2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

Application of FMI 3.0 Synchronous Clocks to Co-simulation of Robotic Systems

Santiago Gil[1,*], Alvaro Miyazawa[2], Ana Cavalcanti[2], Cláudio Gomes[1]

***Abstract*— Given recent technological trends, such as the use of Digital Twins, modeling and simulation have become essential in both the design and the operation of robotic systems. However, developing large simulations that feature several aspects of a robotic system is not trivial. Co-simulation can be used to address this challenge by the hierarchical aggregation of heterogeneous simulators. The recent industry standard Functional Mock-up Interface version 3 (FMI3) adopts the concept of super-dense time to enable both continuous-time and discrete-event simulation. In this work, we extend an existing framework for robotics that supports the creation of modular co-simulation scenarios with simulated or real robots in the loop by upgrading its artifacts to support hybrid co-simulation with FMI3. With these upgraded artifacts, the extended framework accounts for both continuous-time and discrete-event simulations, providing a more comprehensive coverage of scenarios and behaviors of robotic systems. We use a UR5e robotic arm for the demonstration of the FMI3-enabled framework and provide our findings and recommendations from the experiments executed.**

I. Introduction

Modeling and simulation are key to respond *what if* questions in many system-engineering efforts, providing the possibility of risk-free experiments in virtual environments [1]. Different aspects of modeling and simulation are important in the robotics domain, including kinematics, dynamics, and motion planning, among others [2]. With recent technological trends, such as the use of Digital Twin (DT), simulation has come to play a crucial role in the engineering of robots with DTs. The literature reports a wide range of applications [3], including collaborative and cooperative robots [4], [5].

Although there are many robotic simulators [6], integrating simulations of large systems with heterogeneous components is not trivial. Co-simulation addresses this challenge by enabling a seamless integration of software, hardware, and communication infrastructure, serving as an enabling technology to realize DT applications [7], [8].

In previous work [8], we have proposed a co-simulation-driven approach for DTs in robotics using RoboSim [9]. This is a domain-specific language to model robot simulations with support for code generation. Our technique supports the highly automated creation of modular co-simulation scenarios with simulated or real robots in the loop. However, that approach lacks proper management of events, using by default a step-based (i.e., periodic) handling of events. In this work, we address this limitation by adopting the Functional Mock-up Interface (FMI) standard on its third version (FMI3) [10]. It enables hybrid co-simulation through the use of events (*clocks* and *clocked variables*) along with continuous-time simulation using the concept of *super-dense time* [10]–[12].

A key motivation for incorporating FMI3 co-simulation in robotics is that we foresee its potential in multi-robot applications that rely on both continuous-time and discrete-event features, especially those in large collaborative or cooperative settings with multiple stakeholders behind robot models and simulations.

In this paper, we describe the extension of the work in [8] to enable hybrid co-simulation and its application with a single robotic arm. We showcase the added functionality, analyze its benefits, and provide guidance for developers.

II. Background

a) Co-simulation: is a technology and a special kind of simulation that enables the global simulation of coupled systems by combining black-box simulation units and iteratively advancing simulation steps while exchanging data via input and output connections [7]. Co-simulation enables the answer to *what-if* scenarios of larger systems composed of multiple black-box simulation units, where each unit shares a common interface with its fellow simulation units, preserving the same structure as the system represented. Co-simulations can be implemented using the FMI standard.

b) FMI: The FMI standard defines a container and an interface to exchange dynamic simulation models using a combination of files and binaries, distributed as a a black-box *.zip* file [10]. A simulation unit in the context of FMI is called an FMU (Functional Mockup Unit), which includes the implementation of FMI API functions that enable interactions with the FMU and a description file, which determines the structure of inputs and outputs and other metadata of the FMU, such as datatypes and dependencies. A co-simulation execution with FMI requires an external master algorithm, which imports the FMUs and iterates over them following a set of connections that represent the coupling among the simulation units, usually as an orchestration engine.

FMI allows to exchange dynamic simulation models with three interfaces: *model exchange*, which requires that the master algorithm has the solver for the imported models; *co-simulation*, where the FMUs contain both the model and the solver, and the master algorithm only needs to orchestrate the

This work has been funded and supported by the DIGIT-Bench project (case no. 640222-497272), funded by the Energy Technology Development and Demonstration Programme (EUDP).

[1]Department of Electrical and Computer Engineering, Aarhus University, Finlandsgade 22, Aarhus, 8200, Denmark

[2]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, United Kingdom

*Corresponding author: sgil@ece.au.dk

connections; and the FMI3-only *scheduled execution* (SE), where model partitions can be executed separately by the master algorithm [10]. The co-simulation interface in FMI3 works in both continuous-time and discrete-event modes.

c) Synchronous clocks and super-dense time: FMI3 adopts the concept of super-dense time with formulation $t = (t_R, t_I)$ where $t_R \in \mathbb{R}_{\geq 0},\ t_I \in \mathbb{N}_{\geq 0}$, which is a mathematical abstraction of the independent variable *time* to represent continuous time (t_R) and event occurrences (t_I) for a given continuous-time instant [10]–[12]. Event handling in a hybrid co-simulation may introduce discontinuities in the calculation of continuous-time variables.

Synchronous clocks simulations or simply *clocked simulations* are an alternative to achieve the co-simulation of hybrid systems [12]. A clocked-simulation instance in the context of FMI, an FMI3 FMU, needs to advance in both continuous time by time-based steps (`stepT`) and in event iterations by event-based steps (`stepE`). The former executes a periodic computation accounting for continuous time (t_R) and the latter executes an aperiodic computation accounting for discrete events (t_I) [12].

Clocked-simulation instances can also be coupled using an external master algorithm. In the context of FMI3, there are periodic and aperiodic time-based clocks and triggered clocks that tick unpredictably [10]. It is the master algorithm's duty to advance the super-dense time, switch between step and event modes, and tick the time-based clocks of its clocked-simulation instances when they should tick [12]. It is also the master algorithm's task to adapt the co-simulation steps with proper step sizes accordingly when events occur in the middle of two time steps, which are handled with the flag `early return` [10]. Ports in clocked simulations are divided into two categories, the continuous-time ports and the discrete-event ports. In FMI3, variables connected to discrete-event ports are called *clocked variables* and can only be accessed in event mode.

d) Existing framework for DTs in robotics using co-simulation and RoboSim: Our previous work [8] proposes a three-level architecture to achieve co-simulation-driven DTs using RoboSim [9], a tool-independent modeling framework for simulations of robotic systems. At the *model artifact level*, three types of models are used, namely *d-model*, a state-based model providing a platform-independent account of the control software whose evolution is controlled by passage of time that defines simulation cycles; *p-model*, a physical model providing a continuous account of the robot behavior; and *platform mapping*, a model to connect interactions between the d-model and the p-model, whose evolution is defined by simulation cycles. This structure is important for model-driven engineering, as it allows the independent analysis and use of each component and the reuse of the same code for simulation and control of a real robot. These model artifacts are used to create the implementation artifacts *d-model C*, *robotic platform* (real or simulated), and *platform mapping interface* respectively, which can be used to execute applications in real or simulated robots at the *implementation artifact level*. Finally, the implementation artifacts are provided with the FMI interface, so that they work as co-simulation artifacts at the *co-simulation artifact level*. The resulting artifacts at the co-simulation artifact level are the *d-model FMU*, containing the behavior of that defined in the d-model, the *platform mapping FMU*, containing the behavior of that defined in the platform mapping, and the *controller FMU*, which is used as the interface between the user and the co-simulation to send scenarios and tests for execution and to log data from these. The robotic platform interacts with the platform mapping FMU using its underlying interface.

III. Related Work

Previous works [12], [13] have analyzed similarities and differences between the formalisms for clocked simulations with FMI3 and Discrete-Event System Specification (DEVS), a more established framework to describe dynamic systems based on events. Both are designed to handle discrete events in simulation with synchronization to ensure consistent interactions across components, although FMI3 works differently than DEVS. In the case of FMI3, event handling works based on input and output clocks activated at predefined intervals or by external triggers in a modular co-simulation. Multiple FMUs are integrated by an orchestration mechanism with no formal semantics.

Early attempts to use FMI with discrete events considered DEVS as a complement to incorporate clocked simulation formalisms. Some of the existing approaches in this regard include: the work of Camus et al. [14], who wrapped FMUs to integrate them with DEVS in the MECSYCO middleware; the work of Quraishi et al. [15], who proposed a DEVS-FMI interface enabling the hybrid co-simulation of hardware-software interactions; and the work of Lin [16], who proposed a DEVS-FMI adapter for integrating DEVS-Suite and FMUs to enable the co-simulation of discrete-event and continuous-time models. A more recent approach, considering the newer FMI3, is proposed by Vannomeslaeghe et al. [13], where FMI3 is used to wrap DEVS models and enable hybrid co-simulation. In this work, we completely rely on the clocked simulation formalisms of FMI3.

Moving to the application of co-simulation in robotics, there have been approaches to achieve the composition of heterogeneous multi-robot system models and simulations, e.g., in cooperative robot applications [4], [17]. However, achieving a proper combination of system dynamics and behavioral interactions is complex since models may be of a different paradigm, rely on external, usually closed-source, simulation environments, and do not have a common management of the independent variable *time*. Co-simulation has shown the ability to address integration challenges of heterogeneous models and simulations for the composition of larger systems by hierarchical aggregation [18], [19]. Co-simulation has also been used in the robotics domain to run simulations of multi-robot systems with heterogeneous models, simulators, and simulation environments [20], [21] and to realize DTs of cooperative robots [5], [22].

A relevant application of modeling and simulation in modern robotic systems is DTs. The solutions for DTs in

robotics require a high engineering effort [3], and there is also a lack of specialized tooling in the domain [8], [23]. To address these shortcomings, a methodological approach to create co-simulation artifacts that can be used to realize applications with hardware-in-the-loop and DTs was proposed in our previous work [8]. However, its current scope is limited to FMI2 co-simulations, where only continuous-time simulation is available. Therefore, events are handled on a periodic basis on `stepT` simulation steps. In this work, we complement the results in [8] by extending the co-simulation artifacts with FMI3 capabilities to enable both continuous-time and discrete-event co-simulation and achieve a more comprehensive framework supporting clocked simulations.

IV. Approach

As per the FMI3 standard, a clocked simulation performs time-based steps (`stepT`) and event-based steps (`stepE`), as described in Section II. In our architecture, the d-model FMU has the control logic and its behavior is purely event-based. The controller FMU has a combined behavior since it manages the scenario and test execution in event mode, but it periodically receives logging data from the platform mapping FMU; however, it does not perform any computation in time-based steps. The platform mapping FMU interacts with the robotic platform on both continuous time and discrete events. Therefore, the platform mapping FMU and the controller FMU require the functionality for both the `stepT` and `stepE` simulation steps, whereas the d-model FMU only requires the logic for the `stepE` simulation steps.

To enable FMI3 co-simulation, we have updated the co-simulation artifacts of the architecture proposed in [8] as follows. First, we have updated all connections between FMUs so that these match the combination of discrete-event and continuous-time simulation by refactoring the former `Boolean` variables (representing events in the approach with FMI2) to `Clock` type in FMI3, and clocked variable type to those arguments being passed along with events. This change ensures that clocks and clocked variables are only accessed in event mode. Secondly, the *d-model FMU* is updated so that the FMU is compiled with the FMI3 headers with an updated container of variables including those stated in the FMI3 standard (e.g. using `Float32` or `Float64` instead of the former `Real` of FMI2) and the implementation of the FMI3 standard functions. We also updated the d-model FMU functionality so its behavior is executed in `stepE` steps, which triggers the cycles of its internal state machine. Thirdly, the *platform mapping FMU* and the *controller FMU* are updated with the latest release of UniFMU (v0.12.0-beta) [24], a convenient tool that enables the creation of FMUs in universal languages that are not capable of generating C-compatible binaries (as required by FMI) and provides features to use those FMUs in distributed co-simulation settings [25]. For these two FMUs, we update the integration mechanism so that the continuous-time logic stays in the `stepT` steps, whereas the discrete-event logic is moved to the `stepE` steps. Figure 1 describes the resulting *co-simulation artifacts* extending the architecture presented in [8] with FMI3 and hybrid co-simulation support.

Since an FMI co-simulation depends on an external master algorithm, we have developed a case-specific one to our needs that is in charge of managing the switching between step and event modes of the FMUs and passing the data from sources to sinks. Currently, our co-simulation master algorithm uses an extended and non-oficial version of the FMPy library[1] because, to the best of our knowledge, there is not yet full support for FMI3 co-simulation from FMPy or other open-source co-simulation master algorithms.

In our co-simulation master algorithm, we execute the logic with two loops. The outer loop iterates in continuous-time mode until the co-simulation end time is reached. The inner loop iterates in event mode any time event mode is needed, which comes as a result of executing `stepT` in an FMU. If after executing the `stepT` of all FMUs in the continuous-time loop, any of them returns a request for event handling, event mode is activated, and the inner loop runs for as many `stepE` iterations as needed by the FMUs. In this case study, the FMUs go into event mode as follows: The platform mapping FMU periodically reads the robotic platform state in continuous-time mode. If there is a condition in the robotic platform that triggers an event in the platform mapping equations, the platform mapping FMU goes into event mode, and events are forwarded to the d-model FMU. The controller FMU goes into event mode if the scenario or test comes to a point where an action or a set of actions needs to be executed, e.g., when a moving command is to be executed; these actions are represented as events and are forwarded to the d-model FMU. Similarly, for the d-model FMU, if its state machine(s) comes to a point where a command coming from the controller FMU triggers an operation, e.g., a moving command (given a set of conditions and guards), it goes into event mode, executes its state machine, and forwards any eventual operations to the platform mapping FMU as events. Events that occur between the three FMUs are passed as *triggered clocks* in event mode.

RoboSim models, which are used to generate d-model FMUs, are particularly suited for implementation in FMI2, as they provide an explicit definition of the cyclic behavior of the system. Nevertheless, RoboSim is a more concrete modeling language within the RoboStar family[2]. It is intended as an intermediate model between the more abstract RoboChart models [9], which model reactive systems, and implementations for simulation and deployment, where the scheduling of behaviors is made explicit.

The particular use case for RoboSim is the support of the development of cyclic simulations from RoboChart reactive models. In particular, a refinement strategy has been developed to convert reactive RoboChart models into cyclic RoboSim models that are correct with respect to the RoboChart model under assumptions about the timing of input events. The RoboSim model can be manually or

[1] `https://github.com/CATIA-Systems/FMPy`
[2] `https://robostar.cs.york.ac.uk/`

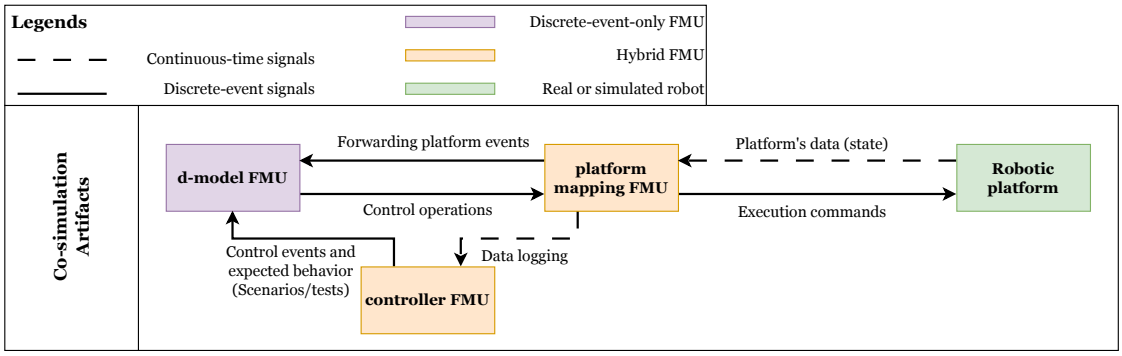

Fig. 1. FMI3-enabled co-simulation artifacts, extended from [8].

automatically compiled into a cyclic implementation suitable for simulation or deployment.

The expanded architecture with FMI3 support allows us to avoid the use of RoboSim models and target RoboChart models directly. Clocked and SE simulations can both be used to implement RoboChart models; the main difference being how RoboChart time is modeled.

V. Demonstration and Findings

To demonstrate the use of our extended framework, we use the same case study as in [8], which is a Universal Robots UR5e robotic arm. We reuse the existing model and implementation artifacts, and update the co-simulation artifacts with the libraries and implementation for FMI3 co-simulation. In this example, we use a set of positions in a discretized Cartesian space (X, Y, Z), where $X, Y, Z \in \mathbb{Z}$, the robot is expected to follow, as proposed in [4], [17]. Such commands are embedded in the controller FMU as part of a plan, which are passed to the d-model FMU as events using the `moveDiscreteCommand` clock and its corresponding arguments as clocked variables. When the d-model FMU receives this event, it triggers the computation defined in the state machine and eventual operations to the platform mapping FMU. The test uses a cyclic task, where the robot repetitively goes to the positions $(0, 23, 1)$, $(3, 20, 2)$, $(8, 10, 0)$, and $(1, 13, 0)$ at given real clock times. We use CoppeliaSim[3] as the robotic simulator for this demonstration. For this test, we are using a step size of $\Delta_t = 0.5s$ and a total simulation time of $100.0s$ following real time, that is, we expect that the average step computation time (including any occurring events) is approximately equal to Δ_t. Figure 2 shows the results of the co-simulation as follows: subplot (1) shows the execution of the triggered clock `moveDiscreteCommand` from the controller FMU, with the target positions in subplot (2) as arguments, to the d-model FMU. The d-model FMU exhaustively executes its internal state machine and sends the resulting operations to the platform mapping FMU. The platform mapping FMU interacts with the robotic platform via commands and also reads its state periodically, as shown in subplot (3), which is used to calculate the robotic platform's events given the rules within the platform mapping, as shown in subplot (4). That is, `robotStopped` is true when all joint velocities are approaching zero ($\dot{q}_i < 0.1\frac{rad}{s}, \forall i$); `moveCompleted` is true when all joint positions are the same as their desired target positions ($q_i == q_{i_{target}}, \forall i$) rounded by two decimal digits. Notice that `moveCompleted` uses a more strict comparison method that causes more toggling in the result than `robotStopped`.

From these experiments and comparisons with our previous approach using FMI2 [8], we can assert the following:

- The continuous-time behavior of the two approaches is the same. There is no advantage in using FMI3 synchronous clocks if only continuous-time simulation is required.
- There is an extra benefit when using FMI3 since the co-simulation can happen on both axis of super-dense time, namely `stepT` and `stepE`, compared to the FMI2 `stepT` which only happens on the continuous-time axis. Additionally, there can be multiple `stepE` iterations in a super-dense continuous-time instant. In this sense, the simulation can account for any intermediate computations between time steps, improving the controllability and observability of simulations of state machines.
- Although there may be more intermediate computations, the exploitation of this feature depends on the design of the FMU generated by the exporting tool and its underlying code that executes the state machine. In our current case study, the state machine contained in the d-model FMU, which is automatically generated, performs an exhaustive computation of all possible state-transition operations per simulation step. This means that instead of computing an n number of super-dense time iterations for t_I, it performs n computations per `stepE`, and hence, $t_I \leq 1$ in all event mode executions. Therefore, the observability and controllability of state-transition operations are not improved.
- It is likely that FMI3 adaptations to existing code or automatically generated code will fall into the same problem of exhaustively executing state-machine operations, since these codes are usually programmed to be self-contained and self-executed. In FMI3 co-simulation, the execution depends on the master algorithm, which feeds and retrieves data from its subordinate FMUs during simulation steps.

[3] `https://www.coppeliarobotics.com/`

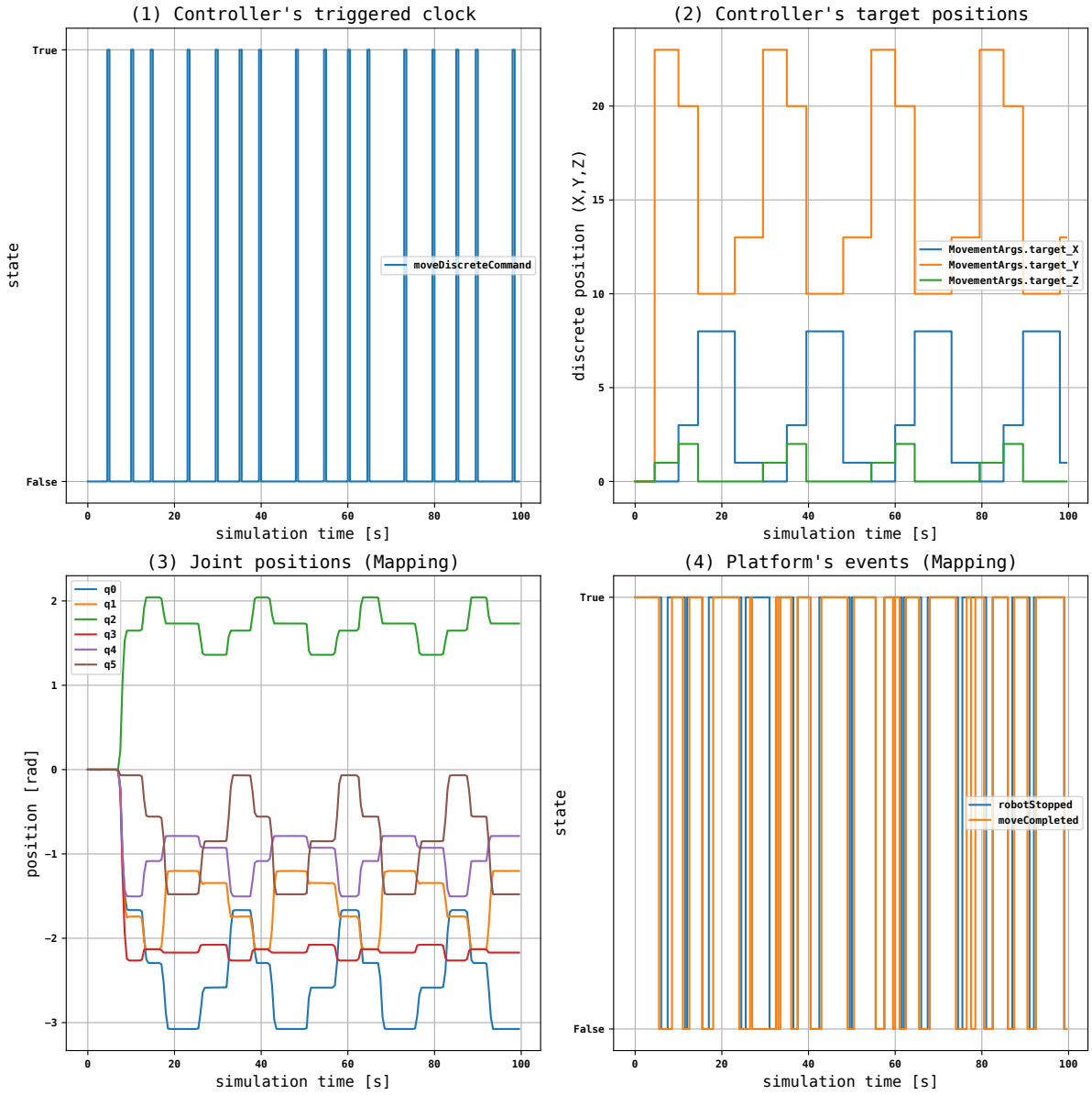

Fig. 2. Results of the co-simulation experiment with FMI3.

Therefore, having a match between `stepE` and execution cycles in coded state machines provides a better compatibility with FMI3 co-simulation, enabling the advance of super-dense time as the state machine fires transitions that activate other states. Thus, there will be full observability and control of the state-transition operations by the master algorithm, enabling more accurate integration of multiple state machine FMUs, since their `stepE` can affect each other and propagate events at a finer grained level.

In our case, as mentioned in Section IV, the FMI3-enabled architecture allows us to go beyond RoboSim and consider the more abstract modeling language RoboChart [9]. RoboChart has a more compatible integration of state machine software with FMI3 clocked simulations, which is also more convenient for not cyclic simulations, such as ROS-based simulations, as we have noticed. For these clocked simulations, RoboChart events and time are modeled uniformly. Events are implemented as triggered clocks and processed in event mode. Time in RoboChart is discrete and recorded by the special event $tock$ and is modeled as periodic time-based clocks. This treatment allows us to handle the timing of events without extraneous assumptions about their occurrence (e.g., events are only observed at the start of a simulation step).

Recalling the formulation of super-dense time in Section II, a RoboChart time unit ($tock$) can be implemented with a time-based periodic input clock C_{RC}. This input clock models the RoboChart discrete time with a period of P_{RC}. Thus, RoboChart time elapses P_{RC} in the t_R axis of super-dense time when the master algorithm ticks the clock C_{RC}.

VI. Concluding Remarks

This work presents a pioneering approach that integrates hybrid co-simulation for robotic system applications using FMI3 and an existing framework for co-simulation-driven DTs and other enabling technologies, such as RoboSim, UniFMU, and CoppeliaSim. The source code, artifacts, templates, examples, and videos for the original and extended approaches are publicly available on GitHub[4].

This extension to the approach proposed in [8] poses a set of challenges that may be worth considering in future work. First of all, since FMI3 is not yet widely adopted, the number of tools, especially regarding co-simulation master algorithms, may be limited or may have limitations in terms of functionality. Moreover, we are creating our own case-specific master algorithm, which only works with this simulation scenario and, therefore, is not very scalable, does not support a faithful real-time execution, and does not support handling of events occurring in the middle of two time steps. Yet, we expect that there will be better upcoming support for this soon from the modeling and simulation community.

More importantly, since the code embedded in the d-model FMU (or potentially any other existing code for state machines) is not compatible with the abstraction of super-dense time and `stepE` iterations, we cannot quantitatively assess the performance of this extended approach in terms of more efficient event handling. We have studied a more compatible integration of state machine software with FMI3 FMUs using RoboChart models instead of RoboSim models; however, automatic code generation for RoboChart models is not yet available, which we plan to address in an upcoming release. Additionally, we need to further investigate a more appropriate way to embed existing state machine software with exhaustive computation into FMI3 FMUs.

Finally, since this approach enables integration with hardware-in-the-loop simulation that can be used in several applications not limited to the robotics domain, we foresee the potential of mixing the features of this extended architecture with FMI3 for the co-simulation of electromechanical testing, in particular for test-bench facilities of wind turbines.

References

[1] A. Maria, "Introduction to modeling and simulation," in *Winter Simulation Conference Proceedings*, 1997, pp. 7–13.

[2] B. Siciliano and O. Khatib, *Springer handbook of robotics*, 2016.

[3] A. Mazumder, M. Sahed, Z. Tasneem, P. Das, F. Badal, M. Ali, M. Ahamed, S. Abhi, S. Sarker, S. Das, M. Hasan, M. Islam, and M. Islam, "Towards next generation digital twin in robotics: Trends, scopes, challenges, and future," *Heliyon*, vol. 9, no. 2, p. e13359, 2023.

[4] S. Gil, C. Schou, P. H. Mikkelsen, and P. G. Larsen, "Integrating Skills into Digital Twins in Cooperative Systems," in *2024 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2024, pp. 1124–1131.

[5] S. Gil, B. J. Oakes, C. Gomes, M. Frasheri, and P. G. Larsen, "Toward a systematic reporting framework for digital twins: a cooperative robotics case study," *Simulation*, vol. 101, no. 3, pp. 313–339, 2025.

[6] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51 416–51 431, 2021.

[7] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Computing Surveys*, vol. 51, no. 3, 2018.

[8] S. Gil, A. Badyal, A. Miyazawa, P. G. Larsen, and A. Cavalcanti, "A model-based approach for co-simulation-driven digital twins in robotics," *Robotics and Autonomous Systems*, vol. 196, p. 105240, 2026.

[9] A. Cavalcanti, A. Sampaio, A. Miyazawa, P. Ribeiro, M. Conserva Filho, A. Didier, W. Li, and J. Timmis, "Verified simulation for robotics," *Science of Computer Programming*, vol. 174, pp. 1–37, 2019.

[10] The Modelica Association Project FMI, "Functional mock-up interface (fmi) 3.0," https://fmi-standard.org/, 2024, accessed on July 22, 2025.

[11] E. A. Lee and H. Zheng, "Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems," in *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, 2007, pp. 114–123.

[12] S. T. Hansen, C. A. G. Gomes, M. Najafi, T. Sommer, M. Blesken, I. Zacharias, O. Kotte, P. R. Mai, K. Schuch, K. Wernersson, C. Bertsch, T. Blochwitz, and A. Junghanns, "The fmi 3.0 standard interface for clocked and scheduled simulations," *Electronics*, vol. 11, no. 21, 2022.

[13] Y. Vanommeslaeghe, B. Van Acker, J. Denil, and P. De Meulenaere, "Integrating devs and fmi 3.0 for the simulated deployment of embedded applications," in *2024 Annual Modeling and Simulation Conference (ANNSIM)*. IEEE, 2024, pp. 1–13.

[14] B. Camus, V. Galtier, M. Caujolle, V. Chevrier, J. Vaubourg, L. Ciarletta, and C. Bourjot, "Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO," in *Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 16)*. Pasadena, CA, United States: Society for Computer Simulation International San Diego, CA, USA, 2016, p. No. 8.

[15] M. H. Quraishi, H. S. Sarjoughian, and S. Gholami, "CO-SIMULATION OF HARDWARE RTL AND SOFTWARE SYSTEM USING FMI," in *2018 Winter Simulation Conference (WSC)*, Dec. 2018, pp. 572–583.

[16] X. Lin, "Co-simulation of Cyber-Physical Systems Using DEVS and Functional Mockup Units," Tech. Rep., 2021.

[17] S. Gil, P. H. Mikkelsen, D. Tola, C. Schou, and P. G. Larsen, "A Modeling Approach for Composed Digital Twins in Cooperative Systems," in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2023, pp. 1–8.

[18] C. Gomes, B. Meyers, J. Denil, C. Thule, K. Lausdahl, H. Vangheluwe, and P. De Meulenaere, "Semantic adaptation for FMI co-simulation with hierarchical simulators," *Simulation*, vol. 95, no. 3, pp. 241–269, 2019.

[19] J. Michael, J. Pfeiffer, B. Rumpe, and A. Wortmann, "Integration Challenges for Digital Twin Systems-of-Systems," in *SESoS*, ser. SESoS. IEEE/ACM, 2022, pp. 9–12.

[20] F. Keppler, M. Kramer, O. Koch, S. Wagner, and K. Janschek, "Prioritized planning for spatiotemporal trajectory coordination of articulated vehicles and co-simulation with ros, docker and gazebo," in *2022 IEEE/SICE International Symposium on System Integration (SII)*, 2022, pp. 279–284.

[21] E. Dey, J. Hossain, N. Roy, and C. Busart, "Synchrosim: An integrated co-simulation middleware for heterogeneous multi-robot system," in *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2022, pp. 334–341.

[22] S. Gil, E. Kamburjan, P. Talasila, and P. G. Larsen, "An architecture for coupled digital twins with semantic lifting," *Software and Systems Modeling*, vol. 24, no. 5, pp. 1379–1404, 2025.

[23] S. Gil, P. H. Mikkelsen, C. Gomes, and P. G. Larsen, "Survey on open-source digital twin frameworks–A case study approach," *Software: Practice and Experience*, vol. 54, no. 6, pp. 929–960, 2024.

[24] C. M. Legaard, D. Tola, T. Schranz, H. D. Macedo, and P. G. Larsen, "A universal mechanism for implementing functional mock-up units," in *11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, ser. SIMULTECH, 2021, pp. 121–129.

[25] S. Gil, E. E. Baş, C. D. Jensen, S. Engelsgaard, G. Abbiati, and C. Gomes, "Fmi-based distributed co-simulation with enhanced security and intellectual property safeguards," in *2025 Annual Modeling and Simulation Conference (ANNSIM)*, 2025.

[4] `https://github.com/INTO-CPS-Association/DigitalTwins_RoboSim`