools. This research is expected to not only facilitate the interoperability of different simulators but also contribute to future collaboration between AI systems and simulation engineers.**

Keywords—co-simulation, functional mock-up interface, hybrid system, network simulator, open simulation platform

I. INTRODUCTION

Computer simulations have played a crucial role in the advance of artificial intelligence, serving as controlled environments and as sources of synthetic data for training and evaluation tasks (Fig. 1) [1]. However, many existing simulation models have been developed in an ad hoc manner using heterogeneous techniques, tools, and algorithms, which hinders their reusability and interoperability [2]. To address these challenges, standardization-based approaches have been proposed to enhance the interoperability from human engineers and computer programs with practical resource for potential extension with AI [3, 4].

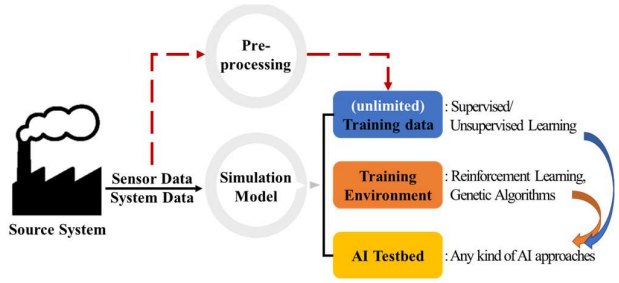

Fig. 1. The role of simulation models in the era of AI [1]

In this context, this paper explores a co-simulation approach through an illustrative case that integrates simulators from physical and network domains by employing Functional Mock-up Interface (FMI) [5] and up-to-date open-source tools (i.e., ns-3 [6], OSP Engine [7], PythonFMU [8]). The FMI-based illustration with this open-source toolchain is expected to facilitate the practical feasibility of co-simulation approaches as well as onboarding for co-simulation beginners.

II. RESEARCH BACKGROUND

Co-simulation is a technique for complex systems by coordinating multiple simulators into a single simulation. Among various co-simulation standards, FMI has gained broad acceptance, with over 200 supporting tools [5]. FMI standardizes both the structure of individual simulation units and their interfaces for data exchange and orchestration.

FMI-based co-simulation typically consists of three key components (as shown in Fig. 2):

- Functional Mock-up Unit (FMU): an FMU is a container-type simulation unit designed to run independently under FMI. Users can generate FMUs using their familiar tools such as Matlab, Python, and OpenModelica. Each FMU encapsulates defined inputs and outputs, internal states, state transition rules, and—if applicable—a local solver for continuous-time integration. While FMI mandates a standardized interface for interoperability, FMUs can be exported either as black-box units (with hidden internals) or as source-accessible modules, depending on the configuration of the modeling tool.
- Co-simulation scenario: a co-simulation scenario defines both the interconnections among FMUs and any external events that influence state transitions during simulation. Depending on the simulation architecture, these scenarios can either be embedded directly within the co-simulation master or provided as external configuration files. The major contributor to FMI ecosystem (i.e., Modelica association) supports the System Structure and Parameterization

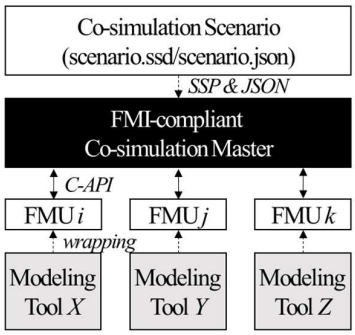

Fig. 2. Illustrative diagram of the ecosystem for FMI-based co-simulation

(SSP) standard, which formalizes the structure, parameterization, and composition of co-simulation systems. Practically, scenario templates are often used to define event schedules in a standardized and machine-readable format (e.g., JSON).

- Co-simulation master: the co-simulation master orchestrates the overall execution of simulations involving multiple FMUs based on a given scenario. While the FMI standard does not specify how the master should be implemented, it requires that each FMU exposes C-API for interactions. In practice, the master handles responsibilities beyond interface-level coordination, including time advancement control, synchronization across simulation units, error compensation, data exchange, and result reporting.

Despite the growing maturity of standard-based co-simulation methodologies, integrating simulators with heterogeneous semantics—such as discrete-event models and continuous-time systems—remains a non-trivial challenge. The fundamental difference in time semantics are as illustrated in Fig. 3 [9]. A few technically feasible solutions have been proposed to advance consistent synchronization, event propagation, and data flow across models. The following summarizes key research efforts to address different co-simulation challenges.

- Non-iterative Algorithms based on Variable Time Steps: F3ORNITS applies a variable step size tailored to the subsystem and effectively handles events and zero-crossings with a non-iterative scheduling approach [10]. An adaptive algorithm using Maestro2 dynamically reorders the algorithms during co-simulation to maintain precision [11]. This approach demonstrates improved computational efficiency and accuracy compared to fixed step sizes.
- Hybrid Time-Event-Driven Approaches: [12] proposes a hybrid time-stepped and event-driven approach that efficiently manages time coordination and information exchange between multiple simulation modules. [13] introduces a superdense time model to address precision issues in environments with mixed floating-point and integer-based time representations and presents a method for integrating multiple time representations. The mosaik 3.0 co-simulation framework also proposes a hybrid approach, integrating models with continuous dynamics and discrete events in a unified simulation environment [13].
- Extending the Master Algorithm through Event Prediction: a structure is proposed in which the master algorithm queries the FMU for the timing of future events, ensuring co-deterministic execution [14]. The FMI 3.0 feature allows an FMU to voluntarily stop its execution within a communication step and return control to the master, signaling that an internal event (e.g., a non-continuous change in outputs) has occurred. This enables the master algorithm to respond immediately to critical events, preventing inaccuracies that might arise from waiting until the end of a fixed interval [15].
- Clock-Based Synchronization Based on FMI 3.0: Synchronous Clocked Simulation, introduced in FMI 3.0, clearly communicates the cause and timing of simultaneous events and enables precise event synchronization [16]. Additionally, FMI 3.0 includes a fine-grained control (Scheduled Execution) feature for real-time simulation, allowing fine-grained tuning of the execution time of FMU submodules [17].

These approaches on co-simulating systems are often tailored to specific cases. For instance, Fig. 4 illustrates the option for co-simulating a hybrid system in a commercial tool [9]. It may pose additional barriers to entry for co-simulation practitioners. Similarly, the adoption of FMI 3.0 introduces native support for hybrid simulation and event handling, while its practical uptake remains limited due to tooling and compatibility issues. You can check the available open-source and commercial tools for each version in [5].

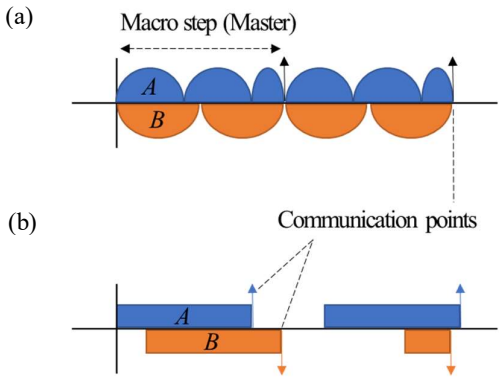

Fig. 3. Illustrated time advances of co-simulating (a) continuous-time system and (b) discrete-event system [9]

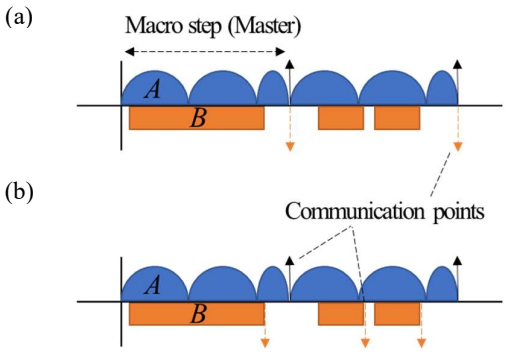

Fig. 4. Illustrated time advances of co-simulating a hybrid system with (a) fixed-step solver and (b) various-step solver [9]

In response to the growing demand for practically reusable approaches [18], recent efforts have presented accessible resources over FMI ecosystem. Open Simulation Platform (OSP) [7] provides an easy-to-use program as a co-simulation master, which is not specified by FMI standard, with reference samples, while DNV [4] presents various sources including the extension for batch experiments and machine learning. [19] developed an ns-3-to-FMI export module that wraps ns-3 scripts as FMUs for integration into FMI-based workflows. These FMUs synchronize ns-3's internal event queue and simulation steps with the master algorithm via standard FMI function calls, enabling flexible composition in multi-domain simulation scenarios. However, the implementation is working on earlier versions of ns-3 and limited to the reconfiguration according to multiple scenarios. [20] proposed an enhanced adapter design that allows the dynamic reconfiguration of ns-3 as a network FMU, supporting runtime interface binding, bidirectional data exchange, and integration into OSP Engine. Based on these works, this study builds an adapter-based illustration integrating two different types of simulators for motional and network dynamics which modern complex systems typically exhibit.

III. METHODS

A. Co-simulation tools

To explore a co-simulation illustration that integrates a continuous-time physical system and a discrete-event network system, this study employs up-to-date open-source tools that constitute the three major components of an FMI-based co-simulation approach, as described below. These tools have provided from installation instructions to utilization examples.

- ns-3: ns-3 [6] is a discrete-event network simulator widely used for research and education in Internet systems. It provides detailed packet-level modeling and supports script-based configuration through C++ or Python. In this work, ns-3 is selected as the representative discrete-event simulator for modeling network-layer behaviors.
- OSP Engine: OSP [7] provides a modular FMI-compliant master implementation known as libcosim. This master coordinates time and data synchronization among distributed FMUs using fixed-step semantics. Additionally, OSP offers an extension called OSP-IS for interface specification and semantic validation. In this study, a demo app of OSP Engine is adopted to serve as the co-simulation master.
- PythonFMU: PythonFMU [8] is an open-source tool that allows developers to easily create and export FMUs using Python. It supports FMI 2.0 and facilitates rapid prototyping of adapter modules or simple simulation components. This tool is used to build the adapter FMU that bridges between ns-3 and the FMI co-simulation framework.

B. Model configuration

A hypothetical scenario is illustrated in which a ship communicates with a port to obtain entrance admission. Depending on the simulation scope, the two entities—ship and port—can be interconnected either directly via FMI or indirectly through a network adapter FMU linked to a discrete-event network simulator. If network-level effects are not of interest, the interaction requires only two FMUs. However, to incorporate more realistic communication dynamics—such as transmission delay, queuing, or packet loss—an adapter FMU can be employed alongside a dedicated network simulator such as ns-3.

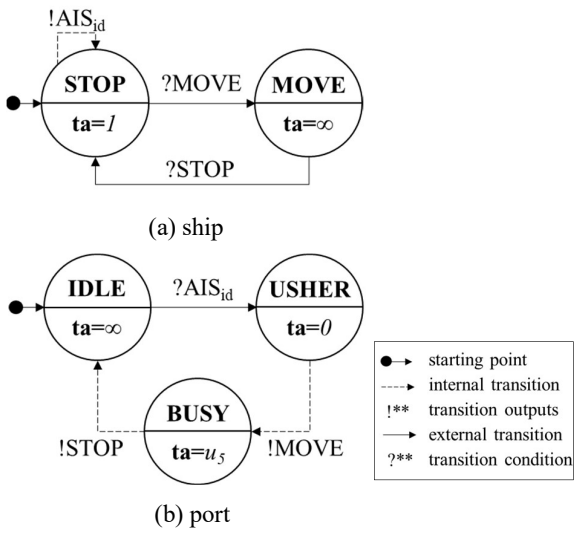

Fig. 5. The state transition diagrams of simulation entities

The ship and port FMUs are illustrated in Fig. 5 using the concept of Discrete Event System Specification (DEVS) [21]. For the sake of simplicity, we assume that a port allows only one vessel to transit at a time. Once the simulation begins, the ship FMU periodically transmits an AIS_{id} message at each simulation step and remains in the STOP state until it receives an entrance approval. Upon receiving a "MOVE" message, the ship transitions to the MOVE state and continues moving at a constant speed of 1 m/s along a one-dimensional coordinate. During this movement, the ship's local coordinate value is updated at each time step, and the updated position is reflected in subsequent AIS_{id} messages. After receiving a "STOP" message, the ship returns to the STOP state. In parallel, the port FMU begins in the IDLE state and waits for incoming "AIS_{id}" messages. Upon receiving one or more, it stops by a transient state (i.e., USHER) to select a ship in the STOP state and transitions to the BUSY state, sending a "MOVE" message to grant access. After a random duration sampled from a uniform distribution over the interval $(0, 5]$, the port returns to the IDLE state by emitting a "STOP" message to handle the next request. This behavior ensures that entrance permission is granted one ship at a time, even when multiple ships are concurrently requesting access.

If network-level dynamics are to be considered, the message exchange between FMUs can be optionally routed through a network adapter FMU, which interfaces with the ns-3 simulator using pre-configured connection parameters and behavior scripts. Based on [20], the adapter handles data exchange via the "REQUEST" and "RESPONSE" calls while maintaining synchronization with the co-simulation master, as shown in Fig. 6. This architecture enables more faithful modeling of maritime communication protocols while preserving modularity of individual simulators and minimizing system complexity. Additionally, it can provide scalability of the simulator structure by simultaneously utilizing network simulators with different environment settings as needed.

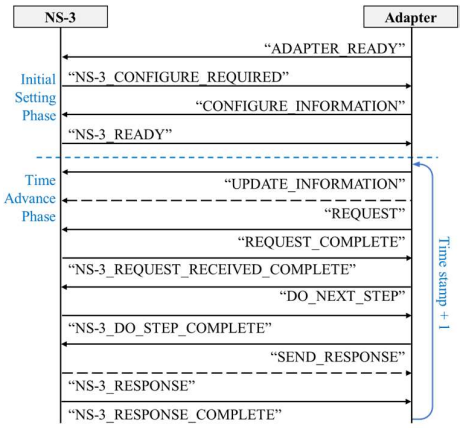

Fig. 6. The sequence diagram of ns-3 and its adapter [20]

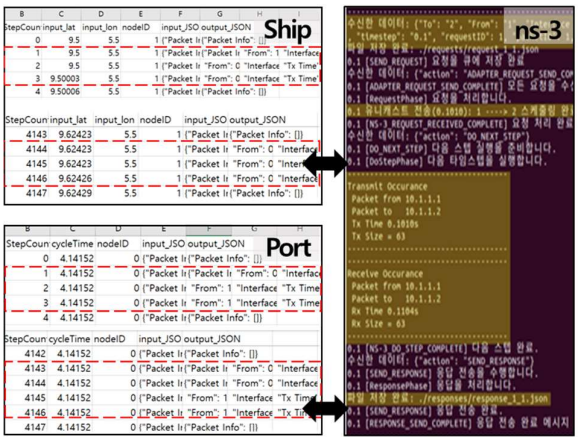

Fig. 7. The outputs of simulation

C. *Illustration*

The simulation environment was configured using Oracle™ VirtualBox 7.0.24 on a Windows 10 x64 workstation equipped with an Intel Core i9 3.20 GHz processor and 64 GB of RAM. The ns-3 simulator (version 3.44) was deployed on Ubuntu 22.04.6 LTS. Two main configuration files were used in the experiment: (1) a system structure description file specifying the connections between FMUs and setting the co-simulation communication step size to 0.001 seconds, and (2) an ns-3 configuration file defining the initialization parameters of the network model, including simulation time step, node positions, and IP configurations.

Fig. 7 presents the output trajectories from the co-simulation tools. The csv format captured by the OSP Engine demonstrates the state transitions of the ship and port FMUs as well as message exchanges between them based on the simulation timelines. The network packet trace generated by ns-3 displays time-stamped messages routed through the adapter FMU.

D. *Observation*

In this scenario, the overall network traffic remains minimal and the two simulation entities exchange messages with low latency. However, the adapter FMU introduces a deliberate delay governed by the configured co-simulation step size. As a result, using the adapter can lead to noticeable communication delays between FMUs compared to the direct (no-adapter) connection mode. For example, in the no-adapter case, a message transmitted prior to the first communication step is delivered to the target FMU in the first communication step. In contrast, when the adapter is adopted, the message is registered by ns-3 scheduler after the first communication step and is delivered in the subsequent one, regardless of the actual delivery time within ns-3. This observation implies that the choice of communication mode should consider both computational overhead and acceptable communication latency.

A benchmark of 100,000 simulation steps was conducted and repeated 20 times to assess computational performance. The average execution time in the no-adapter configuration was approximately 10 seconds, whereas the adapter-enabled configuration required around 60 seconds. This result suggests that the increased runtime observed in the adapter-enabled configuration may stem from the added communication layer as well as potential inter-process latency between simulators.

IV. Discussion

This study explored a FMI-based co-simulation approach utilizing open-source toolchains to verify its technical capability and potential applications. Although the latest version of FMI was not adopted, the presented illustration demonstrates that up-to-date open-source tools can effectively support co-simulation between ns-3 and FMI-based simulators by enabling the integration of heterogeneous dynamics as well as the optional consideration of network behaviors. Furthermore, it was confirmed that each simulator can operate modularly on a separate virtual machine, providing interoperability for independent tool configurations (e.g., version upgrades and algorithmic modifications) without compromising the overall co-simulation workflow. The interoperability from open-source tools and protocols could contribute to the accessibility from human and AI engineers as well as their future collaboration.

Several limitations constrain the generalizability of the illustrated co-simulation approach. First, the illustration does not incorporate real-time operation scenarios (e.g., hardware-in-the-loop experiments or live data injection), where the temporal synchronization is crucial. Second, its scalability remains untested under high-load conditions involving numerous FMUs or dense interconnections. Third, the current verification covers only a limited range of continuous-time models, leaving its applicability to broader physical domains uncertain. Moreover, the extensibility of the framework should be further examined to support emerging applications such as AI-assisted control and its testbed with generative scenario synthesis [22].

Future research could expand this work in several directions. For real-time applications, incorporating extended FMI-compatible interfaces for X-in-the-loop simulation (XiLS) could enable integration with embedded platforms and physical hardware. Coupling with domain-specific simulators—such as robotic or drone dynamics engines—would facilitate validation across diverse continuous-time systems. In terms of scalability, deploying distributed co-simulation over cloud-based infrastructure could enable batch simulations and performance benchmarking for large-scale models. Additionally, the interoperability from open-source tools and protocols is expected to accelerate integrating AI agents into the co-simulation workflow to pave the way toward self-evolving simulation environments.

ACKNOWLEDGMENT

This research was supported by Korea Institute of Maritime Science & Technology Promotion(KIMST) funded by the Ministry of Oceans and Fisheries, Korea(20220534, The Development of Simulation-based Evaluation Technology).

REFERENCES

[1] S. M. M. Sajadieh, and S. D. Noh, "From simulation to autonomy: reviews of the integration of artificial intelligence and digital twins," Int. J. Pr. Eng. Man-GT, pp. 1-32, 2025.

[2] M. Hussain, N. Masoudi, G. Mocko, and C. Paredis, "Approaches for simulation model reuse in systems design—A review," SAE Int. J. Adv. Curr. Pract. Mobil., vol. 4(2022-01-0355), pp. 1457-1471, 2022.

[3] V. Grimm *et al.*, "The ODD protocol for describing agent-based and other simulation models: A second update to improve clarity, replication, and structural realism," J. Artif. Soc. Soc. Simul., vol. 23(2), 2020.

[4] DNV Open Source, "mlfmu: Export ML models represented as ONNX files to Functional-Mockup-Units (FMU)," 2025, github repository. [Online]. Available: https://github.com/dnv-opensource/mlfmu

[5] Modelica Association, "Functional mock-up interface specification," 2025. [Online]. Available: https://fmi-standard.org/

[6] NSnam, "Ns-3: network simulator," 2025. [Online]. Available: https://www.nsnam.org/

[7] Open Simulation Platform, "Open simulation platform: towards a maritime ecosystem for efficient co-simulation," 2022. [Online]. Available: https://opensimulationplatform.com/

[8] L. I. Hatledal, F. Collonval, and H. Zhang, "Enabling python driven co-simulation models with pythonfmu," Proc. 34th Int. ECMS Conf. Model. Simul, 2020. [Online]. Available: https://github.com/NTNU-IHB/PythonFMU

[9] Matlab, "Variable step solvers in Simulink," 2025. [Online]. Available: https://www.mathworks.com/help/simulink/ug/variable-step-solvers-in-simulink-1.html

[10] Y. Eguillon, B. Lacabanne, and D. Tromeur-Dervout, "F3ORNITS: a flexible variable step size non-iterative co-simulation method handling subsystems with hybrid advanced capabilities," Eng. comput., vol. 38.5, pp. 4501-4543, 2022..

[11] S. T. Hansen *et al.*, "Co-simulation at different levels of expertise with Maestro2," J Syst. Softw, vol. 209, p.111905, 2024.

[12] J. Tang et al., "Research on a hybrid time-event-driven co-simulation framework based on the FMI standard," Int. J. Mechatron. Appl. Mech. vol. 18, pp. 222-231, 2024.

[13] A. Ofenloch et al., "Mosaik 3.0: combining time-stepped and discrete event simulation," IEEE OSMSES, pp. 1-5, May 2022, DOI: 10.1109/OSMSES54027.2022.9769116

[14] D. Brodman et al., "Determinate composition of FMUs for co-simulation," Proc. IEEE EMSOFT, p. 1-12, 2013, DOI: 10.1109/EMSOFT.2013.6658580

[15] A. Junghanns et al., "The functional mock-up interface 3.0-new features enabling new applications," Proc. Modelica conf., pp. 17-26, 2021, DOI: 10.3384/ecp2118117

[16] S. T Hansen *et al.*, "The FMI 3.0 standard interface for clocked and scheduled simulations," Electronics, 11(21), p. 3635, 2022, DOI: 10.3390/electronics11213635

[17] C. Coïc *et al.*, "Modelica, FMI and SSP for LOTAR of Analytical mBSE models: first implementation and feedback," Proc. Modelica conf. pp. 49-56, Sep. 2021, DOI: 10.3384/ecp2118149

[18] J. Kim, W.-S. Jung, and N. Kim, "Study on modeling formalism and co-simulation for the system-level test of maritime components," 2023 14th International Conference on Information and Communication Technology Convergence (ICTC), IEEE, pp. 1504-1506, 2023, DOI: 10.1109/ICTC58733.2023.10392735

[19] ERIGrid, "ns-3-fmi-export: Module fmi-export enables the FMI-compliant simulation coupling with ns-3 scripts," 2021, github repository. [Online] Available: https://github.com/ERIGrid/ns3-fmi-export

[20] W.-S. Jung, J. Kim, and D. S. Yoo, "An FMI compliant co-simulation approach for ns-3 network simulator," 2024 15th International Conference on Information and Communication Technology Convergence (ICTC), IEEE, pp. 1954–1955, 2024, DOI: 10.1109/ICTC62082.2024.10827338

[21] B. P. Zeigler, A. Muzy, and E. Kofman. Theory of modeling and simulation: discrete event & iterative system computational foundations. Academic press, 2018.

[22] S. Hong, G. Park, J. S. Kim, "Automated deep-learning model optimization framework for microcontrollers," ETRI Journal, vol. 47(2), p. 179-19, 2025.