Speeding Up AC Circuit Co-Simulations Through Selective Simulator Decoupling of Predictable States

López, Claudio David; Cvetković, Miloš; Palensky, Peter

DOI
10.1109/ACCESS.2019.2907773

Publication date
2019

Document Version
Final published version

Published in
IEEE Access

Received March 6, 2019, accepted March 22, 2019, date of publication March 27, 2019, date of current version April 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2907773

Speeding Up AC Circuit Co-Simulations Through Selective Simulator Decoupling of Predictable States

CLAUDIO DAVID LÓPEZ, (Member, IEEE), MILOŠ CVETKOVIĆ, AND PETER PALENSKY, (Senior Member, IEEE)
Department of Electrical Sustainable Energy, Delft University of Technology, 2628 Delft, The Netherlands
Corresponding author: Claudio David López (c.d.lopez@tudelft.nl)

ABSTRACT Co-simulation has become increasingly popular as a tool for dealing with the unprecedented complexity of modern engineering systems, such as electrical power systems and the AC circuits that compose them. Co-simulation is useful when migrating the models of each subsystem to a single monolithic simulator is either impractical or impossible, and the need for understanding the interactions between the subsystems does not leave room for model simplifications. However, co-simulation can suffer from long execution times, caused by the overhead introduced by exchanging variables between simulators. In this paper, we propose a method that mitigates this overhead by decoupling the simulators whenever their inputs become predictable. We applied this method to the co-simulation of an AC circuit composed of two subsystems and obtained speedups of up to 39% with errors that remain around 1% most of the time. Although questions regarding the scalability of the method persist, these results indicate that the method has the potential to make co-simulation an even more valuable tool for the user.

INDEX TERMS AC systems, co-simulation, electromagnetic transient, power systems, simulation.

I. INTRODUCTION

Co-simulation has experienced a surge in popularity as a method for tackling the unprecedented complexity of modern engineering systems, such as electrical power systems and the AC circuits that compose them. In a co-simulation, a system is simulated using a set of simulators (or simulation tools), each tasked with simulating only a subsystem of the larger system. These simulators collaborate with each other by exchanging a set of selected variables, called interface variables, at run time. In many cases, the execution of a co-simulation is orchestrated by a so called co-simulation master, tasked with keeping the participating simulators synchronized and providing them with the input variables they require.

The ability to couple different simulators in a co-simulation setting is useful when migrating the models of each subsystem to a single monolithic simulator is either impractical or impossible. This happens, for example, when there is a need to simulate subsystems that have been modeled using different tools or languages, when the models cannot be shared due to privacy or intellectual property restrictions, or when the available tools are not compatible with one of the models (e.g., a continuous simulator and a discrete event model). Situations such as these become commonplace as the complexity of the systems under analysis increases, and the need to understand the way their subsystems interact with each other does not allow for model simplifications [1]. In the case of AC circuits, such as those found in electrical power systems, co-simulation has been advantageous for coupling models that use different circuit representations (e.g., phasor and point on wave [2], three phase and three sequence [3]), power grids of neighboring geographical areas [4], transmission and distribution grids [5], and to couple circuit models to models from other domains, such as communication systems [6] and energy markets [7]. Despite their inherent advantages, co-simulations can suffer from long execution times, even if the participating simulators are executed in parallel [4].

Co-simulation requires exchanging and processing interface variables. These operations introduce overhead that can significantly impact the total execution time of the co-simulation, depending on how fast (or slow) they are, and how often they are executed. Naturally, one way of mitigating the overhead introduced by exchanging interface variables

The associate editor coordinating the review of this manuscript and approving it for publication was Salvatore Favuzza.

is to reduce the frequency of these exchanges, nevertheless, reducing the frequency beyond a certain point has a negative impact on the accuracy and numerical stability of the co-simulation [1].

In this paper we propose a method for speeding up co-simulations by selectively decoupling the participating simulators. This method is based on the premise that the trajectories followed by the interface variables can be predicted at run time, at least during a portion of the co-simulation. Hence, as long as the interface variables remain in this predictable state, it is not necessary to exchange them, since each simulator should be able to predict them. As a consequence, the overhead introduced by exchanging interface variables is reduced. To define and implement the selective decoupling co-simulation method we assume that the simulators are black boxes and that the only information available to determine when to decouple and recouple them is the interface variables themselves.

To test our method we co-simulate an AC circuit composed of two subsystems, that represents a simple electrical power system. In the tests we obtained speedups of up to 39% with respect to a traditional co-simulation, with errors with respect to a monolithic simulation that remain around 1% most of the time. This shows that it is possible to speed up a co-simulation through simulator decoupling. However, questions regarding the scalability of the method with respect to the number of coupled subsystems, and and its applicability to physical systems other than AC circuits remain open.

This paper is structured as follows: Section II discusses how co-simulation overhead affects the total execution time, Section III introduces the selective decoupling method for reducing co-simulation overhead, Section VI describes how the method can be applied to AC circuit co-simulation, Section V evaluates the method and discusses the results, and Section VI concludes the paper.

II. THE CHALLENGE OF CO-SIMULATION OVERHEAD

Let us consider a system modeled with the initial value problem

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), \quad \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}), \ \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \tag{1}$$

where $\boldsymbol{u}$ are the system inputs, $\boldsymbol{x}$ are the state variables, $\boldsymbol{y}$ are the system outputs, and $\boldsymbol{f}$ and $\boldsymbol{g}$ are vector-valued functions. If we split this system into two subsystems, A and B, modeled with

$$\dot{\boldsymbol{x}}_\mathrm{A} = \boldsymbol{f}_\mathrm{A}(\boldsymbol{x}_\mathrm{A}, \boldsymbol{u}_\mathrm{A}), \quad \boldsymbol{y}_\mathrm{A} = \boldsymbol{g}_\mathrm{A}(\boldsymbol{x}_\mathrm{A}, \boldsymbol{u}_\mathrm{A}), \ \boldsymbol{x}_\mathrm{A}(t_0) = \boldsymbol{x}_{\mathrm{A}_0}, \tag{2}$$

$$\dot{\boldsymbol{x}}_\mathrm{B} = \boldsymbol{f}_\mathrm{B}(\boldsymbol{x}_\mathrm{B}, \boldsymbol{u}_\mathrm{B}), \quad \boldsymbol{y}_\mathrm{B} = \boldsymbol{g}_\mathrm{B}(\boldsymbol{x}_\mathrm{B}, \boldsymbol{u}_\mathrm{B}), \ \boldsymbol{x}_\mathrm{B}(t_0) = \boldsymbol{x}_{\mathrm{B}_0}, \tag{3}$$

where the outputs of one subsystem are the inputs of the other, the results of the monolithic simulation of (1) can be approximated by a co-simulation of (2) and (3) by enforcing the coupling equations

$$\boldsymbol{u}_\mathrm{A} = \boldsymbol{y}_\mathrm{B} \qquad \boldsymbol{u}_\mathrm{B} = \boldsymbol{y}_\mathrm{A}, \tag{4}$$

at every point in a discrete time grid $\boldsymbol{t} := \{t_0, t_1, \ldots t_k, \ldots t_K\}$. This time grid defines when the simulators should exchange interface variables. The interval $[t_k, t_{k+1}[$ is known as the co-simulation macro time step and $H_k := t_{k+1} - t_k$ as the macro time step size. Although there are accuracy and performance benefits to having a dynamically-adjusted macro time step size [8], for simplicity it is often chosen to be $H_k = H$ a constant. Within each macro time step, each simulator can perform several micro time steps. To solve each micro time step in the k^{th} macro time step, the simulators approximate the inputs of each subsystem $\boldsymbol{u}_\mathrm{A}(t)$ and $\boldsymbol{u}_\mathrm{B}(t)$ with the k^{th} interpolation polynomials $\tilde{\boldsymbol{u}}_{\mathrm{A},k}(t)$ and $\tilde{\boldsymbol{u}}_{\mathrm{B},k}(t)$, which are obtained from the history of interface variables exchanged until t_k.

From the description above it follows that in the monolithic simulation of (1), the total execution time is the time it takes to solve the model equations from t_0 to t_K (solver time), whereas in the co-simulation of (2) and (3) the total execution time also includes the time it takes to construct and evaluate $\tilde{\boldsymbol{u}}_{\mathrm{A},k}(t)$ and $\tilde{\boldsymbol{u}}_{\mathrm{B},k}(t)$ (interpolation time) and the time it takes to enforce (4) (communication time). We will refer to the time spent on interpolation and communication-related operations as *co-simulation overhead*.

To understand how co-simulation overhead affects total execution time, let us now consider a simple case in which subsystems A and B are solved with a constant micro time step of size h, the solver time per micro time step is T_h, and the overhead per macro time step is T_O. Then, the time it takes to solve one macro time step in a parallel co-simulation is

$$T_H = T_h \frac{H}{h} + T_\mathrm{O}, \tag{5}$$

and the execution time of the entire co-simulation as a function of H is

$$T_\mathrm{CS}(H) = T_H \frac{t_K - t_0}{H} = (t_K - t_0)\left[\frac{T_h}{h} + \frac{T_\mathrm{O}}{H}\right]. \tag{6}$$

As (6) indicates, T_CS increases rapidly as H decreases ($\lim_{H\to 0} T_\mathrm{CS}(H) = \infty$). This means that co-simulations of systems that have fast dynamics and require frequent communication between subsystems can have their performance heavily penalized. Such is the situation of electromagnetic transient models of AC circuits. This effect is especially noticeable when the overhead is large with respect to the solver time, which is the case when the models are small and/or the solvers are fast, or when the interpolation of inputs or communication between simulators is slow. The presence of this overhead is why co-simulations can be slower than monolithic simulations of the same model, even if the subsystems are co-simulated in parallel [4].

III. REDUCING CO-SIMULATION OVERHEAD THROUGH SELECTIVE SIMULATOR DECOUPLING

As discussed in Section II, the total execution time of a co-simulation can be reduced by reducing the total co-simulation overhead. Since the co-simulation user has limited control over the performance of the operations that cause this overhead, an alternative is to reduce the total number of times these operations are performed. If we assume that this cannot be achieved by increasing the macro time step size without compromising the accuracy of the results, we could further reduce the co-simulation overhead if each simulator had the capability of predicting its own inputs, so that the interface variables do not need to be exchanged.

A. PREDICTABILITY OF INTERFACE VARIABLES

Intuitively, we can argue that the predictability of the interface variables changes as the co-simulation runs. For example, when the system is in steady state it is easier to guess what the outputs of each subsystem will be on the next macro time step. On the contrary, guessing the outputs of each simulator becomes more difficult when the system is experiencing a fast transient. If the simulators could identify, at run time, a closed-form expression or set of expressions that describes the trajectories followed by the interface variables over time, each simulator could effectively predict its own inputs. We will refer to these expressions as *trajectory models*. Using the concept of trajectory models we can now introduce the more precise Definitions 1 and 2 for predictable and unpredictable interface variables.

Definition 1 (Predictable interface variables): Interface variables are considered predictable when their trajectories can be computed with sufficient accuracy from a given trajectory model or set thereof.

Definition 2 (Unpredictable interface variables): Interface variables are considered unpredictable if they do not comply with Definition 1.

Note that according to Definitions 1 and 2 interface variables are classified as predictable or unpredictable based on an available trajectory model or set of models, not on whether those models exist. This distinction has as consequence that the same trajectory could be classified as predictable or unpredictable depending on the trajectory model identification method used. Furthermore, what constitutes sufficient accuracy is entirely dependent on the requirements of the co-simulation application.

B. TWO MODES OF CO-SIMULATION OPERATION

Given that the predictability of the interface variables changes throughout the co-simulation, the co-simulation should be able to operate in two different modes and transition between them as needed.

The first mode is the *coupled mode*. The co-simulation operates in this mode when the interface variables are considered unpredictable. In this mode, the simulators exchange interface variables at every macro time step. During the k^{th} macro time step, subsystem s is simulated using as inputs the k^{th} interpolation polynomials $\tilde{\boldsymbol{u}}_{s,k}(t)$, where $t \in [t_k, t_{k+1}[$ and $s \in \boldsymbol{s}$ the set of all subsystems. The coupled mode is the default mode of operation.

The second mode is the *decoupled mode*. The co-simulation operates in this mode when the interface variables are considered predictable. In this mode, the simulators do not exchange variables, but predict their own inputs using trajectory models. For a decoupled mode that starts on the k^{th} macro time step and lasts κ macro time steps, subsystem s is simulated using as inputs the k^{th} trajectory models $\hat{\boldsymbol{u}}_{s,k}(t)$, $t \in [t_k, t_{k+\kappa}]$ and $s \in \boldsymbol{s}$.

This bimodal co-simulation poses two main challenges. The first one is to identify appropriate trajectory models. The second one is to seamlessly transition between modes. In this section we only deal with the second challenge, since the first one is application-dependent. Section IV deals with the first challenge for the case of AC circuit co-simulation.

C. SIMULATOR DECOUPLING

Any pair of coupled simulators can be decoupled if the interface variables they share follow predictable trajectories. One way of determining when an interface variable is following a predictable trajectory is to attempt to identify its trajectory model and to measure the deviation between the trajectory this model predicts and the true trajectory. If the deviation falls below a given threshold, the trajectory can be considered predictable in the sense of Definition 1. Since the inputs of a subsystem are the outputs of another, we can express this idea either in terms of inputs or outputs. Thus, any output $y \in \boldsymbol{y}_s$, $s \in \boldsymbol{s}$ can be considered predictable if

$$\max \left| \frac{\hat{y}(t) - y(t)}{\max \hat{y}(t) - \min \hat{y}(t)} \right| < \epsilon_{\mathrm{p}}, \quad t \in \boldsymbol{t}_{\mathrm{w}}, \tag{7}$$

where $\boldsymbol{t}_{\mathrm{w}} := \{t_{k-N_{\mathrm{s}}+1}, t_{k-N_{\mathrm{s}}+2} \ldots t_k\}$ is a discrete moving time window of length N_{s} samples and duration T_{w}, $\hat{y}$ is the trajectory model of y, and ϵ_{p} is the allowed normalized deviation. The number of samples N_{s} should be selected according to the needs of the trajectory model identification method.

Note that (7) measures deviation relative to the dynamic range of the trajectory model. Using a relative deviation measure simplifies the choice of a suitable ϵ_{p}. Using the dynamic range instead of $\hat{y}(t_k)$ or $y(t_k)$ prevents that (7) becomes indeterminate when the outputs approach zero. One caveat is that constantly recomputing $\hat{y}$ to evaluate (7) can be computationally expensive if H is small. This means that the window hop size R_{w}, that is, the number of samples $\boldsymbol{t}_{\mathrm{w}}$ moves every time (7) is evaluated, might need to be adjusted. An $R_{\mathrm{w}} = 1$ requires (7) to be evaluated at every macro time step, which incurs the highest computational expense. Higher values of R_{w} reduce the computational expense but might delay the transition to decoupled mode. However, this should not negatively impact the accuracy of the co-simulation, only its total execution time.

D. SIMULATOR RECOUPLING

Any pair of decoupled simulators must be recoupled if one of the interface variables they share stops following a predictable trajectory, which in the sense of Definition 2 happens when the trajectory model in place is no longer representative of the interface variable. Since Definitions 1 and 2 are mutually exclusive, we get that a pair of simulators needs to be recoupled when

$$\max\left|\frac{\hat{y}(t)-y(t)}{\max\hat{y}(t)-\min\hat{y}(t)}\right| \geq \epsilon_{\mathrm{p}}, \quad t \in \boldsymbol{t}_{\mathrm{w}}, \tag{8}$$

which is complementary to (7). As opposed to the case of simulator decoupling, delaying simulator recoupling would likely have a negative impact on the accuracy of the co-simulation, so an $R_{\mathrm{w}} = 1$ is recommendable.

In this case we can reduce the computational expense of evaluating (8) at every macro time step if we take into account that when a transition from predictable to unpredictable interface variables occurs, the maximum deviation between the true outputs and those calculated from the trajectory model occurs at the last sample in $\boldsymbol{t}_{\mathrm{w}}$ (provided that $R_{\mathrm{w}} = 1$). Thus we can simplify (8) to

$$\left|\frac{\hat{y}(t_k)-y(t_k)}{\max\hat{y}(t)-\min\hat{y}(t)}\right| \geq \epsilon_{\mathrm{p}}, \quad t \in \boldsymbol{t}_{\mathrm{w}}, \tag{9}$$

which is more computationally efficient. Every time we evaluate (9), $\hat{y}$ and y are evaluated only at t_k. Furthermore, $\hat{y}$ does not need to be recomputed.

The importance of expressing (7) and (9) in terms of subsystem outputs instead of inputs becomes apparent when we consider that in decoupled mode the inputs are obtained from trajectory models that are not updated to reflect possible changes in the operating conditions of other subsystems. On the contrary, a change in the operating conditions of a subsystem does reflect on its own outputs, causing them to deviate from their trajectory model.

A trajectory model can cease to be representative of an interface variable either due to its own limitations (e.g., limited model accuracy) or due to a change in the operating conditions of a subsystem (e.g., change of a model parameter). Changes in the operating conditions are caused by simulation events, which can be classified as external or internal. External events are scheduled, either by the co-simulation user or another entity, and their time of occurrence is known in advance. On the other hand, internal events are a product of the co-simulation itself and their time of occurrence might be unknown (e.g., a stochastic event). External events are the most favorable for simulator recoupling because the simulators know when to recouple before decoupling. Aside from mode transitions caused by external events, all other transitions back to the coupled mode pose an additional challenge for non-real time co-simulation.

In a non-real time environment there are no guarantees on the time it takes to execute a process. This means that as soon as the simulators decouple, they will likely progress at different rates. When a transition to the coupled mode becomes necessary, the simulator that discovers the need for recoupling can either be ahead of all the others in simulation time or behind at least one simulator. In the first case recoupling is simple: the simulator that discovers the need for recoupling informs the others and waits for them to catch up so they can all resume coupled execution from the same point in simulation time. In the second case the simulators that are ahead in simulation time must roll back before recoupling is possible. This is unfavorable not only because rolling back comes with a performance penalty, but also because in practice not all simulators support the roll-back operation. A possible solution to this problem is to slow down the execution of the faster simulators during decoupled execution so all simulators advance at the same rate, but in a non-real time environment this is not trivial and we consider it beyond the scope of this paper.

E. ALGORITHMIC DESCRIPTION

Algorithm 1 presents a pseudocode description of the selectively-decoupled co-simulation from the point of view of a simulator. We assume that all decoupling and recoupling requests are handled by a co-simulation master. This master should accept or reject decoupling requests and forward recoupling requests to the corresponding simulators. Additionally, the master is in charge of receiving the outputs of each simulation and providing them with the inputs they require.

IV. SELECTIVE DECOUPLING FOR AC CIRCUIT CO-SIMULATION

In Section III we introduced the selectively-decoupled co-simulation independently from the system it is applied to by leaving the only application-dependent component unspecified: the trajectory model identification method. Specifying this key component requires knowledge of the physical behavior that can be expected from the system under analysis, in our case, an AC circuit.

A. A TRAJECTORY MODEL FOR STEADY STATE

In AC circuit co-simulation, the interface variables are typically voltage and current, although in some cases power is used as well [9]. We know that these interface variables mainly follow sinusoidal trajectories that may contain harmonic distortion caused by non-linear devices, such as transformers and power electronic converters. If we define predictable interface variables as those that follow these sinusoidal trajectories, we can define the trajectory model as

$$\hat{u}(t) = A_0 + \sum_{n=1}^{N} A_n \sin\left(2\pi f_n t + \phi_n\right), \tag{10}$$

where A_n,f_n and ϕ_n are the amplitude, frequency and phase of the n^{th} harmonic, and N is the total number of harmonics. This trajectory model is valid when the circuit is in steady state. Thus, identifying a suitable trajectory model $\hat{u}(t)$ is equivalent to estimating the parameters A_n, f_n, ϕ_n and N in (10).

B. ACCURACY OF DISCRETE FOURIER METHODS

In the case of continuous trajectories, a Fourier Transform would yield the required trajectory model parameters. However, in a discrete case such as a co-simulation, neither the Discrete Fourier Transform (DFT) nor its more efficient implementation, the Fast Fourier Transform (FFT), are likely to produce accurate results due to their discrete frequency resolution. When estimating the frequency of a harmonic,

Algorithm 1 Selectively-Decoupled Co-Simulation

```
1: Define: subsystem $s$, macro time step index $k$, recoupling macro time step index $k_r$, $t_k \in \{t_0, t_1, \dots t_k, \dots t_K\}$ the discrete time grid of macro time steps, and $\boldsymbol{t}_w = \{t_{k-N_s+1}, t_{k-N_s+2} \dots t_k\}$ a discrete moving time widow.
2:
3: Initialize $\dot{\boldsymbol{x}}_s = \boldsymbol{f}_s(\boldsymbol{x}_s, \boldsymbol{u}_s), \boldsymbol{y}_s = \boldsymbol{g}_s(\boldsymbol{x}_s, \boldsymbol{u}_s)$
4: mode ← COUPLED
5: $k \leftarrow 0$
6:
7: while $k < K$ do
8:
9:     if mode = COUPLED then
10:
11:        if $\boldsymbol{y}_s(t), t \in \boldsymbol{t}_w$ are predictable then
12:            Request decoupling to master
13:        end if
14:
15:        Send $\boldsymbol{y}_s(t_k)$ and receive $\boldsymbol{u}_s(t_k)$
16:
17:        if Decoupling request accepted by master then
18:            Identify trajectory models $\hat{\boldsymbol{u}}_{s,k}(t)$
19:            $\boldsymbol{u}_s(t) \leftarrow \hat{\boldsymbol{u}}_{s,k}(t)$
20:            mode ← DECOUPLED
21:        else
22:            Create interp. polynomials $\tilde{\boldsymbol{u}}_{s,k}(t)$
23:            $\boldsymbol{u}_s(t) \leftarrow \tilde{\boldsymbol{u}}_{s,k}(t)$
24:        end if
25:
26:    else if mode = DECOUPLED then
27:
28:        if Recoupling requested by master then
29:            Receive recoupling index $k_r$
30:            Roll back or catch up to $k = k_r$
31:            mode ← COUPLED
32:        else if $\exists y(t_k) \in \boldsymbol{y}_s(t_k) : y(t_k)$ is unpredictable then
33:            $k_r \leftarrow k - 1$
34:            Request recoupling to master at $k_r$
35:        else if Event in next macro time step then
36:            mode ← COUPLED
37:        end if
38:
39:    end if
40:
41:    Solve $\dot{\boldsymbol{x}}_s = \boldsymbol{f}_s(\boldsymbol{x}_s, \boldsymbol{u}_s(t)), \boldsymbol{y}_s = \boldsymbol{g}_s(\boldsymbol{x}_s, \boldsymbol{u}_s(t))$ until $t = t_{k+1}$
42:
43:    $k \leftarrow k + 1$
44:
45: end while
```

the accuracy of a DFT is restricted to

$$\pm\frac{\Delta f_{\mathrm{DFT}}}{2} = \pm\frac{f_s}{2\,N_s},$$

where Δf_{DFT} is the frequency resolution of the DFT, f_s is the sampling frequency, and N_s is the number of acquired samples.

As a reference, a macro time step of 0.1ms is a common choice for co-simulations of a 50Hz electrical power system AC circuit, which means that the interface variables are sampled at a frequency $f_s = 1/0.1\text{ms} = 10\text{kHz}$. At that sampling frequency, 25 periods need to be acquired to obtain a DFT accuracy within ± 1Hz. Taking into account that a frequency deviation of 0.1Hz is significant for these systems, an accuracy of ± 1Hz is unacceptably low, especially considering how many periods need to be acquired. For applications that require more accuracy, methods that interpolate the DFT (or the FFT) to better approximate a continuous Fourier Transform are available.

C. INTERPOLATED FOURIER METHODS

The Quadratically Interpolated Fast Fourier Transform (QIFFT) [10] is one of the methods that approximates a continuous Fourier Transform. The idea behind the QIFFT is to fit a parabola to the tuple $(|X(b-1)|, |X(b)|, |X(b+1)|)$, where $|X(b)|$ is a peak in the discrete spectrum and b its location in normalized frequency $f/\Delta f_{\mathrm{DFT}}$ (bin number), as Fig. 1 (a) shows. The figure shows how neither the true peak value $|X_t|$ nor its location b_t can be directly obtained from the discrete spectrum, but the vertex of the fitted parabola $(\hat{b}_t, |\hat{X}_t|)$ provides a good approximation. To estimate the phase of each harmonic $\hat{\phi}_t$ we find the intersection between the spectrum phase and $\hat{b}_t$ through interpolation, as in Fig. 1 (b).

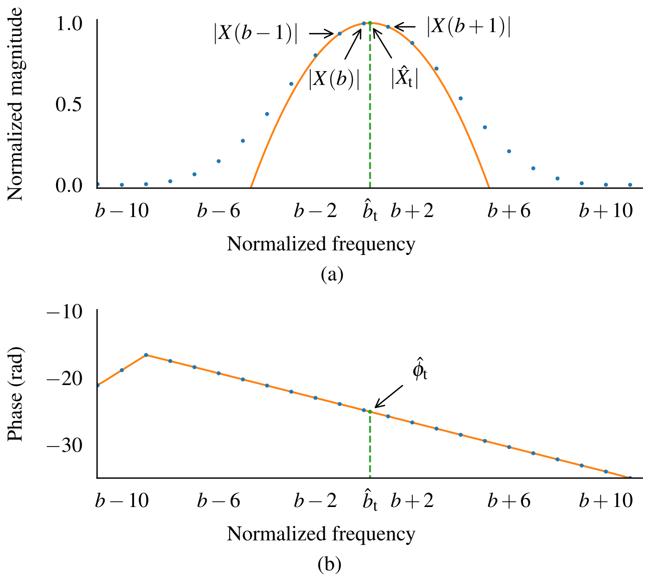

FIGURE 1. **QIFFT of a discrete spectrum. (a) Spectrum magnitude. (b) Spectrum phase.**

The eXponentially weighted QIFFT (XQIFFT) [11] differs from the QIFFT in that it weighs $|X(b-1)|$, $|X(b)|$ and $|X(b+1)|$ using an exponential function before fitting the parabola. This modification improves the accuracy of the estimates $\hat{b}_\mathrm{t}$ and $|\hat{X}_\mathrm{t}|$ with negligible impact on computational performance. By defining

$$\alpha = |X(b-1)| \tag{11}$$
$$\beta = |X(b)| \tag{12}$$
$$\gamma = |X(b+1)|, \tag{13}$$

we can obtain $\hat{b}_\mathrm{t}$ and $|\hat{X}_\mathrm{t}|$ from

$$\hat{b}_\mathrm{t} = b + \frac{1}{2}\frac{f(\alpha)-f(\gamma)}{f(\alpha)-2f(\beta)+f(\gamma)} \tag{14}$$
$$|\hat{X}_\mathrm{t}| = f^{-1}\left(f(\beta) - \frac{1}{8}\frac{\left[f(\alpha)-f(\gamma)\right]^2}{f(\alpha)-2f(\beta)+f(\gamma)}\right), \tag{15}$$

where $f(\Theta) = \Theta^p$ and $f^{-1}(\Phi) = \Phi^{\frac{1}{p}}$ are the exponential weighing function and its inverse. According to [11], $p = 0.2308$ is a good choice for an accurate $\hat{b}_\mathrm{t}$ and $p = 0.2318$ is a good choice for an accurate $|\hat{X}_\mathrm{t}|$. Experimentally we found both of these values to be appropriate for our application.

Finally, if we ignore the negative frequencies in the spectrum, we can apply (14) and (15) to the n^{th} peak in the discrete spectrum and estimate A_n, f_n and ϕ_n as

$$\hat{A}_n = 2|\hat{X}_\mathrm{t}| \tag{16}$$
$$\hat{f}_n = \hat{b}_\mathrm{t}\Delta f_{\mathrm{DFT}} \tag{17}$$
$$\hat{\phi}_n = \hat{\phi}_\mathrm{t} \tag{18}$$

D. SPECTRUM PREPROCESSING

In practice, the trajectories followed by the interface variables need to be preprocessed to maximize the accuracy of the XQIFFT. The two main challenges that need to be addressed are the possibility of an insufficient frequency resolution, which is detrimental to spectrum interpolation, and spectral leakage [12], which modifies the shape of the spectrum.

Fig. 2 (a) shows the spectrum magnitude of a 0.06s window of a current trajectory sampled at a 10kHz rate. The trajectory has one frequency component around 50Hz, that appears as the most prominent peak, and one around 250Hz that is almost indistinguishable. In the case of the the most prominent magnitude peak, the large separation between magnitude samples would make it difficult to fit a parabola to them as precisely as in Fig. 1. This problem can be mitigated by zero-padding the trajectory before obtaining its spectrum. This results in the smoother spectrum magnitude shown in Fig. 2 (b), where the actual location of both frequency components becomes easier to estimate from the main lobes.

However, the resulting spectrum is affected by spectral leakage, as the presence of side lobes around each main lobe indicates. These side lobes are a challenge for peak detection, as they are difficult to distinguish from the main lobes without supervision, and modify the amplitude of the main lobes. We can mitigate spectral leakage by applying a windowing function to the zero-padded trajectory. Fig. 2 (c) shows the result of applying a Blackman window [12] to the zero-padded trajectory. The resulting spectrum has two smooth and easily distinguishable main lobes around 50Hz and 250Hz. For each main lobe it is now straightforward to identify $|X(b-1)|$, $|X(b)|$ and $|X(b+1)|$ and to apply the XQIFFT.

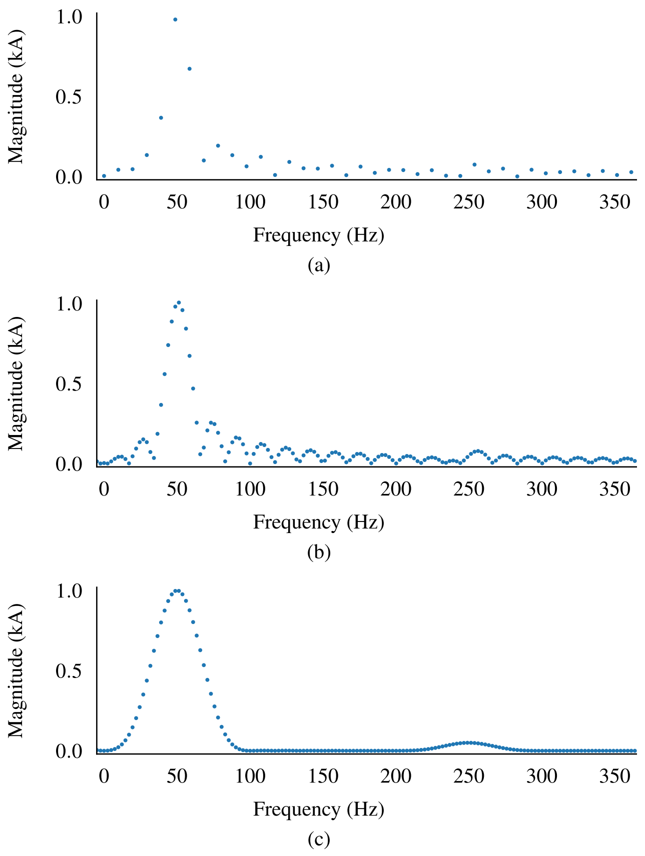

FIGURE 2. Spectrum preprocessing of a current trajectory with one frequency compoent at 50Hz and another at 250Hz, sampled for 0.06s at a 10kHz rate. (a) Original trajectory. (b) Zero-padded trajectory. (c) Zero-padded and Blackman-windowed trajectory.

E. PARAMETER POSTPROCESSING

Experimentally we found that the ϕ_n that the XQIFFT produces are not accurate enough for our application. To remedy this, we resorted to curve fitting based on least squares optimization. For this, we transform the trajectory model into a function of time an phase and solve

$$\{\phi_1,\ldots,\phi_N\} = \underset{\varphi_1,\ldots,\varphi_N}{\operatorname{argmin}} \sum_{t\in\boldsymbol{t}_\mathrm{w}} \left[u(t) - \hat{u}(t,\varphi_1,\ldots,\varphi_N)\right]^2, \tag{19}$$

using the Levenberg-Marquardt algorithm, which is an iterative method. We use the ϕ_n obtained from the XQIFFT as starting point for the first iteration, and let the algorithm refine them further.

V. METHOD EVALUATION

To evaluate the selective decoupling method we measured its accuracy with respect to a monolithic simulation and its speedup with respect to a traditional co-simulation, using a

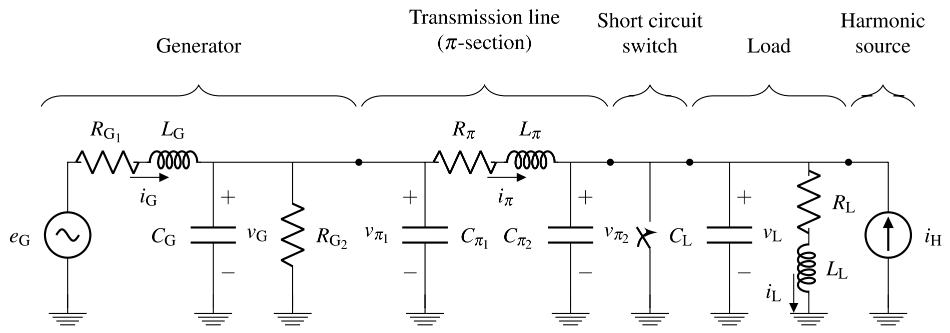

FIGURE 3. Diagram of the test circuit.

test circuit. We quantified accuracy by measuring the deviation (error) of the state variables computed with co-simulation from those obtained from a monolithic simulation. In every case we present the error of a given state variable in percent of the dynamic range of said state variable. We calculated speedup as the ratio between the execution time of a traditional co-simulation and a selectively-decoupled co-simulation. All of these (co-)simulations are of the same AC circuit.

A. TEST CIRCUIT

Fig. 3 shows the test circuit we used to evaluate the selective decoupling method. This circuit represents one phase of a simple electrical power system, composed of a generator, a transmission line and a load, and it is based on the electromagnetic transient models from [13]. The switch connected between the transmission line and the load simulates line-to-ground short circuits, and the current source connected in parallel to the load injects 3rd and 5th harmonics to simulate the presence of non-linear devices. Table 1 specifies the parameters of this test circuit.

TABLE 1. Test circuit parameters.

Symbol	Value	Unit
e_G	$60\sin(100\pi t)$	kV
R_{G_1}	0.1	Ω
R_{G_2}	100	Ω
L_G	0.2	mH
C_G	1	µC
ℓ_π	15	km
r_π	0.01273	Ω/km
l_π	0.9337	mH/km
c_{π_1}, c_{π_2}	6.37	µC/km
R_π	$r_\pi \ell_\pi$	Ω
L_π	$l_\pi \ell_\pi$	mH
C_{π_1}, C_{π_2}	$c_{\pi_1}\ell_\pi$ or $c_{\pi_2}\ell_\pi$	µC
R_L	20	Ω
L_L	4	mH
C_L	20	µC
i_H	$100\sin(300\pi t) + 60\sin(500\pi t)$	A

For co-simulation, we split the test circuit in two subsystems as in Fig. 4, where v_{π_1} and i_π are the interface variables. At every macro time step, subsystem A sends v_{π_1} to subsystem B, and subsystem B enforces v_{π_1} with a controlled

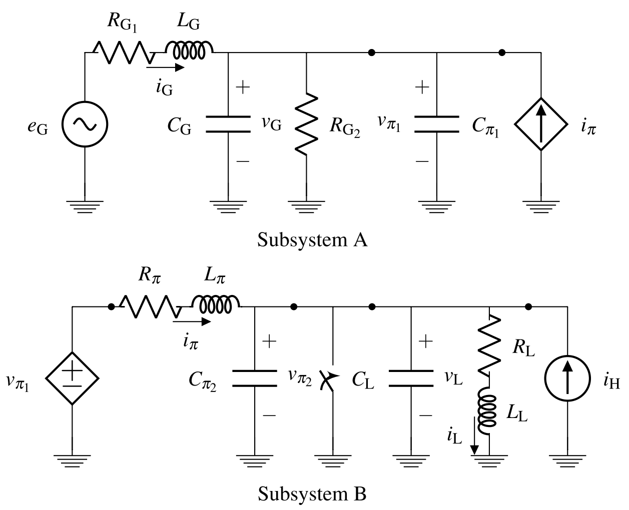

FIGURE 4. Diagram of the co-simulated test circuit split in two subsystems that exchange the interface variables v_{π_1} and i_π.

voltage source. At the same time, subsystem B sends i_π to subsystem A, and subsystem A enforces i_π with a controlled current source.

B. TEST ENVIRONMENT

We implemented the circuit model, the simulators and the co-simulation master in Python 3.6, aided by the numerical methods provided by SciPy [14], and by the ØMQ [15] messaging library for communication between the simulators and the master. We ran all the (co-)simulations on a desktop computer with a 3.5GHz Intel Xeon CPU and 8GB of RAM. All processes (i.e., both simulators and the co-simulation master) run in parallel, each on a different CPU core. In our implementation, the simulators are in charge of analyzing their own inputs and outputs and of requesting mode transitions to the co-simulation master. In turn, the co-simulation master has the additional task of synchronizing mode transitions at the request of the simulators.

C. CASE 1: VALIDATION

To validate the selective decoupling co-simulation method, we compared it to a monolithic simulation and a traditional co-simulation of the test circuit. The (co-)simulated scenario includes a short circuit event that starts at $t = 0.05$s and clears at $t = 0.15$s, and a load event at $t = 0.25$s represented as a step reduction of R_L to 5Ω. For the selectively decoupled co-simulation we considered two cases: one where these events are known in advance and another where they are unknown and must be detected. This is to study how the method reacts to external and internal events. To solve the differential equations that model our test circuit we used the DOPRI5 solver, which is a Runge-Kutta solver of order 4(5) with step size control [16], and limited its maximum step size to the size of the macro time step. For the co-simulations we used a macro time step $H = 0.1$ms, an acquisition window size $T_w = 2/50\text{Hz} = 0.04$s, a window hop size $R_w = 1$ sample, and a predictability threshold $\epsilon_p = 0.02$p.u.

Table 2 presents the execution time of each method. In the table we can see that the co-simulation is more than four times slower than the monolithic simulation. The table also shows that the selectively-decoupled co-simulation provides a speedup of about 20% with respect to the traditional co-simulation. Even though this is a substantial improvement, it is not enough to come close to the execution time of the monolithic simulation. Unexpectedly, the selectively-decoupled co-simulation with unknown event times provides a higher speedup than the one with known event times, despite the additional operations the former executes. This is because an event can only be detected after it happens, which causes the co-simulation with unknown event times to remain decoupled for slightly longer that its counterpart. Nevertheless, we do not believe this result would necessarily extend to larger models, where the penalty for rolling back a simulator is higher and could offset the gains from longer decoupled execution.

TABLE 2. Execution times and speedup for Case 1 (average of 10 runs).

Method	Execution time (s)	Speedup (p.u)
Monolithic	3.10	–
Co-simulation	13.67	–
Selective decoupling (known event times)	11.58	1.18
Selective decoupling (unknown event times)	11.42	1.20

Fig. 5 compares the trajectories of all the state variables in the test circuit, computed with each (co-)simulation method. The colored background indicates that the co-simulation is decoupled. The figure shows that all the trajectories overlap to the point where they are practically indistinguishable from each other, even when the co-simulation is decoupled. The selectively decoupled co-simulations are able to seamlessly transition between modes, and of accurately reproducing fast transients, such as the peaks in v_L and i_L at $t = 0.15$s, or the small oscillations in v_L at $t = 0.25$s.

It is not until we examine the error of each co-simulation with respect to the monolithic simulation in Fig. 6, that the differences between the methods become clear. With the

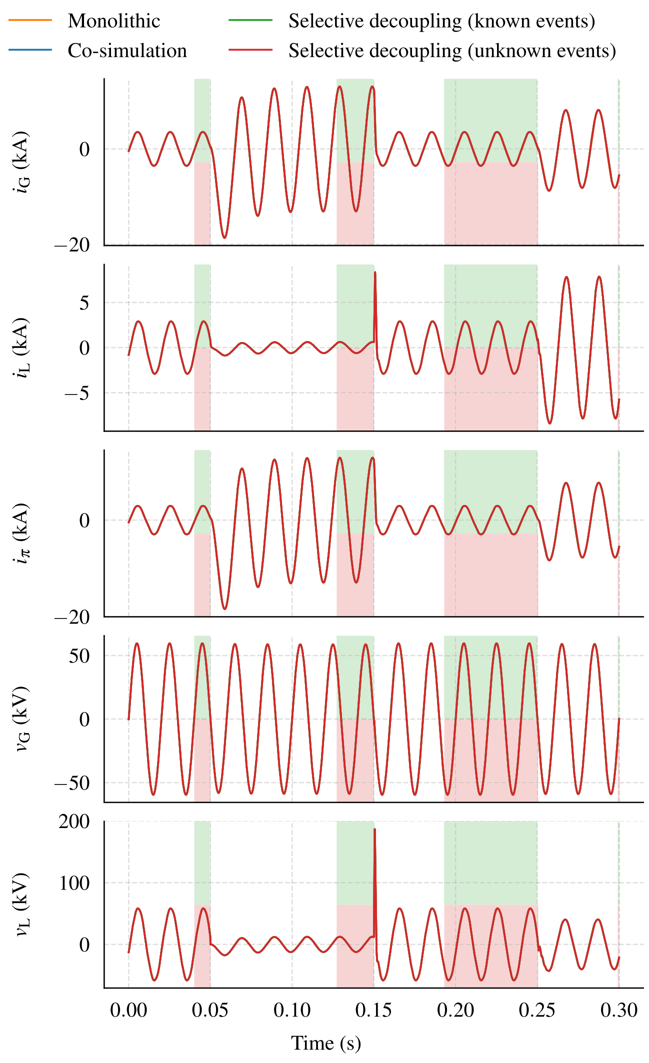

FIGURE 5. State variables computed with a monolithic simulation, a traditional co-simulation, a selectively-decoupled co-simulation with known event times, and a selectively-decoupled co-simulation with unknown event times (Case 1). The colored background indicates when the co-simulation is in decoupled mode: green for known event times, red for unknown event times. Note that $v_L = v_{\pi_2}$ and $v_G = v_{\pi_1}$.

exception of a few peaks that occur at mode transitions, the errors obtained from the selectively decoupled co-simulations are well below 1%. The error plots show that all three co-simulations are similarly accurate in coupled mode, and that the error increases as soon as the simulators decouple. During the longest decoupled mode we can also see that the error has a tendency to increase, which we attribute to the limited accuracy of the trajectory models. Although both selectively-decoupled co-simulations show similar accuracy, after recoupling the error is slightly higher for the co-simulation with unknown event times. The delay between event occurrence and event detection is to blame for this additional deviation.

D. CASE 2: INFLUENCE OF THE MACRO TIME STEP

The objective of this case is to study the influence of the macro time step H on the accuracy and execution time

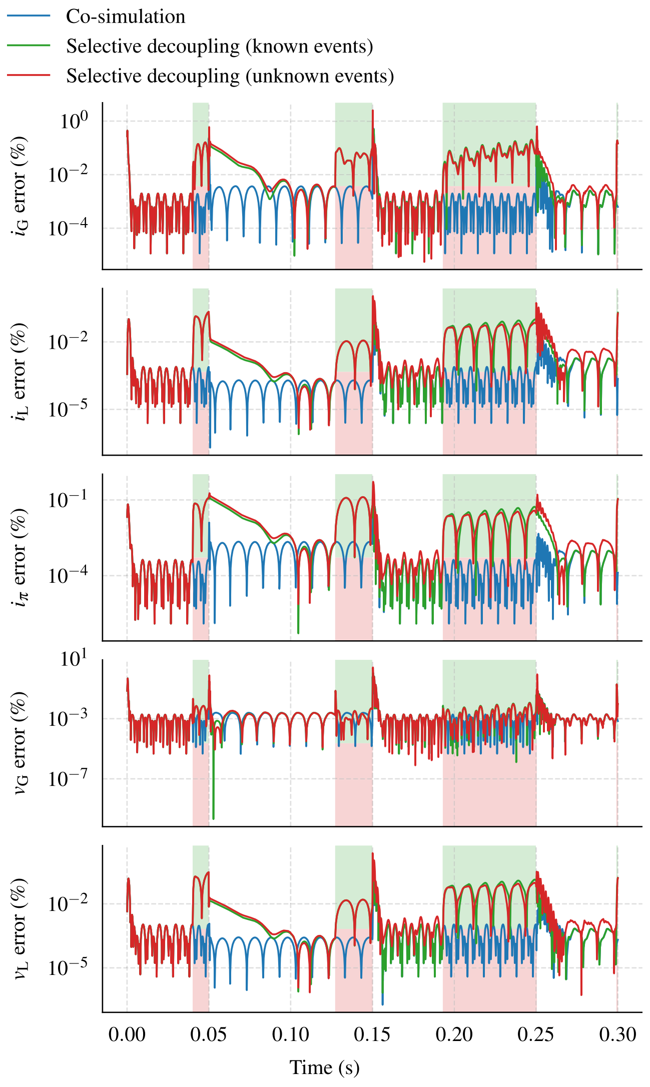

FIGURE 6. State variable errors of the traditional co-simulation, the selectively-decoupled co-simulation with known event times, and the selectively-decoupled co-simulation with unknown event times (Case 1). The errors are measured with respect to the monolithic simulation and are in percent of the dynamic range of the corresponding state variable. The colored background indicates when the co-simulation is in decoupled mode: green for known event times, red for unknown event times.

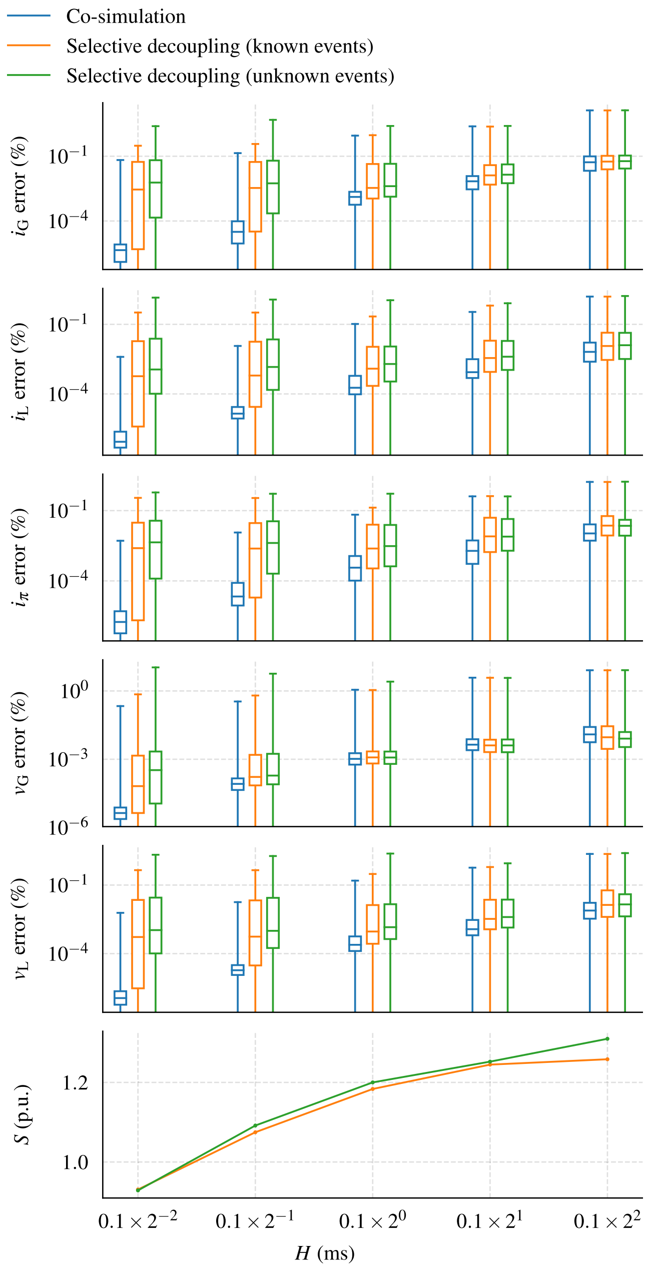

FIGURE 7. State variable errors with respect to the monolithic simulation and speedup with respect to the traditional co-simulation for different macro time step sizes (Case 2). The error is in percent of the dynamic range of the corresponding state variable. Each box plot marks the 25th, 50th, 75th and 100th error percentiles. The speedup is the average of 10 runs.

of a selectively decoupled co-simulation. For this purpose, we considered the same scenario and settings as in Case 1, but repeated the co-simulations for different values of H. Fig. 7 shows the results of these (co-)simulations in terms of state variable errors and speedup with respect to H. The figure summarizes the errors using box plots that mark the 25th, 50th, 75th and 100th error percentiles.

The results show an overall tendency for both the error and the speedup to grow as H grows. We can observe that most of the error of the traditional co-simulation lays in a lower range than the error of the selectively-decoupled co-simulations, with some exceptions for large H, where the ranges are approximately the same. We can observe this tendency most clearly if we compare the 75th error percentiles.

Unexpectedly, there are cases where a smaller H produces a higher error range. One example of this is the error in v_G, which lays in a lower range for $H = 0.1 \times 2^{-1}$ms than for $H = 0.1 \times 2^{-2}$ms. In all of these cases, the 25th, 50th and 75th do not follow this tendency, indicating that only a few error points cause the higher error range. By examining the results of each co-simulation individually, we found that the error points that cause the higher error range come from small oscillations that occur at the mode transition right after $t = 0.15$s, the amplitude of which does not show a clear tendency with respect to H.

If we now compare both selectively decoupled co-simulations, we observe that the 75th error percentile is similar for every value of H, but that the 25th percentile drops much lower for the co-simulation with known event times as H decreases. This is because the upper error bound is mostly influenced by the accuracy of the trajectory model, whereas the lower error bound is mostly influenced by the accuracy of the coupled co-simulation (see Fig. 6). The accuracy of the trajectory model does not significantly improve as H decreases, because the accuracy of the DFT depends on the size of the acquisition window (number of acquired periods), not the sample rate (see Section IV-B), provided that the minimum sample rate requirement is met. Since the co-simulation with unknown event times spends more time in decoupled mode for the reasons exposed in Case 1, a larger portion of its error lays towards the higher extreme of the error range.

Regarding the speedup, we see that a selectively decoupled co-simulation can become slower than a traditional co-simulation if H is sufficiently small. As H decreases, (7) must be evaluated more often. Additionally, the trajectory model identification method has to process a larger number of samples. As a result, the overhead of detecting predictable interface variables grows to the point where the selectively decoupled co-simulation yields no benefit.

E. CASE 3: INFLUENCE OF THE PREDICTABILITY THRESHOLD

The objective of this case is to study the influence of the predictability threshold ϵ_p on the accuracy and execution time of a selectively decoupled co-simulation. For this purpose, we considered the same scenario and settings as in Case 1, but repeated the (co-)simulations for different values of ϵ_p. Fig. 8 shows the results of these (co-)simulations in the same style as in Case 2. Although the traditional co-simulation does not depend on ϵ_p, we show its error for each ϵ_p for ease of visual comparison.

The results in Fig. 8 share some characteristics with those from Fig. 7. We observe that the 75th error percentile of the selectively-decoupled co-simulations grows with ϵ_p. We also observe that there is no clear tendency for the 100th error percentile, although higher error ranges do tend to appear for higher ϵ_p. In addition, we see that in most cases both the 25th and 75th error percentiles are lower for the selectively decoupled co-simulation with known event times.

Even though the 75th error percentile increases with ϵ_p, it always remains below 0.5%. However, the 100th error percentile reaches values above 10% for high ϵ_p. By examining the results of each co-simulation individually, we found that the error points that cause such a high 100th percentile come, once more, from small oscillations that occur at the mode transition right after $t = 0.15$s. This indicates that variations of ϵ_p do not affect all mode transitions the same way, and that while some remain seamless, others do not.

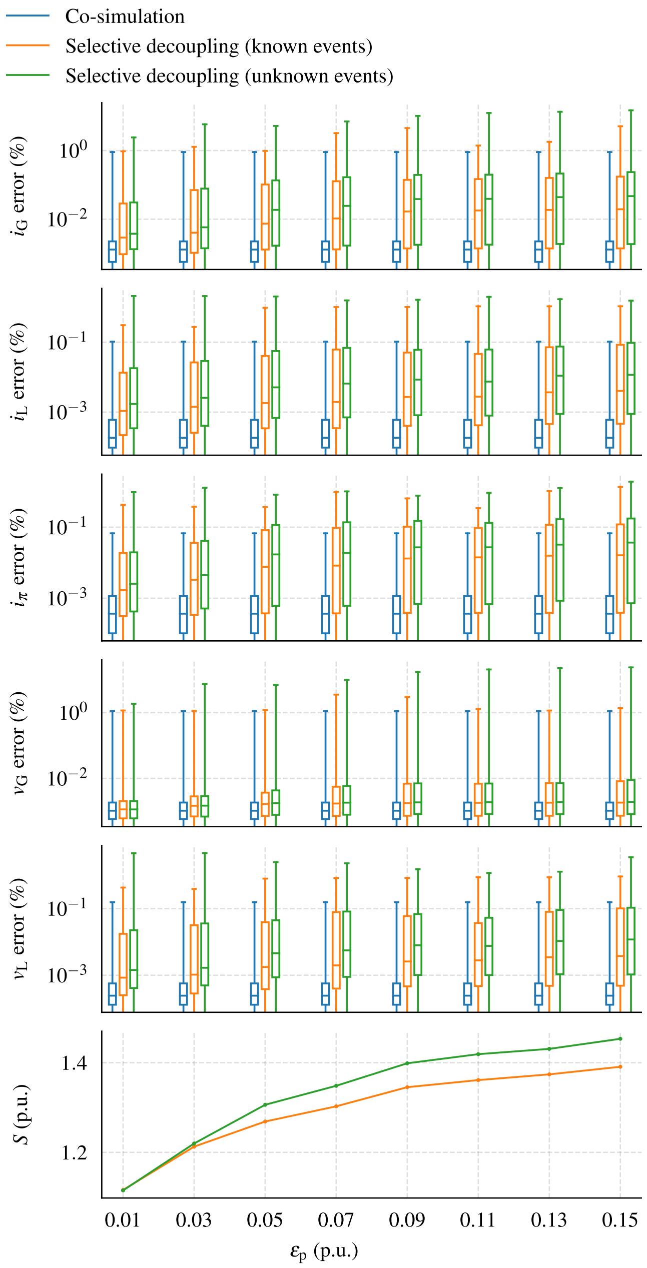

FIGURE 8. State variable errors with respect to the monolithic simulation and speedup with respect to the traditional co-simulation for different predictability thresholds (Case 3). The error is in percent of the dynamic range of the corresponding state variable. Each box plot marks the 25th, 50th, 75th and 100th error percentiles. The speedup is the average of 10 runs.

Fig. 9 shows how changes in ϵ_p shift the time when mode transitions occur. According to the figure, as ϵ_p increases, the transitions to the decoupled mode happen earlier, whereas the transitions to the coupled mode happen later. The figure also confirms that not all mode transitions are equally affected by changes in ϵ_p. For example, the 2nd decoupling happens around 300 macro time steps earlier for $\epsilon_p = 0.15$ than for $\epsilon_p = 0.01$, whereas all the other transitions are shifted by 20 macro time steps or fewer. This means that this transition alone produces most of the additional speedup.

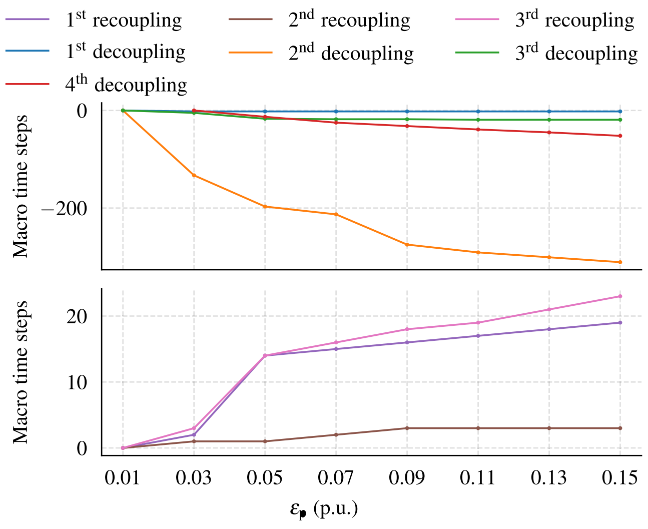

FIGURE 9. Change in the mode transition times measured in macro time steps for different predictability thresholds (Case 3). As the threshold grows, the simulators decouple earlier and recouple later.

How much a mode transition shifts in time as a consequence of a change in ϵ_p has to do with how quickly an interface variable deviates from (or converges towards) its trajectory model. Fig. 10 shows the deviation of i_π from its trajectory model $\hat{i}_\pi$. Here we see that the transitions to decoupled mode with the largest shift are those where the deviation decreases slowly (second and fourth), whereas the least affected transitions are those where there is virtually no deviation (first) or the deviation falls sharply (third).

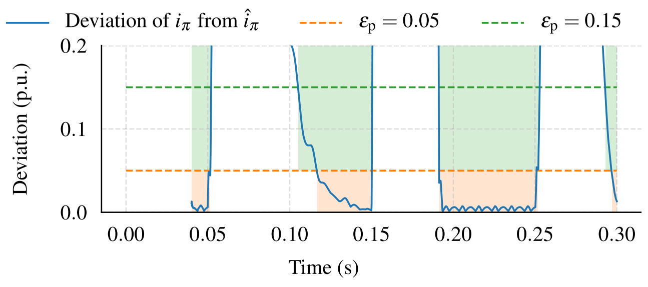

FIGURE 10. Deviation between the true trajectory of i_π and the trajectory model $\hat{i}_\pi$, and mode transitions for two predictability thresholds (Case 3). The colored background indicates when the co-simulation is in decoupled mode: orange for $\epsilon_p = 0.05$, green for $\epsilon_p = 0.15$.

F. CASE 4: INFLUENCE OF THE WINDOW HOP SIZE

The objective of this case is to study the influence of the acquisition window hop size R_w on the accuracy and execution time of a selectively decoupled co-simulation. Once more, we considered the same scenario and settings as in Case 1, but repeated the (co-)simulations for different values of R_w. Fig. 11 shows the results of these (co-)simulations in the same style as in Cases 2 and 3. Since a change in R_w only affects the transitions to decoupled mode, we omit the results of the selectively decoupled co-simulation with

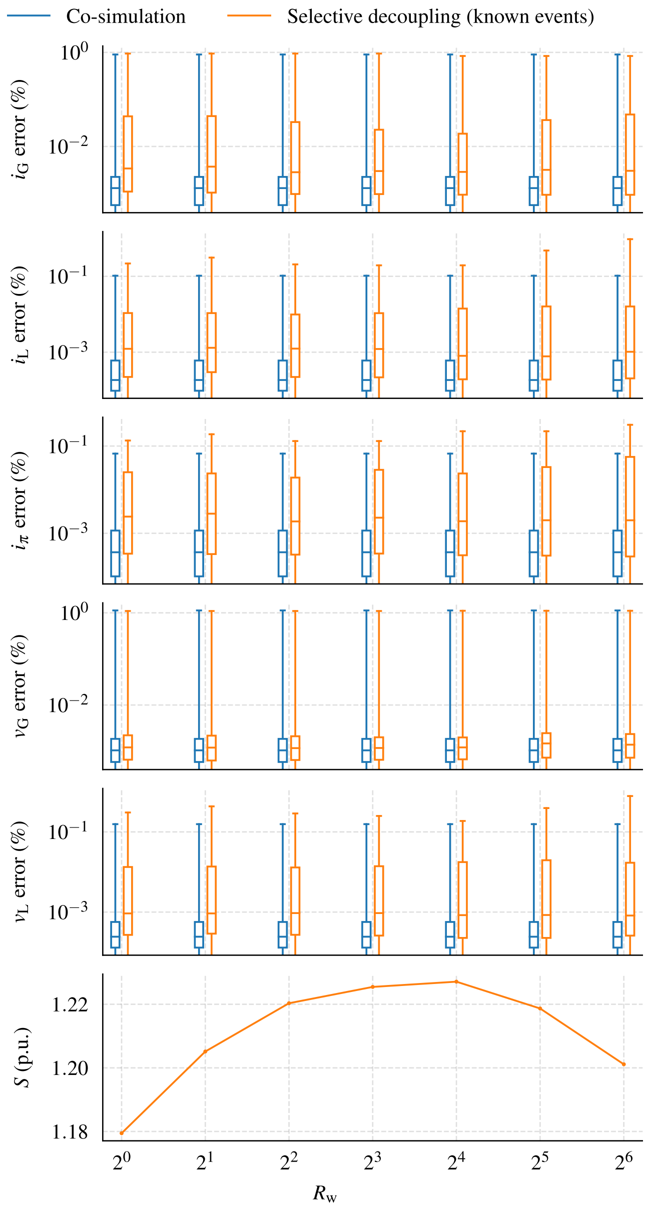

FIGURE 11. State variable errors with respect to the monolithic simulation and speedup with respect to the traditional co-simulation for different acquisition window hop sizes (Case 4). The error is in percent of the dynamic range of the corresponding state variable. Each box plot marks the 25th, 50th, 75th and 100th error percentiles. The speedup is the average of 10 runs.

unknown events. Although the traditional co-simulation does not depend on R_w, we show its error for each R_w for ease of visual comparison.

The results in Fig. 11 show that the error does not change significantly for different values of R_w. Additionally, the maximum speedup that we can achieve by increasing this parameter is more modest than in Case 3. As opposed to previous cases, where the speedup shows a tendency to settle at a certain value, in this case we find that the speedup drops significantly for large R_w. This happens because as R_w increases, so does the probability of delaying transitions to the decoupled mode.

G. CASE 5: SELECTING PARAMETERS FOR ADDITIONAL SPEEDUP

The objective of this case is to tune the selectively decoupled method to obtain a higher speedup than that of Case 1, guided by the results of Cases 2 to 4. Here, we considered the same scenario and settings as in Case 1 but set $\epsilon_p = 0.07$ and $R_w = 2^4$ based on the relationship between error and speedup found in Cases 3 and 4.

TABLE 3. Speedup for Case 5 (average of 10 runs).

Method	Speedup (p.u)
Selective decoupling (known event times)	1.30
Selective decoupling (unknown event times)	1.39

Table 3 shows the speedups for Case 5, which are around 10% and 20% higher than in Case 1. If we now observe Fig. 12, we can see that the first and third transitions to the coupled mode occur much later for the co-simulation with unknown event times than for the one with known event times, which explains the speedup difference between them. In addition, by comparing Fig. 12 to Fig. 5 we see that much of the additional speedup comes from the second and fourth transitions to decoupled mode, which is in accordance with the results from Fig. 9.

Regarding the accuracy of the results, it is still difficult to perceive the differences between the methods, as Fig. 12 shows. Once more, these differences become clear when we observe the error in Fig. 13. Indeed, the error is higher in this case than in Case 1, and the differences between both selectively decoupled co-simulations are also more prominent. Nevertheless, the error remains under or around 1% for all state variables, with the exception of some peaks that reach almost 10% at the second transition to the coupled mode. These errors might be acceptable if we consider the appearance of the trajectories in Fig. 12.

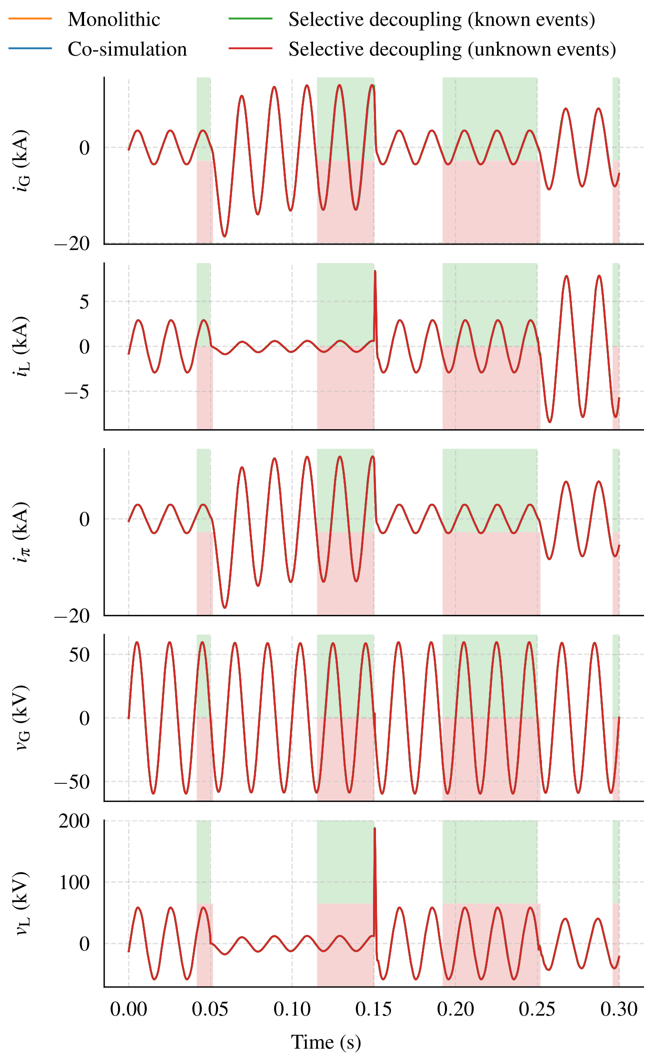

FIGURE 12. State variables computed with a monolithic simulation, a traditional co-simulation, a selectively-decoupled co-simulation with known event times, and a selectively-decoupled co-simulation with unknown event times (Case 5). The colored background indicates when the co-simulation is in decoupled mode: green for known event times, red for unknown event times.

H. DISCUSSION

Our results show that it is possible to speed up a co-simulation by decoupling the simulators, if the interface variables go through predictable states. If more events happened in the time span of our (co-)simulations, there would be no predictable states and no speedup would be possible. However, since our definition of predictability depends on the trajectory model, a more sophisticated trajectory model that can represent the interface variables during slow transients might be able to produce speedup if events occurred more frequently, on the condition that these trajectory models can be identified at a reasonable computational cost.

We found that there are cases where detecting predictable interface variables becomes so computationally expensive that the selectively-decoupled co-simulation turns out to be slower than the traditional co-simulation. This opens up the question of how effective the method would be for systems that have multiple inputs, and therefore, incur a higher computational expense testing for predictability. A selectively-decoupled co-simulation will likely become slower than a traditional co-simulation if the number of interface variables is large enough, and all of them are continuously tested for predictability. Nevertheless, in the case of only two coupled subsystems it is not necessary to test all variables simultaneously. Instead, a simulator could tests only one of its inputs, and only when that input becomes predictable it would test the predictability of the others. This would substantially decrease the additional computational expense.

This brings us to the question of how effective the selective decoupling method is for co-simulations with more than two subsystems. The answer to this questions depends on whether all simulators are expected to couple and decouple simultaneously, or if some can decouple while the others

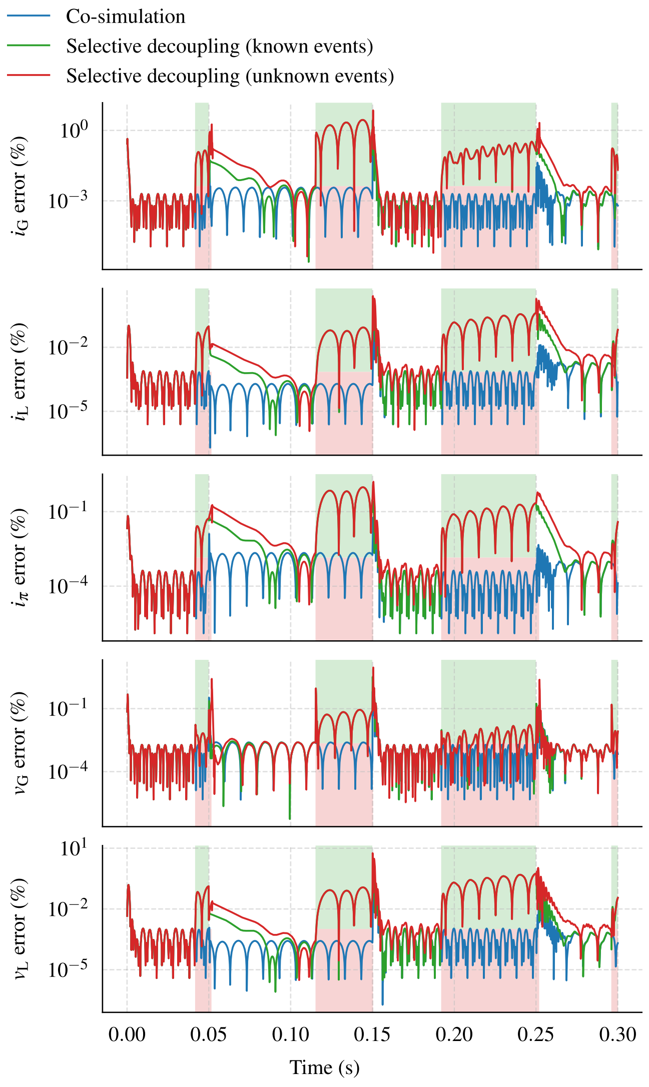

FIGURE 13. State variable errors of the traditional co-simulation, the selectively-decoupled co-simulation with known event times, and the selectively-decoupled co-simulation with unknown event times (Case 5). The errors are measured with respect to the monolithic simulation and are in percent of the dynamic range of the corresponding state variable. The colored background indicates when the co-simulation is in decoupled mode: green for known event times, red for unknown event times.

remain coupled. In the first case, the opportunities for decoupling can become scarce, since not all simulators can be expected to behave predictably at the same time. However, this approach has the advantage that not all interface variables need to be continuously tested for predictability. On the other hand, allowing some simulators to decouple while the rest remain coupled might open up more opportunities for decoupling, but this comes at the cost of having to analyze at least one interface variable per pair of coupled simulators.

As mentioned in Section III, the selective decoupling method requires that the simulators are able to roll back in time. This significant practical limitation can be circumvented by ensuring that all the simulators run at the same rate while in decoupled mode. Yet, this is difficult to ensure in a non-real time environment. One possible compromise could be to have the simulators exchange synchronization messages at low frequency while decoupled. This is a compromise because depending on how infrequent the synchronization messages are, either the speedup or the accuracy when recoupling would suffer.

We defined the selective decoupling method independently from the system it is applied to, but we only analyzed the case of AC circuits. In principle, there is no reason to think that the method cannot be applied to other physical systems. Yet, finding appropriate trajectory models for their interface variables might be more challenging than in our case.

VI. CONCLUSION

This paper proposed a method for speeding up co-simulations by selectively decoupling the simulators when their outputs are predictable, and presented its application to the co-simulation of an AC circuit that represents an electrical power system. After comparing a monolithic simulation, a traditional co-simulation, and two selectively-decoupled co-simulations, our method yielded a speedup of around 20% with errors below 1%, with the exception of some brief peaks. After selecting method parameters based on systematic experimentation we were able to increase the speedup up to 39%, while keeping the error around 1%, again with the exception of some brief peaks. Our results show that it is possible to speed up co-simulations composed of two simulators by decoupling them, while keeping the co-simulation error low. However, questions regarding the scalability of the method to co-simulations with more than two simulators remain open. All in all, the method shows characteristics that can make co-simulation an even more valuable tool for the user.

REFERENCES

[1] P. Palensky, A. A. V. D. Meer, C. D. Lopez, A. Joseph, and K. Pan, "Cosimulation of intelligent power systems: Fundamentals, software architecture, numerics, and coupling," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 34–50, Mar. 2017.

[2] V. Jalili-Marandi, V. Dinavahi, K. Strunz, J. A. Martinez, and A. Ramirez, "Interfacing techniques for transient stability and electromagnetic transient programs IEEE task force on interfacing techniques for simulation tools," *IEEE Trans. Power Del.*, vol. 24, no. 4, pp. 2385–2395, Oct. 2009.

[3] Q. Huang and V. Vittal, "Integrated transmission and distribution system power flow and dynamic simulation using mixed three-sequence/three-phase modeling," *IEEE Trans. Power Syst.*, vol. 32, no. 5, pp. 3704–3714, Sep. 2017.

[4] C. D. López, A. A. van der Meer, M. Cvetkovic, and P. Palensky, "A variable-rate co-simulation environment for the dynamic analysis of multi-area power systems," in *Proc. IEEE Manchester PowerTech*, Jun. 2017, pp. 1–6.

[5] R. Huang, R. Fan, J. Daily, A. Fisher, and J. Fuller, "Open-source framework for power system transmission and distribution dynamics co-simulation," *IET Gener., Transmiss. Distrib.*, vol. 11, no. 12, pp. 3152–3162, Sep. 2017.

[6] K. Mets, J. A. Ojea, and C. Develder, "Combining power and communication network simulation for cost-effective smart grid analysis," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1771–1796, 3rd Quart., 2014.

[7] M. Mirz *et al.*, "A cosimulation architecture for power system, communication, and market in the smart grid," *Complexity*, vol. 2018, pp. 1–12, Feb. 2018.

[8] S. Sadjina, L. T. Kyllingstad, S. Skjong, and E. Pedersen, "Energy conservation and power bonds in co-simulations: Non-iterative adaptive step size control and error estimation," *Eng. Comput.*, vol. 33, no. 3, pp. 607–620, Jul. 2017.

[9] A. A. van der Meer, "Offshore VSC-HVDC networks–impact on transient stability of AC transmission systems," Ph.D. dissertation, Dept. Elect. Sustain. Energy, Delft Univ. Technol., Delft, The Netherlands, Sep. 2017.

[10] J. O. Smith and X. Serra, "PARSHL: A program for the analysis/synthesis of inharmonic sounds based on a sinusoidal representation," in *Proc. Int. Comput. Music Conf.*, Urbana, IL, USA, 1987, p. 87.

[11] K. J. Werner, "The XQIFFT: Increasing the accuracy of quadratic interpolation of spectral peaks via exponential magnitude spectrum weighting," in *Proc. Int. Comput. Music Conf. (ICMC)*, Denton, TX, USA, Sep. 2015, pp. 326–333.

[12] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proc. IEEE*, vol. 66, no. 1, pp. 51–83, Jan. 1978.

[13] R. Thomas, "Fast calculation of electrical transients in power systems after a change of topology," Ph.D. dissertation, Dept. Elect. Sustain. Energy, Delft Univ. Technol., Delft, The Netherlands, Nov. 2017.

[14] E. Jones, T. Oliphant, and P. Peterson. (2001). *SciPy: Open Source Scientific Tools for Python*. [Online]. Available: https://www.scipy.org/

[15] iMatix. (2007). *ØMQ*. [Online]. Available: http://zeromq.org/

[16] J. R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," *J. Comput. Appl. Math.*, vol. 6, no. 1, pp. 19–26, 1980.

CLAUDIO DAVID LÓPEZ (M'16) received the M.Sc. degree in energy technologies from the Karlsruhe Institute of Technology, Germany, and Uppsala University, Sweden, in 2015, and the Engineering degree in electronics from the University of Concepción, Chile, in 2009. He was a Research Assistant with Fraunhofer IEE (formerly IWES) and as a Consulting Engineer on energy-related projects for the public and private sectors. He is currently a Doctoral Researcher with the Intelligent Electrical Power Grids Group, Delft University of Technology, The Netherlands. His current research interests include co-simulation of energy systems.

MILOŠ CVETKOVIĆ received the B.Sc. degree in electrical engineering from the University of Belgrade, Serbia, in 2008, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2011 and 2013, respectively. From 2014 to 2016, he was a Postdoctoral Researcher with the Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently an Assistant Professor with the Electrical Sustainable Energy Department, Delft University of Technology, The Netherlands. His research interest includes the development of co-simulations for energy grids, and modeling for control and optimization of the electricity grids.

PETER PALENSKY (M'03–SM'05) was a Principal Scientist with the Austrian Institute of Technology, an Associate Professor with the Department of Electrical, Electronic and Computer Engineering, University of Pretoria, South Africa, an University Assistant with the Vienna University of Technology, Austria, and a Researcher with the Lawrence Berkeley National Laboratory, CA, USA. He is currently a Full Professor of intelligent electric power grids with Delft University of Technology. His current research interests include energy automation networks, and modeling intelligent energy systems. He is active in international committees such as ISO, IEEE, and CEN.