SYNTHESIZING ORCHESTRATION ALGORITHMS FOR FMI 3.0

Simon Thrane Hansen
Cláudio Gomes
Zahra Kazemi

Department of Electrical and Computer Engineering
Aarhus University
Helsingforsgade 10
Aarhus, DENMARK
{sth, claudio.gomes, zka}@ece.au.dk

ABSTRACT

An essential part of building reliable cyber-physical systems is to be able to predict the behavior of such systems using accurate simulations. Standards for describing the behavior of such systems and simulations are evolving to facilitate a broader range of applications. The Functional Mockup Interface (FMI) standard is no exception. FMI 3.0 introduces synchronous clocks to facilitate efficient and repeatable simulation of event-driven systems. Nevertheless, the standard does not specify how to implement the synchronization of models, and it is up to the tool vendors to implement the orchestration algorithm responsible for this task. This paper presents the first approach to synthesizing the orchestration algorithm for FMI 3.0 supporting synchronous clocks. A prototype implementation of the algorithm is presented.

Keywords: Functional Mockup Interface, synchronous clocks, reactive systems, scheduling.

1 INTRODUCTION

Cyber-physical systems (CPSs) are a vital part of modern society, with applications ranging from nuclear power plants and airplanes to cars and other complex systems. These systems usually consist of both cyber components (like controllers) and physical components, and their modeling and simulation often require multiple paradigms, including continuous-time, modal models, and discrete events. However, the wide variety of tools and formalisms used by different specialized companies to develop these systems can make it challenging to achieve interoperability (Paris et al. 2019). To address this challenge, the Functional Mockup Interface (FMI) Standard was developed to enable the exchange and cooperative simulation of black-box models (Blockwitz et al. 2012) using a vendor-independent interface. The black-box models are called Functional Mockup Units (FMUs) and describe the behaviour of a continuous subsystem as a discrete trace. An FMU can be composed with other models using input and output variables to form a larger system. Version 2.0 of the standard has been widely adopted, with more than 170 tools supporting it (FMI 2014), which has placed a growing demand on the standard to support simulation of real-time and reactive systems.

The FMI 3.0 standard (Functional Mockup Interface Steering Committee 2021) introduces two notable new features inspired by other simulation standards such as Modelica (Association 2021), DEVS (Zeigler 1976), and HLA (Dahmann 1997) to address these challenges: synchronous clocks (SC) for event-driven synchronization and scheduled execution to provide real-time simulation capabilities. Our work focuses on the first of these features, SC (Elmqvist et al. 2012, Benveniste et al. 2003), which is a powerful feature that allows FMUs to synchronize their execution based on the occurrence of time-based and state-based events to simulate reactive systems. While FMI 3.0 provides a standardized interface for FMUs, it does not specify how to implement their synchronization. Therefore, it is up to the tool vendors to implement the orchestration algorithm responsible for this task. The question of how to implement the orchestration algorithm is particularly challenging for SC, as the orchestration algorithm must be able to detect and devise specific strategies for handling the different types of events that can occur in the system.

Contribution This paper presents what we believe to be the first approach for synthesizing orchestration algorithms enabling simulations of SC FMUs following FMI 3.0. We propose a graph-based approach based on earlier work on FMI 2.0, which we have extended to support the event-driven nature of FMI 3.0.

Related Work Synthesizing co-simulation algorithms for FMUs has been a topic of research for several years and has been addressed in several works (Gomes et al. 2019, Galtier et al. 2015, Broman et al. 2015, Hansen et al. 2021). These works have addressed the problem of synthesizing co-simulation algorithms for FMUs using various graph-based approaches (Gomes et al. 2019, Galtier et al. 2015, Broman et al. 2015). None of these approaches, however, support the event-driven nature of the FMI 3.0, making them unsuitable for simulating SC FMUs. On the other hand, some works have been addressing simulations of event-based systems with synchronous clocks in other simulation standards, such as Modelica, DEVS (Zeigler 1976). Compared to SC, DEVS and HLA take a purely discrete event based approach, where continuous dynamics are quantized (Kofman and Junco 2001, Zeigler and Lee 1998). However, this approach is not suitable for simulating FMUs, as it does not offer mechanism to handle algebraic loops or to solve differential algebraic equations, which occur commonly in numerical simulations.

Organization of the Paper The rest of the paper is organized as follows. Section 2 provides background on the FMI 3.0 standard and the synchronous clocks feature and describes the challenges that arise when simulating synchronous clocks FMUs. Section 3 presents the proposed approach for synthesizing orchestration algorithms for synchronous clocks simulations. Section 4 present the results of the validation of the proposed approach. Finally, Section 5 concludes the paper and discusses future work and limitations.

2 BACKGROUND

This section provides background on the FMI 3.0 standard and the synchronous clocks feature and describe the challenges of simulating synchronous clocks FMUs. Due to space limitations, we can only provide a brief overview of these topics and refer the reader to FMI 3.0 (Functional Mockup Interface Steering Committee 2021, Hansen et al. 2022) for more details. A general introduction to co-simulation is provided in (Gomes et al. 2019). We admit that the notation and abbreviations that follows used in this section may be hard to follow for new readers, and we, therefore, provide a brief introduction to the notation in Appendix A on page 12. The nomenclature follows (Kubler and Schiehlen 2000).

FMI is a standard for describing composable self-contained executable subsystems called FMUs (Blockwitz et al. 2012). FMUs communicate with their environment through port variables (continuous, discrete, or clock inputs/outputs), which are connected to ports of other FMUs to denote dependencies/coupling restrictions between them. FMI 3.0, includes features to support event-based simulation using the synchronous clocks feature, which allows FMUs to trigger events at specific times (time-based events) or under certain conditions (state-based events). A clock is a discrete boolean variable used to trigger events and synchronize the execution of FMUs - the importer is responsible for activating input clocks, and the FMU is responsible for activating its output clocks. An event is a request from the FMU to the importer to temporarily stop the simulation and apply a specific event handling algorithm to update the FMU state and to exchange data with other FMUs to solve the event. The strategy for solving the event is not specified by the FMI standard but can, using the techniques described in this paper, be inferred from the FMU interface. FMI 3.0 defines two types of clocks: time-based clocks and state-based clocks. While the FMI standard describes the functions available for simulation orchestration, it does not specify when or how to use them, which is the paper's focus. The interface of an SC FMU is summarized in Definition 1.

Definition 1 (SC FMU Instance)**.** *An SC FMU instance with identifier m is represented by the tuple:*

$$\langle S_m, U_m, Y_m, U_m^c, Y_m^c, \texttt{set}_m, \texttt{get}_m, \texttt{set}_m^c, \texttt{get}_m^c, \texttt{stepT}_m, \texttt{stepE}_m, \texttt{nextT}_m, V_m^P, F_m^C, V_m^c, D_m \rangle$$

where:

- S_m *represents the abstract set of possible FMU states. A given state* $s_m \in S_m$ *of m represents the complete internal state of m: active clocks, active equations, current mode (Step or Event mode) current valuations for input and output variables, etc. The state of an SC FMU is defined in Definition 3.*
- U_m *and* Y_m *represent the set of input and output variables, respectively. A variable* $v \in U_m \cup Y_m$ *is discrete if* $Discrete(v) = true$*, and continuos if* $Discrete(v) = false$*. The sets* $U_m^D = \{u_m \in U_m \mid Discrete(m)\}$ *and* $Y_m^D = \{u_m \in U_m \mid Discrete(m)\}$ *are the set of discrete input and output variables.*
- U_m^c *and* Y_m^c *represent the set of input and output clocks, respectively. The set* U_m^{TC} *denotes the time-based clocks, note that* $U_m^{TC} \subseteq U_m^c$*. The set of triggered input clocks are described by* $U_m^c \setminus U_m^{TC}$*.*
- $\texttt{set}_m : S_m \times U_m \times \mathscr{V} \to S_m$ *and* $\texttt{get}_m : S_m \times Y_m \to S_m \times \mathscr{V}$ *are functions to set the inputs and get the outputs, respectively (we abstract the set of values that each input/output variable can take as* $\mathscr{V}$*). Both* $\texttt{set}_m$ *and* $\texttt{get}_m$ *return a new state because both can trigger the computation of equations, essentially changing the state of the FMU.*
- $\texttt{set}_m^c : S_m \times U_m^c \times \mathbb{B} \to S_m$ *and* $\texttt{get}_m^c : S_m \times Y_m^c \to S_m \times \mathbb{B}$ *are the functions that (de-)activate the input clocks and query the output clocks for its activation status, respectively, and* $\mathbb{B}$ *is the boolean set.*
- $\texttt{stepT}_m : S_m \times \mathbb{R}_{\geq 0} \to S_m \times \mathbb{R}_{\geq 0} \times \mathbb{B}$ *is a function representing the Step mode computation. If m is in state* s_m *at simulated time* $(t_R, 0)$*,* $(s_m', h, b) = \texttt{stepT}_m(s_m, H)$ *approximates the state* s_m' *of m at time* $(t_R + h, 0)$*, with* $h \leq H$*. When* $b = true$*, we know that the* importer *and m have agreed to interrupt the Step mode prematurely, and m is ready to go into Event mode.*
- $\texttt{stepE}_m : S_m \to S_m \times \mathbb{B}$ *represents one super-dense time iteration of the Event mode. If m is in state* s_m *at time* (t_R, t_I)*, then* $(s_m', b) = \texttt{stepE}_m(s_m)$ *represents the discrete state computation m, where* s_m' *represents the new state at* $(t_R, t_I + 1)$ *and b states whether an event has occurred.*
- $\texttt{nextT}_m : S_m \times U_m^{TC} \to \mathbb{R}_{\geq 0} \cup \{NaN\}$ *is the function that allows the* importer *to query the time of the next clock tick. This function is only applicable to a subset of time-based clocks. The value NaN can be returned for countdown clocks, and it means that the clock currently has no schedule.*
- $V_m^P : W_m^C \to 2^{Y_m^D}$ *is a function linking a clock with its variables. The clock partition is the set of discrete output variables that can only be observed when the clock is active.*
- $F_m^C : Y_m^c \to 2^{U_m^c}$ *is a function, linking an output clock with the input clocks that can influence the state of the output clock. It denotes when the input clock* $u_m^c \in U_m^c$ *affects the state of the output clock* $y_m^c \in Y_m^c$*. This means that there exists state of the FMU* s_m *and of the output clock* y_m^c *such that updating the input clock* u_m^c *changes the state of* y_m^c*. Formally,* $\texttt{get}_m^c(\texttt{set}_m^c(s_m, u_m^c, v_1), y_m^c) \neq \texttt{get}_m^c(s_m, y_m^c)$*.*
- $V_m^c : (Y_m^D \cup U_m^D) \to 2^{Y_m^c}$ *is a function describing influence of discrete variable on output clocks.*
- $D_m : Y_m \to \mathscr{P}(U_m)$ *is a function that describes for each output port* y_m *the set of input ports* u_m *that can influence the value of the output port* y_m*. The type of the connected variables can be different.*

Due to space limitations, we omit the formal definition of the functions and refer the reader to the standard and (Hansen et al. 2022) for more details.

Definition 2 (Scenario)**.** *A scenario is a structure* $\langle M, L, L^C, \mathscr{M}, F \rangle$ *where:*

- *M is a finite set (of FMU identifiers).*

- *L is a function $L : U \to Y$, where $U = \bigcup_{m\in M} U_m$ and $Y = \bigcup_{m\in M} Y_m$, and where $L(u) = y$ means that the output y is coupled to the input u. Note that the function is not necessarily injective and that the output variable y and the input variable u must be of the same type and belong to different FMUs.*
- *L^C is a function $L^C : U^C_T \to Y^C$, where $U^C_T = \bigcup_{m\in M}(U^c_m \setminus U^{TC})$ and $Y^C = \bigcup_{m\in M} Y^c_m$. The notation $L^C(u^c) = y^c$ states that the output clock y^c is connected to the input clock u^c.*
- *$\mathscr{M} \subseteq M$ denotes the FMUs that may prematurely terminate the invocation to $\texttt{stepT}_m$ in Step mode.*

The run-time state of an FMU is a syntactic abstraction that captures its state during simulation. This abstraction serves as a concise and implementation-neutral representation of the FMU's state, achieved by disregarding the internal representation of the FMU.

Definition 3 (Run-time State of an SC FMU)**.** *Given an SC FMU m as defined in Definition 1, the run-time state of m is a member of the set $S^R_m = (\mathbb{R}_{\geq 0}, \mathbb{N}) \times Mode \times S^R_{U_m} \times S^R_{Y_m} \times S^R_{U^c_m} \times S^R_{Y^c_m} \times S^R_{W^C_m}$, where:*

- *$(\mathbb{R}_{\geq 0}, \mathbb{N})$ is the super-dense time of the FMU. We write $s_m^{(t_R,t_I)}$ to indicate the FMU m is at time (t_R,t_I).*
- *Mode is the simulation mode of the FMU. An FMU can be in one of the following modes: INIT, EVENT, and STEP. We omit the Terminate mode, since it is irrelevant for the semantics.*
- *$S^R_{U_m} : U_m \to (\mathbb{R}_{\geq 0}, \mathbb{N}) \cup \{NaN\}$, $S^R_{Y_m} : Y_m \to (\mathbb{R}_{\geq 0}, \mathbb{N}) \cup \{NaN\}$, $s^R_{U^c_m} : U^c_m \to (\mathbb{R}_{\geq 0}, \mathbb{N}) \cup \{NaN\}$, and $s^R_{Y^c_m} : Y^c_m \to (\mathbb{R}_{\geq 0}, \mathbb{N}) \cup \{NaN\}$ are functions mapping a variable to a timestamp denoting the last time it was set. The value NaN indicates that the corresponding variable has never been set.*
- *$S^R_{W^C_m}$ is a set describing the active clocks of the FMU. Notice, clocks can only tick in Event mode.*

The co-simulation state is the combination of the run-time states of the FMUs in a scenario.

Definition 4 (SC Co-simulation State)**.** *Given a co-simulation scenario as defined in Definition 2. The co-simulation state is a member of the set $S^R_{\mathscr{S}} = time \times Mode \times S^R_U \times S^R_Y \times S^R_{U^C} \times S^R_{Y^C} \times S^R_{W^C}$, where:*

- *$time : M \to (\mathbb{R}_{\geq 0}, \mathbb{N})$ is a function, where $time(c)$ is a function denoting the current super-dense simulation time of FMU c. The time value $t \in (\mathbb{R}_{\geq 0}, \mathbb{N})$ is denoted by $\lambda m.t$, which is used if all FMUs are at the same time.*
- *$Mode : M \to Modes$ is a function, where $Mode(m)$ denotes the mode of the FMU m. We denote by a value $mode \in Modes$. The function $\lambda m.mode$, which we use if all FMUs are in the same mode.*
- *$S^R_U = \prod_{m\in M} S^R_{U_c}$, $S^R_Y = \prod_{c\in M} S^R_{Y_c}$, $s^R_{U^C} = \prod_{m\in M} S^R_{U^c_m}$, and $s^R_{Y^C} = \prod_{m\in M} S^R_{Y^c_m}$ are functions mapping a variable to a timestamp denoting the time when the variable was last updated.*
- *$S^R_{W^C} = \bigcup_{m\in M} S^R_{W^C_m}$ is the set of all active clocks in the scenario.*

Definition 5 (Consistent State)**.** *The co-simulation state of a given scenario is consistent if:*

$$\begin{aligned} Consistent(\langle t, Modes, s^R_U, s^R_Y, s^R_{U^C}, s^R_{Y^C}, s^R_{W^C}\rangle) &\triangleq (\forall u, y \cdot L(u) = y \implies s^R_{\mathscr{V}}(u) = s^R_{\mathscr{V}}(y)) \\ &\wedge (\forall u^c \in U^C, \exists y^c \in Y^C \cdot L^C(u^c) = y^c) \wedge (\forall u^c, y^c \cdot L^C(u^c) = y^c \implies s^R_{U^C}(u^c) = s^R_{Y^C}(y^c)) \end{aligned}$$

Informally, a consistent state is a state where all coupled input and output variables have the same value and all FMUs are in the same simulation mode and synchronized at the same time t.

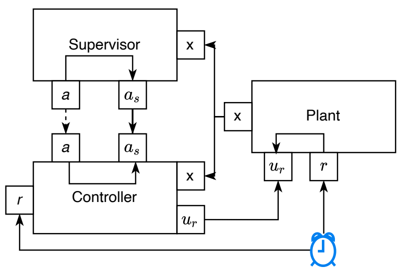
The importer simulates the scenario by executing the orchestration algorithm, which consists of an initialization procedure and a co-simulation step procedure. The initialization procedure sets up and connects the FMUs, while the co-simulation step procedure is responsible for simulating the scenario over time by moving the scenario from a consistent state at time t to a future consistent state at time $t+h$ (see Definition 6). This is done by exchanging values between continuous port variables, computing future states for all FMUs, choosing the appropriate step duration h, and solving all events occurring in the interval $[t; t+h[$. The orchestration algorithm is designed to maintain consistency in the co-simulation state by satisfying coupling restrictions and ensuring that the FMUs move in lock-step. The co-simulation step is executed repeatedly until the simulation ends. We write $s \xrightarrow{\text{P}} s'$ if executing the action P in the state s results in the state s'.

Definition 6 (Co-Simulation Step of an SC Scenario)**.** *A co-simulation step P is a sequence of FMU actions to simulate the scenario by moving it from one consistent state at time t to a consistent state at time $t+h$:*

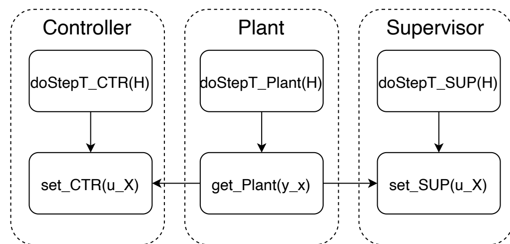
$$\left\langle t, s_U^R, s_Y^R, s_{UC}^R, s_{YC}^R, s_{\mathscr{V}}^R, s_{WC}^R \right\rangle \xrightarrow{P} \left\langle t', s_U^{R\prime}, s_Y^{R\prime}, s_{UC}^{R\prime}, s_{YC}^{R\prime}, s_{\mathscr{V}}^{R\prime}, s_{WC}^{R\prime} \right\rangle \implies$$
$$(Consistent(\left\langle t, s_U^R, s_Y^R, s_{UC}^R, s_{YC}^R, s_{\mathscr{V}}^R, s_{WC}^R \right\rangle) \implies (Consistent(\left\langle t', s_U^{R\prime}, s_Y^{R\prime}, s_{UC}^{R\prime}, s_{YC}^{R\prime}, s_{\mathscr{V}}^{R\prime}, s_{WC}^{R\prime} \right\rangle) \wedge t' \geq t))$$

Definition 2 extends the definition of a co-simulation step for the FMI 2.0 (Hansen et al. 2022) to support the event-driven nature of FMI 3.0. This means that the co-simulation step is responsible for computing the next state of the scenario and detecting and handling events. Nevertheless, the co-simulation step is still a sequence of FMU actions.

The step procedure is computed as the topological ordering of a graph where the actions in Definition 1 form the nodes of the graph and the edges represent the dependencies between the FMU actions (see Definition 7). The graph-based approach originates in (Gomes et al. 2019, Broman et al. 2015) and was further developed in (Hansen et al. 2022) to cover scenarios with cyclic dependencies and step restrictions.



(a) Example of a scenario composed of three SC FMUs. The arrows represent the coupling restrictions, while the dashed arrows represent the couplings between clocks. The clocks variables are represented using internal arrows.



(b) The step operation graph of the scenario in Fig. 1a built using the rules in Definition 7.

Figure 1: Example of a scenario and its step operation graph.

Definition 7 (Step Operation Graph)**.** *Given a co-simulation scenario defined in Definition 2, we define the step operation graph where each node represents an operation* $\texttt{set}_m(\_, u_m, \_)$, $\texttt{stepT}_m(\_, H)$, *or* $\texttt{get}_m(\_, y_m)$, *of some fmu* $m \in M$, $y_m \in Y_m$, *and* $u_m \in U_m$. *The edges are created according to the rules:*

1. *For each* $m \in M$ *and* $u_m \in U_m$, *if* $L(u_m) = y_d$, *add an edge* $\texttt{get}_d(\_, y_d) \to \texttt{set}_m(\_, u_m, \_)$*;*
2. *For each* $m \in M$ *and* $y_m \in Y_m$, *add an edge* $\texttt{stepT}_m(\_, H) \to \texttt{get}_m(\_, y_m)$*;*
3. *For each* $m \in M$ *and* $u_m \in U_m$, *if* $R_m(u_m) = true$, *add an edge* $\texttt{set}_m(\_, u_m, \_) \to \texttt{stepT}_m(\_, H)$*;*
4. *For each* $m \in M$ *and* $u_m \in U_m$, *if* $R_m(u_m) = false$, *add an edge* $\texttt{stepT}_m(\_, H) \to \texttt{set}_m(\_, u_m, \_)$*;*
5. *For each* $m \in M$ *and* $(u_m, y_m) \in D_m$, *add an edge* $\texttt{set}_m(\_, u_m, \_) \to \texttt{get}_m(\_, y_m)$*.*

An example of a scenario composed of SC FMUs is shown in Fig. 1a, its associated orchestration algorithm is shown in Algorithm 1 and the corresponding step operation graph is shown in Fig. 1b. Algorithm 1 is an optimization that takes advantage of the fact that the FMUs are independent and can be executed in parallel.

The step procedure in Algorithm 1 is used to simulate an event-driven system in FMI 3.0, as described in Fig. 1a. However, it only covers a partial implementation of the procedure, as it does not specify how to detect and handle events, which are a crucial aspect of the co-simulation algorithm. In FMI 3.0, simulations are event-driven, meaning that the step procedure can be interleaved with an event-handling procedure that

Algorithm 1 The co-simulation step procedure of the scenario in Fig. 1a.

1: $(s^{(h)}_{CTRL}, s^{(h)}_{SUP}, s^{(h)}_{PLANT}) \leftarrow (\texttt{stepT}_{CTRL}(s^{(0)}_{CTRL}, h), \texttt{stepT}_{SUP}(s^{(0)}_{SUP}, h), \texttt{stepT}_{PLANT}(s^{(0)}_{PLANT}, h))$ ▷ Compute the new state of the FMUs.
2: $(s^{(h)}_{PLANT}, x) \leftarrow \texttt{get}_{PLANT}(s^{(h)}_{PLANT}, y_x)$ ▷ Read the output of the plant.
3: $(s^{(h)}_{SUP}, s^{(h)}_{CTRL}) \leftarrow (\texttt{set}_{SUP}(s^{(h)}_{SUP}, u_x, x), \texttt{set}_{CTRL}(s^{(h)}_{CTRL}, u_x, x))$ ▷ Set the input of the supervisor and the controller.

allows the importer to resolve detected events. The event handling procedure is discussed in more detail in Section 3.2, but it involves bringing relevant FMUs into Event mode and computing the values of affected discrete variables. Discrete variables can only be changed during the event handling procedure, as they cannot change in Step mode. The simulation time in FMI 3.0 follows a super-dense time formulation (Lee and Zheng 2005), with a tuple $t = (t_R, t_I)$ where t_R is the real part of time, and t_I is the integer part. During Step mode, t_R increases while t_I remains at 0, and during Event mode, t_I increases while t_R remains constant. The super-dense time allows a discrete variable to take multiple values at the same real-time instant during the event handling procedure. The following section covers how the importer detects events, computes the step size, and handles events to complete the partial co-simulation step procedure in Algorithm 1.

3 ORCHESTRATION ALGORITHMS

In this section, we present how the approach for synthesizing orchestration algorithms presented in the previous section can be extended to support the FMI 3.0 standard. We can due to space constraints only present the orchestration algorithm for the step mode of the FMI 3.0 standard. Nevertheless, the initialization phase is similar to the initialization phase of the FMI 2.0 standard (Hansen et al. 2022, Hansen et al. 2020) with the only difference being that the FMI 3.0 importer must compute a schedule for all time-based clocks.

3.1 Co-simulation step

The approach presented in Definition 7 is a good starting point for synthesizing the co-simulation step of the FMI 3.0 standard. Nevertheless, the approach is not sufficient as it does not account for the event-driven nature of the FMI 3.0 standard. To account for the event-driven nature of the FMI 3.0 standard, the co-simulation step can be divided into three subsequent phases: event detection and event handling, step size computation, and co-simulation step execution. The following sections describe the two first of these phases, while the third phase is similar to the co-simulation step of the FMI 2.0 present in the previous section.

3.1.1 Detecting Events

The importer detects events and computes the set of affected FMUs by checking the state of the FMUs and the schedule of the time-based clocks. Assuming that the schedule of all time-based clocks is stored in the map `Schedule`, linking all time-based clocks to the next time they should be ticked. The importer can detect all time-based clocks that should be ticked ($W_{Ticking}$) at a given time t_R using $W_{Ticking} = dom(\texttt{Schedule} \rhd T_R)$. State-based events are detected by stepping the FMUs using the function `stepT` as described in Definition 1 and checking the event indicator. Assuming that the importer stores the FMUs that have triggered an event in the set M_A, and that the importer has computed the set of ticking clocks $W_{Ticking}$. An event is detected if either $W_{Ticking} \neq \emptyset$ or $M_A \neq \emptyset$. Their union $(M_A \cup W_{Ticking})$ is referred to as the “event cause”, which is used to determine the event resolution strategy, as described in Section 3.2.

3.2 Handling Events

The importer resolves events by bringing the relevant FMUs into Event mode, exercising them according to an appropriate event strategy, and returning them to Step mode. This process may need to be repeated until all events are resolved. An event strategy consists of steps to resolve an event, including activating clocks, computing discrete equations, sharing data, and updating discrete states. When selecting an event strategy, the importer must consider the event cause, which may be state-based, time-based, or a combination thereof. All possible combinations of event causes, represented as the non-empty powerset of the union of output clocks and time-based clocks ($E_c = \mathscr{P}(U^{TC} \cup Y^C)$), must be considered as only FMUs can determine which output clocks are activated during the event resolution. We use the notation E_A to denote an event cause $E_A \in E_c$. Input clocks are ignored as their connected output determines their activation status. The event strategy is calculated using a graph-based approach based on the following three sets of clocks: *Active clocks*: (W^c_A) is the set of clocks the importer knows to be active. *Potentially active clocks*: (W^c_P) is the set of clocks that can either be active or inactive. *Inactive clocks*: (W^c_I) is the set of clocks that the importer knows to be inactive. Next, we demonstrate how these sets are calculated and how they are used to construct the event-strategy graph using the rules in Definition 8. A concrete example of the approach is provided in Section 3.2.3. The set of active clocks W^c_A consists of the clocks present in the event cause and the input clocks to which they are connected: $W^c_A = E_A \cup dom(L^C \rhd E_A)$, where the set E_A is the set of clocks in the event cause. The potentially active clocks W^c_P are the clocks that could be activated while solving the current event. This set is computed using the following recurrence relation:

$$W^c_{P0} = \{y^c \mid m \in M \wedge y^c \in Y^c_m \setminus W^c_A \wedge W^c_A \cap (Y^c_m \cup U^c_m) \neq \emptyset\}$$
$$W^c_{Pn+1} = W^c_{Pn} \cup dom(L^C \rhd W^c_{Pn}) \cup \{y^c \mid m \in M \wedge y^c \in Y^c_m \setminus W^c_A \wedge W^c_A \cap (Y^c_m \cup U^c_m) \neq \emptyset\}$$

The initial set W^c_{P0} consists of the output clocks of FMUs that have an active clock. This set is iteratively expanded to include transitively connected clocks until a fixed point is reached, which account for all possible combinations of output clocks that may be activated during event resolution. The set of inactive clocks, W^c_I, is simply the clocks that are neither active nor potentially active. The importer must bring all relevant FMUs into Event mode. An FMU is relevant if it has a clock in the set of active or potentially active clocks. The three sets allow the importer to compute the event strategy graph using the rules in Definition 8.

Definition 8 (Event Graph)**.** *Given a co-simulation scenario $\langle M, L, L^C, \mathscr{M}, F, R, V^P_, F^C \rangle$, and an event with the following clock configuration $\langle W^c_A, W^c_P, W^c_I \rangle$. We define the event graph where each node represents an operation $\texttt{set}^c_m(\_, u^c_m, \_)$, $\texttt{set}_m(\_, u_m, \_)$, $\texttt{get}_m(\_, y_m)$, or $\texttt{get}^c_m(\_, y^c_m)$ of some fmu $m \in M$, $y_m \in Y_m$, $y^c_m \in Y^c_m$, $u^c_m \in U^c_m$, and $u_m \in U_m$. The edges are formed by the following rules:*

1. *For each $y^c \in W^c_P \cup W^c_A$ and $u^c \in U^C$ where $L^C(u^c) = y^c$ add an edge $\texttt{get}^c(\_, y^c) \rightarrow \texttt{set}^c(\_, u^c, \_)$;*
2. *For each $u^c \in W^c_P \cup W^c_A$ and $y^c \in Y^C$ where $u^c \in F^C(y^c)$ add an edge $\texttt{set}^c(\_, u^c, \_) \rightarrow \texttt{get}^c(\_, y^c)$;*
3. *For each $y \in Y^D$ and $u \in U^D$ where $V^P(y) \cap (W^c_P \cup W^c_A) \neq \emptyset \wedge L(u) = y$ add an edge $\texttt{get}(\_, y) \rightarrow \texttt{set}(\_, u, \_)$;*
4. *For each $y \in \mathrm{ran}(V^P \lhd (W^c_P \cup W^c_A))$ and input clock $u^c \in dom(W^c_P \cup W^c_A \rhd y)$ add an edge $\texttt{set}^c(\_, u^c, \_) \rightarrow \texttt{get}(\_, y)$;*
5. *For each $y \in \mathrm{ran}(V^P \lhd (W^c_P \cup W^c_A))$ and output clock $y^c \in dom(W^c_P \cup W^c_A \rhd y)$ add an edge $\texttt{get}^c(\_, y^c, \_) \rightarrow \texttt{get}(\_, y)$;*
6. *For each $y^c_{m1} \in Y^c_{m1}$ and $u^c_{m2} \in U^{TC}_{m2}$ where $y^c_{m1} \in W^c_P \cup W^c_A$ and $u^c_{m2} \in W^c_P \cup W^c_A$ and $m1 \neq m2$ add an edge $\texttt{get}^c(\_, y^c_{m1}) \rightarrow \texttt{set}^c(\_, u^c_{m2}, \_)$.*

Rules 1 and 2 connect active or potentially active clocks connected by the clock coupling function L^C or the feedthrough clock function F^C to ensure clocks are triggered in the correct order. The third rule connects discrete inputs and outputs connected by the coupling function L and in the clock partition of an active or potentially active clock to ensure correct computation order. The fourth and fifth rules connect active or potentially active clocks to variables in their clock partition to ensure a clock is activated before its partition variables are computed. The sixth rule ensures that state-based events are handled before timed-based events by connecting the triggered clocks to timed-based clocks of other FMUs. The actions (`stepE` and `nextT`) are not part of the event graph because they are always executed after an event iteration. The event graph accounts for the importer's inability to assume anything about the potentially active clocks. The import must account for both the fact that an output clock $y^c \in W_P^c$ can turn out to be active or inactive and adjust the sets W_A^c, W_P^c and W_I^c and therefore also the event graph according to the observed behaviour.

The event graph is created based on the "worst-case" scenario, where all potentially active clocks become active. For example, assuming that the clock $y^c \in W_P^c$ is active, it is removed from the set W_P^c and added to the set of W_A^c together with all input clocks connected to it. Since the event graph is created based on the set $W_P^c \cup W_A^c$, we do not need to adjust the event graph. Contrary, if the clock $y^c \in W_P^c$ is inactive such that $y^c \in W_I^c$, we need to adjust the event graph by removing the clock y^c from the set W_P^c, whereafter we recalculate the sets W_P^c and W_I^c before constructing a new event graph g_1 using the rules in Definition 8 with the updated sets. The graph g_1 is used to compute the event strategy for the case where the clock is inactive.

All $\texttt{get}^c$ actions of output clocks in W_P^c are so-called "split actions" since they "split" the event graph in two, one where the clock is active and one where it is inactive. A split action resembles a conditional statement, which allows the importer to react to the observed behaviour; see Algorithm 3 for an example.

3.2.1 Algorithm for Synthesizing the Event Strategy

The synthesis of an event strategy for a specific event cause is presented in Algorithm 2. This algorithm employs the sets W_A^c, W_P^c, and W_I^c and begins by computing the event graph of the given event cause E_A. Subsequently, Tarjan's algorithm is used to sort the event graph to determine the order of actions. The *SYNTHESIZE* procedure is then utilized to synthesize the event strategy by recursively processing the action sequence *sccs*, accounting for split actions with the conditional statement in Line 13. For split actions, where an output clock can be active or inactive, two event strategies $\mathscr{A}_A$ and $\mathscr{A}_I$ are generated by calling *SYNTHESIZE* recursively with the two corresponding event graphs. The two event strategies are combined using a "conditional" action" in Line 19 to produce the final event strategy. All other actions are handled in the else branch (Lines 22 and 23) by adding the action to the event strategy before moving on to the next action. The event strategies are stored in the map M (Line 6) and retrievable by their event cause.

Algorithm 2 Algorithm for Synthesizing the Event Strategies

1: **for all** $E_A \in E_c$ **do**
2: $\quad (W_A^c, W_I^c, W_P^c) \leftarrow computeClocks(E_A)$ ▷ Compute the different sets of clocks
3: $\quad eventGraph \leftarrow createGraph(W_A^c, W_P^c, W_I^c)$ ▷ Create event graph
4: $\quad sccs \leftarrow tarjan(eventGraph)$ ▷ Sort the event graph
5: $\quad A_E \leftarrow Synthesize(W_A^c, W_P^c, W_I^c, [], sccs)$ ▷ Compute the event strategy
6: $\quad M \leftarrow M \cup \langle E \mapsto A_E \rangle$ ▷ Add the event strategy to the map
7: **end for**
8: **procedure** SYNTHESIZE($W_A^c, W_P^c, W_I^c, \mathscr{A}, sccs$)
9: $\quad$ **if** $sccs = []$ **then**
10: $\quad\quad$ **return** $\mathscr{A}$ ▷ Return the event strategy
11: $\quad$ **else**
12: $\quad\quad scc \leftarrow head(sccs)$
13: $\quad\quad$ **if** $scc = \texttt{get}^c$ **then** ▷ A split action!
14: $\quad\quad\quad \mathscr{A}_a \leftarrow Synthesize(W_A^c \cup scc, W_P^c \setminus scc, W_I^c, [], tail(sccs))$ ▷ Assuming $scc \in W_A^c$
15: $\quad\quad\quad eventGraph \leftarrow createGraph(W_A^c, W_P^c \setminus scc, W_I^c \cup scc)$ ▷ Create a new graph.
16: $\quad\quad\quad eventGraph_R \leftarrow (eventGraph \cup scc) \setminus \mathscr{A}$ ▷ Remove existing actions.
17: $\quad\quad\quad sccs_I \leftarrow tarjan(eventGraph_R)$ ▷ Sort the new graph.
18: $\quad\quad\quad \mathscr{A}_i \leftarrow Synthesize(W_A^c, W_P^c \setminus scc, W_I^c \cup scc, [])$ ▷ Assuming $scc \in W_I^c$
19: $\quad\quad\quad \mathscr{A} \leftarrow \mathscr{A} \oplus Case(scc, \mathscr{A}_a, \mathscr{A}_i)$ ▷ Make a split action.
20: $\quad\quad\quad$ **return** $\mathscr{A}$
21: $\quad\quad$ **else** ▷ A non-split action!
22: $\quad\quad\quad \mathscr{A} \leftarrow \mathscr{A} \oplus scc$ ▷ Add the action to the event strategy
23: $\quad\quad\quad$ **return** $Synthesize(W_A^c, W_P^c, W_I^c, \mathscr{A}, tail(sccs))$ ▷ Treat next node.
24: $\quad\quad$ **end if**
25: $\quad$ **end if**
26: **end procedure**

3.2.2 Updating the schedule and discrete states

After executing the event strategy, the importer needs to update the schedule of all affected time-based clocks using the `nextT`-action. The affected time-based clocks are a subset of the timed-based clocks belonging to a relevant FMU. The importer should also update the states of all activated FMUs using the `stepE`-action while monitoring if any of them invoke an event. If an event is invoked, the importer should update the set M_A accordingly. Once all FMUs have been stepped, the importer updates the superdense time and checks if any events have been invoked. If an event has occurred, a new event iteration is needed, and the importer must decide the next event strategy to execute by looking up the event strategy in the map M using the event cause as key. If no events have been invoked, the importer brings all FMUs back to simulation mode and performs the remaining part of the co-simulation step, which is already explained in Section 3.1.

3.2.3 An Example of Computing the Event Strategy

The approach is illustrated using the scenario in Fig. 1a, a scenario with different types of clocks that the importer must account for. We start by calculating the set E_c of all possible event causes for the scenario: $E_c = \{\{u_r^c, y_s^c\}, \{u_r^c\}, \{y_s^c\}\}$. The next step is to compute the event strategy for each event entrance. We will showcase the approach of computing an event strategy for the event entrance defined by the following set of clocks $\{y_s^c, u_r^c\}$. The initial sets of clocks are $W_A^c = \{u_r^c\}$ $W_P^c = \{u_r^c, y_s^c\}$ $W_I^c = \emptyset$. The event graph seen in Fig. 4 on page 13 is computed using the rules in Definition 8. The graph is a DAG where the event strategy is the topological ordering of the graph. The set of potentially active clocks contains an output clock (y_s^c), indicating that we need to account for the fact that the clock y_s^c can turn out to be either active or inactive when we query it - resembling a conditional statement in the event strategy shown in Algorithm 3. The other event strategies for the scenario is shown in Algorithms 4 and 5 on page 12.

Algorithm 3 The Event Strategy for the event entrance $\{y_s^c, u_r^c\}$ of the scenario in Fig. 1a

$b_s \leftarrow \mathtt{get}^c_{SUP}(s^{(t)}_{SUP}, y^c_s)$
if b_s **then** ▷ Check if clock y_s^c is active
  $a\_s_v \leftarrow \mathtt{get}_{SUP}(s^{(t)}_{SUP}, y_{a\_s})$
  $s^{(t)}_{CTRL} \leftarrow \mathtt{set}^c_{CTRL}(s^{(t)}_{CTRL}, u^c_s, \mathit{defined})$
  $s^{(t)}_{CTRL} \leftarrow \mathtt{set}_{CTRL}(s^{(t)}_{CTRL}, u_{a\_s}, a\_s_v)$
end if
$s^{(t)}_{CTRL} \leftarrow \mathtt{set}^c_{CTRL}(s^{(t)}_{CTRL}, u^c_r, \mathit{defined})$ ▷ Activate clock.
$s^{(t)}_{PLANT} \leftarrow \mathtt{set}^c_{PLANT}(s^{(t)}_{PLANT}, u^c_r, \mathit{defined})$ ▷ Activate clock.
$u\_r_v \leftarrow \mathtt{get}_{CTRL}(s^{(t)}_{CTRL}, y_{u\_r})$
$s^{(t)}_{PLANT} \leftarrow \mathtt{set}_{PLANT}(s^{(t)}_{PLANT}, u^c_{u\_r}, u\_r_v)$

4 VALIDATION

The approach is validated through a case study of Dynamic State Estimation (DSE) in power systems. DSE is a crucial task for improving the stability, control, and performance of protection systems in power systems, which consist of interconnected physical and cyber components with real-time data from Phasor Measurement Units (PMUs) and Remote Terminal Units (RTUs) transmitted to a central control centre at various scan rates. Synchronizing this large amount of data is challenging due to the size of the power grid. DSE addresses this issue by dividing the computational burden between multiple computational centres. However, different centres may use different solvers and time scales for their algorithms, making the orchestration of these algorithms a critical task (Aweya and Al Sindi 2013).

The case study is an IEEE 14-bus power system (Christie 2023) with constant loads partitioned into five subsystems each has an PMU installed to send measurements to the two DSE centres (DSE1 and DSE2) at different rates. DSE1 and DSE2 use different solvers to estimate the state variables of angular rotor positions and angular frequencies of the power system, and the results are sent to a Frequency Controller

(FC) responsible for adjusting the input power to the generators to stabilize the power system frequency. The FMI representation comprises eight SC FMUs as shown in Fig. 2.

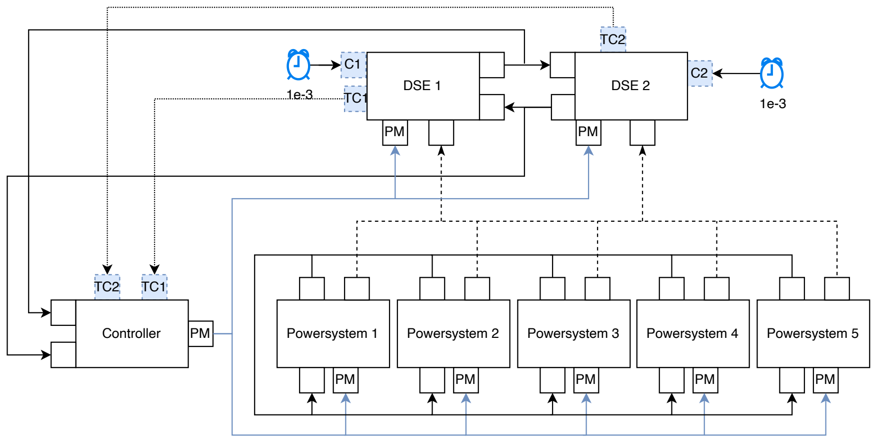


Figure 2: The power system represented in FMI 3.0. Two time-based clocks are connected to the DSE centers to sample the power system at a rate determined by the timed-based clocks $C1$ and $C2$. There are two discrete connections (one in each direction) between the DSE centers to enable a distributed estimation of the state variables. The data flow between the DSE centers is triggered using a triggered clock to ensure only relevant data is being transmitted. The data is sent to the controller using a discrete connection, which is the reason for the connection (data and clock) between the DSE centers and the controller.

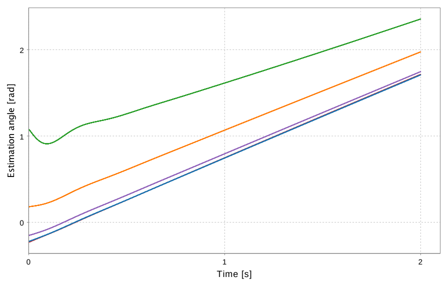
The case study is a good candidate for validating the proposed algorithm as it is a realistic scenario with a complex interaction including both state-based and time-based events, and different rates of communication between the FMUs. The orchestration algorithm must account for different rates in the system - the DSEs have different rates controlled by their connected timed-based clock. Additionally, event-based communication is employed between the DSEs and the FC to reduce the unnecessary data transmission.

We simulated the scenario using virtual FMUs (Scala classes implementing the FMI 3.0 SC API) rather than compiled FMUs, due to the lack of tool support for the FMI 3.0. The virtual FMUs share the same API as compiled FMUs and can be used similarly. The simulation results shown in Fig. 3 were validated against a reference model developed by a domain expert.

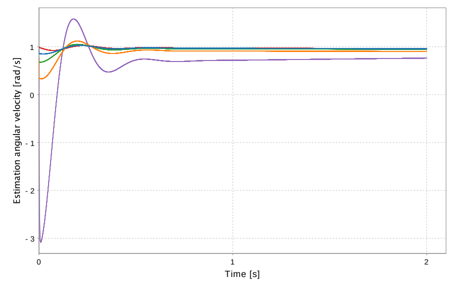
The reference model, the prototype implementation and the virtual FMUs and more details on the case study are available at https://github.com/SimplisticCode/ANNSIM_Reprod. The results demonstrate that the proposed algorithm can accurately simulate a complex system with different simulation rates, and event-based communication and behavior. The quality of the reference model is beyond the scope of this work.

5 SUMMARY

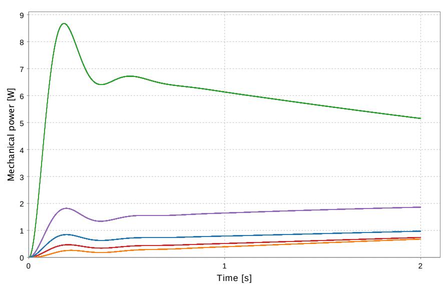
This paper presented what we believe to be the first approach for synthesizing orchestration algorithms for synchronous clocks simulations according to the FMI 3.0 standard. The proposed algorithms are graph-based and are based on earlier work on the FMI 2.0 standard, which has been extended to support the event-driven nature of the FMI 3.0 standard. The methods have been implemented in the FMI 3.0 reference implementation and have successfully simulated several FMUs with synchronous clocks. Nevertheless, the developed tool is still a prototype and is not ready for industrial use yet. Future work will focus on extending the algorithms to support the scheduled execution feature of the FMI 3.0 standard and to facilitate the integration of the algorithms into existing simulation tools.



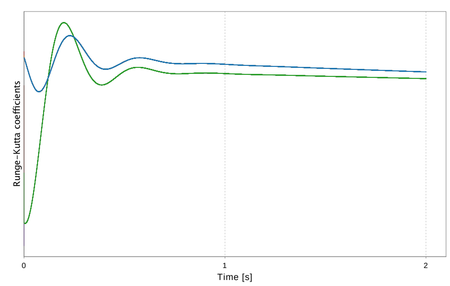
(a) Each color shows the estimated angular velocity of different synchronous machines 1 to 5 computed by the DSE centres.



(b) Each color shows the estimated angular velocity of different synchronous machines 1 to 5 computed by the DSE centres.



(c) Each color shows the calculated input mechanical power of different synchronous machines 1 to 5.



(d) Each color shows a different of the Runge-Kutta parameters that the DSE center uses to calculate the state variables and send to the controller for it to calculate the input mechanical power .

Figure 3: Simulation results of the powersystem in FMI 3.0 SC. Each subfigure shows the results of a different simulation parameter and each line represents a different FMU/Generator.

ACKNOWLEDGMENTS

The authors are grateful to the Poul Due Jensen Foundation, which has supported the establishment of the Centre for Digital Twin Technology at Aarhus University. The authors would like to thank the anonymous reviewers for their helpful comments.

AUTHOR BIOGRAPHIES

SIMON THRANE HANSEN is a PhD student at Aarhus University, Denmark. His research interests lie in the area of co-simulation and formal verification. His email address is sth@ece.au.dk.

CLÁUDIO GOMES is an assistant professor at Aarhus University. He received his PhD at the University of Antwerp, for his work on the foundations of co-simulation. His email address is claudio.gomes@ece.au.dk.

ZAHRA KAZEMI is a postdoctoral researcher at Aarhus University She received her PhD at Shiraz University for her work in Control Engineering. Her email address is zka@ece.au.dk.

A NOTATION

We borrow some notation from the Event-B method that may be less familiar. The complement $-c$ of a set $c \subseteq \Gamma$ is defined as the set of elements in Γ that are not in c. We define the domain restriction $c \lhd R$ as the subset of R consisting of all pairs (x,y) such that $x \in c$ and $(x,y) \in R$. The domain subtraction $c \mathbin{⩤} R$ is defined as the complement of c within the domain of R, i.e., $(-c) \lhd R$. The range restriction $R \rhd c$ is defined as the inverse of the domain restriction of the inverse of R on c, i.e., $(c \lhd R^{-1})^{-1}$. Similarly, the range subtraction $R \mathbin{⩥} c$ is defined as the range restriction of the complement of c, i.e., $R \rhd (-c)$. The relational image $R[c]$ is defined as the range of the domain restriction of R on c, i.e., ran, $(c \lhd R)$, where ran denotes the range of a relation. The operations are exemplified on the relation $R = \{(1,2),(2,3),(3,4)\}$ below:

$$
\begin{aligned}
\{1\} \lhd R &= \{(1,2)\} \\
R \rhd \{4\} &= \{(3,4)\} \\
dom(R) &= \{1,2,3\} \\
R^{-1} &= \{(2,1),(3,2),(4,3)\}
\end{aligned}
\qquad
\begin{aligned}
\{1\} \mathbin{⩤} R &= \{(2,3),(3,4)\} \\
R \mathbin{⩥} \{4\} &= \{(1,2),(2,3)\} \\
\mathrm{ran}(R) &= \{2,3,4\}
\end{aligned}
$$

B EVENT STRATEGIES

This appendix presents the two event strategies for the scenario Fig. 1a.

Algorithm 4 The Event Strategy for the event entrance $\{u_r^c\}$ of the scenario in Fig. 1a.

1: $s_{CTRL}^{(t)} \leftarrow \mathtt{set}_{CTRL}^{c}(s_{CTRL}^{(t)}, u_r^c, \mathit{defined})$ ▷ Activate clock.
2: $u\_r_v \leftarrow \mathtt{get}_{CTRL}(s_{CTRL}^{(t)}, y_{u\_r})$
3: $s_{PLANT}^{(t)} \leftarrow \mathtt{set}_{PLANT}(s_{PLANT}^{(t)}, u_{u\_r}^c, u\_r_v)$

Algorithm 5 The Event Strategy for the event entrance $\{y_s^c\}$ of the scenario in Fig. 1a.

1: $b_s \leftarrow \mathtt{get}_{SUP}^{c}(s_{SUP}^{(t)}, y_s^c)$
2: **if** b_s **then** ▷ Check if clock y_s^c is active
3: $\quad a\_s_v \leftarrow \mathtt{get}_{SUP}(s_{SUP}^{(t)}, y_{a\_s})$
4: $\quad s_{CTRL}^{(t)} \leftarrow \mathtt{set}_{CTRL}^{c}(s_{CTRL}^{(t)}, u_s^c, \mathit{defined})$
5: $\quad s_{CTRL}^{(t)} \leftarrow \mathtt{set}_{CTRL}(s_{CTRL}^{(t)}, u_{a\_s}, a\_s_v)$
6: **end if**

REFERENCES

Modelica Association 2021. “Modelica Language Specification Version 3.5”. Online at https://modelica.org/documents/MLS.pdf.

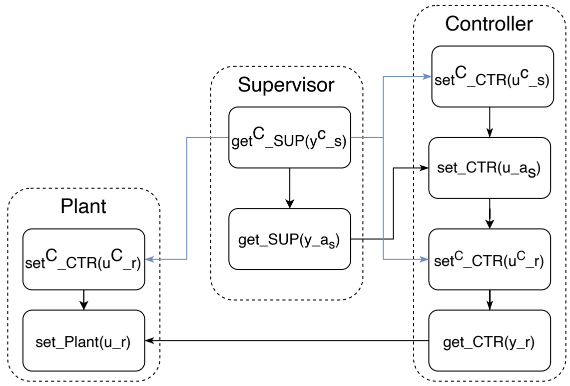


Figure 4: Event graph for the event cause $\{y_s^c, u_r^c\}$ of the scenario in Fig. 1a. The synthesized event strategy is shown in Algorithm 3. The different colors in the graph is used to distinguish crossing edges. The edge between the clock y_s^c in the supervisor and the clock u_r^c in the controller and in the plant ensures that the equation inside the supervisor is executed before the equations inside the controller and the plant.

Aweya, J., and N. Al Sindi. 2013. “Role of Time Synchronization in Power System Automation and Smart Grids”. In *2013 IEEE ICIT*.

Benveniste, A., P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. 2003. “The Synchronous Languages 12 Years Later”. *Proceedings of the IEEE* vol. 91 (1).

Blockwitz, T., M. Otter, J. Åkesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. 2012. “Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models”. In *Proc. 9th International Modelica Conference*, Linköping University Electronic Press.

Broman, D., L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. 2015. “Requirements for Hybrid Cosimulation Standards”. In *Proc. HSCC ’15*, ACM, ACM New York, NY, USA.

Richard D. Christie 2023, March. “Power Systems Test Case Archive - UWEE”. [Accessed 8. Mar. 2023].

Functional Mockup Interface Steering Committee 2021. “Functional Mock-up Interface for Model Exchange, Co-Simulation, and Scheduled Execution”.

Dahmann, J. S. 1997. “High Level Architecture for Simulation”. In *DIS-RT*, IEEE Computer Society.

Elmqvist, H., M. Otter, and S. E. Mattson. 2012. “Fundamentals of Synchronous Control in Modelica”. In *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press.

FMI 2014. “Functional Mock-up Interface Tools”. Online https://fmi-standard.org/tools/.

Galtier, V., S. Vialle, C. Dad, J.-P. Tavella, J.-P. Lam-Yee-Mui, and G. Plessis. 2015. “FMI-based distributed multi-simulation with DACCOSIM”. In *SpringSim (TMS-DEVS)*, edited by F. Barros, M. H. Wang, H. Prähofer, and X. H. 0002, SCS/ACM.

Gomes, C., L. Lucio, and H. Vangheluwe. 2019. “Semantics of Co-Simulation Algorithms with Simulator Contracts”. In *Proc. ACM/IEEE MODELS’19*, edited by L. Burgueño, A. Pretschner, S. Voss, M. Chaudron, J. Kienzle, M. Völter, S. Gérard, M. Zahedi, E. Bousse, A. Rensink, F. Polack, G. Engels, and G. Kappel. , IEEE.

Gomes, C., C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe. 2019. “Co-Simulation”. *ACM Computing Surveys* vol. 51 (3).

Hansen, S. T., C. Gomes, P. G. Larsen, and J. van de Pol. 2021. "Synthesizing Co-Simulation Algorithms with Step Negotiation and Algebraic Loop Handling". In *Proc. ANNSIM'21*, edited by C. R. Martin, M. J. Blas, and A. Inostrosa-Psijas, IEEE.

Hansen, S. T., C. 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. 2022, January. "The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations". *Electronics* vol. 11 (21).

Hansen, S. T., C. Thule, and C. Gomes. 2020. "An FMI-Based Initialization Plugin for INTO-CPS Maestro 2". In *SEFM'2020 Collocated Workshops.*, edited by L. Cleophas and M. Massink, Springer.

Hansen, S. T., C. Thule, C. Gomes, J. van de Pol, M. Palmieri, E. O. Inci, F. Madsen, J. Alfonso, J. Á. Castellanos, and J. M. Rodriguez. 2022. "Verification and synthesis of co-simulation algorithms subject to algebraic loops and adaptive steps". *STTT* vol. 24 (6).

Kofman, E., and S. Junco. 2001. "Quantized-State Systems: A DEVS Approach for Continuous System Simulation". *SIMULATION* vol. 18 (3).

Kubler, R., and W. Schiehlen. 2000. "Two Methods of Simulator Coupling". *Mathematical and Computer Modelling of Dynamical Systems* vol. 6 (2).

Lee, E. A., and H. Zheng. 2005. "Operational Semantics of Hybrid Systems". In *Proc. HSCC 2005*, LNCS, Springer Berlin Heidelberg.

Paris, T., J. Wiart, D. Netter, and V. Chevrier. 2019. "Teaching co-simulation basics through practice". In *Proc. SummerSim'19*, edited by U. Durak, ACM.

Zeigler, B. P. 1976. *Theory of Modeling and Simulation*. John Wiley, Hoboken, NJ, USA.

Zeigler, B. P., and J. S. Lee. 1998, August. "Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment". In *Enabling Technology for Simulation Science II*, Volume 3369. Orlando, FL, United States, SPIE 3369.