Modeling and Simulation of Cyber-Physical Electrical Energy Systems with SystemC-AMS

Modeling and Simulation of Cyber-Physical Electrical Energy Systems with SystemC-AMS

Yukai Chen, *Member, IEEE*, Sara Vinco, *Member, IEEE*, Daniele Jahier Pagliari, *Member, IEEE*, Paolo Montuschi, *Fellow, IEEE,* Enrico Macii, *Fellow, IEEE,* Massimo Poncino, *Fellow, IEEE*

Abstract—Modern Cyber-Physical Electrical Energy Systems (CPEES) are characterized by wider adoption of sustainable energy sources and by an increased attention to optimization, with the goal of reducing pollution and wastes. This imposes a need for instruments supporting the design flow, to simulate and validate the behavior of system components and to apply additional optimization and exploration steps. Additionally, each system might be tested with a number of management policies, to evaluate their economic impact. It is thus evident that simulation is a key ingredient in the design flow of CPEES. This paper proposes a framework for CPEES modeling and simulation, that relies on the open-source standard SystemC-AMS. The paper formalizes the information and energy flow in a generic CPEES, by focusing on both AC and DC components, and by including support for mechanical and physical models that represent multiple energy sources and loads. Experimental results, applied to a complex CPEES case study, will prove the effectiveness of the proposed solution, in terms of accuracy, speed up w.r.t. the current state of the art Matlab/Simulink, and support for the design flow.

Index Terms—Design-time Optimization, Power Modeling and Simulation, Cyber-Physical System, Electrical Energy System, Sustainable Energy Planning, Design Space Exploration, SystemC, SystemC-AMS.

1 Introduction

The increased attention to climate change and pollution, together with the adoption of the Paris agreement of 2015, impose a more sustainable design of Cyber-Physical Electrical Energy Systems (CPEES), with the objective of optimizing the processes of generation, distribution, storage, and consumption of energy [1]–[3]. The adoption of sustainable power sources allows indeed to reduce the environmental impact, at the price of a higher instability of the CPEES due to the intermittent nature of the environmental quantities, e.g., solar irradiance and wind [4]–[6]. A careful design is thus crucial to ensure a good balance between power generation and consumption, and to correctly size the components of the CPEES. Computer-aided modeling and simulation tools are the key solutions to assess the CPEES performance under different scenarios (e.g., choice of components number and type, topologies, and management policies), but also to evaluate its economic impact [7], [8].

The traditional approach to the design of CPEES relies on a model-based paradigm, which uses built-in models provided by commercial simulation platforms like Matlab/Simulink. While robust and easy to use, commercial tools lack many important and desirable features: as proprietary tools, they are not easily extensible, and across-version compatibility is not guaranteed. Furthermore, they are not designed to efficiently co-simulate the physical portion (usually continuous-time) and the cyber portion (usually discrete-time) of the system.

To tackle these limitations, recently several approaches have appeared in the literature that aim at applying methods borrowed from the domain of electronic systems design [9]–[12]. One common feature shared by these solutions is that they rely on a database of pre-characterized models of CPEES components, with a pre-defined abstraction level and semantics, and do not allow to seamlessly replace a model of a component with a different implementation [10].

The goal of this work is to go beyond the limitations of the current state of the art, by building *an open-source framework for CPEES modeling and simulation that allows easy extensibility and modularity*. This work borrows the underlying paradigm introduced in [13], which proposes a multi-layer framework based on SystemC-AMS. Such framework, however, has as its main objective that of simultaneously simulating different extra-functional properties (e.g., temperature, reliability), where power is yet another property. Moreover, its focus are smart electronic systems rather than large-scale CPEES; as such, it supports only the DC power domain, and it does not envision any modeling of the environment or of physical evolution. Thus, this work takes inspiration from [13], but steers that paradigm towards the support for larger-scale CPEES, targeting also the AC domain and including the support of the physical domain.

The following are the specific contributions of this paper:

- The construction of a modeling and simulation framework for CPEES that takes into account both the AC and the DC domains, thanks to the introduction of a novel representation in SystemC-AMS of AC voltage and currents, that are non natively supported by the language. The AC domain is supported with three different levels of detail, to have three different accuracy-simulation speed tradeoffs;

- *Y. Chen, S. Vinco, D. Jahier Pagliari, P. Montuschi and M. Poncino are with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy (e-mail: {yukai.chen, sara.vinco, daniele.jahier, paolo.montuschi, massimo.poncino}@polito.it).*
- *E. Macii is with the Interuniversity Department of Regional and Urban Studies and Planning, Politecnico di Torino, Turin, Italy (e-mail: enrico.macii@polito.it).*

Manuscript received xxxxx; revised xxxx.

- The support of the physical domain, by adapting the semantics of the Models of Computations (MoCs) of SystemC-AMS to the characteristics of mechanical components, as an example of physical domain;
- An explorative analysis of an instance of CPEES that shows the effectiveness of the proposed framework in terms of accuracy, flexibility, modularity and simulation speed.

The paper is organized as follows. Section 2 reviews the current state of the art on CPEES simulation, and it provides the necessary background on SystemC-AMS. Section 3 presents the proposed framework, detailing the models, their interfaces and their implementation in SystemC-AMS. The experimental analysis occupies Sections 4-7. Section 4 focuses on the AC domain. Sections 5 to 7 analyze a reference CPEES case study, that undergoes design and exploration of alternative configurations. Finally, Section 8 draws our conclusions.

2 Background

2.1 CPEES design and simulation

Several approaches for modeling and simulation of CPEES have been proposed in literature, addressing different application contexts and CPEES sizes.

Hardware-in-the-loop approaches mix real devices with software-simulated models through sensors and actuators, or through the integration of power electronic devices such as inverters, to test the integration of new technology in a controlled environment [14], [15]. The resulting accuracy is higher w.r.t. software simulation, but application is restricted to small- and mid-scale CPEES.

Proprietary tools, such as Matlab/Simulink, are usually considered the de facto standard [12]. The designer can choose among a number of pre-defined components, or rather implement his own designs, by relying on the provided libraries. However, as closed and proprietary tools, they are not easily extensible and they restrict the possibility of developing custom component libraries and the evaluation of alternative models. Furthermore, the simulation backbone is proprietary and it does not guarantee across-version compatibility.

Equation-based approaches, such as Modelica, decompose the system into elementary components, modeled with basic physics equations or with predefined models [16], [17]. This constrains the types of descriptions that are supported, and it does not allow to effectively model the cyber portion of a CPEES. The same limitation applies to ad-hoc C++ simulators [9], as they tend to focus on specific aspects of the CPEES and to provide restricted libraries of components that can be instantiated and configured.

Co-simulation approaches simulate specific aspects of the CPEES in their native environment, combined with other tools to estimate, e.g., the impact of network latency on control policies, or the application of electricity rates [12], [18], [19]. This leads to a very time-demanding and error-prone process for integrating components with different implementations, e.g., with discrete and continuous time behaviors. Additionally, co-simulation moves the focus from CPEES design to its interaction with other domains, and therefore to a global estimation of the power flow, without allowing to accurately reproduce the behavior of the CPEES components.

The main limitation of the previously presented approaches is that support for CPEES modeling is limited, either in terms of models or in the scale of the supported CPEES. Our work targets a wider support for CPEES in an open source framework, based on SystemC-AMS, thus avoiding the integration of heterogeneous tools and allowing the application of the methodology to a wider range of component models. Some attempts have been made to adopt the standard SystemC framework also in the context of CPEES. [11] uses the Transaction-level Modeling (TLM) and AMS extensions of SystemC for abstracting and modeling physical behaviors. However, the support for the power domain is limited to high-level waveforms or to physical equations. At the same time, [13], [20] proved that SystemC-AMS can support also complex circuit models for the CPEES components. In spite of that, only DC components are supported and the environment or the physical evolution are supported only as input traces.

This work takes inspiration from [13], but with the goal of enlarging the support for EES, targeting also the AC domain and a more accurate modeling of those physical aspects that heavily affect power production and consumption.

2.2 SystemC-AMS

SystemC-AMS extends C/C++ for modeling and simulating analog/mixed-signal systems, including hardware-software systems and also non-functional, continuous time domains [13]. SystemC-AMS is extremely flexible, as it provides different MoCs to cover a variety of domains that can be executed within the same simulation infrastructure, such as pure algorithms (e.g., control policies), equations and linear networks, and also timed behaviors, as exemplified by the high-level schema in Figure 1.

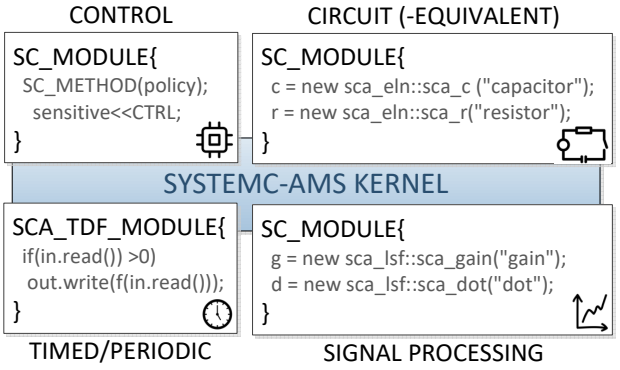


Figure 1. Heterogeneity of the descriptions supported by SystemC-AMS.

Control algorithms and digital discrete-time computation can be modeled as standard SystemC processes, by following the typical event-driven semantics of hardware description languages (HDL, top-left corner), or by adopting the *Timed Data-Flow* (TDF) MoC, which builds a static schedule of modules by considering their producer-consumer communication dependencies and their activation time step (bottom-left corner).

Continuous-time models can be modeled in two ways. Signal processing modules are implemented at *Linear Signal*

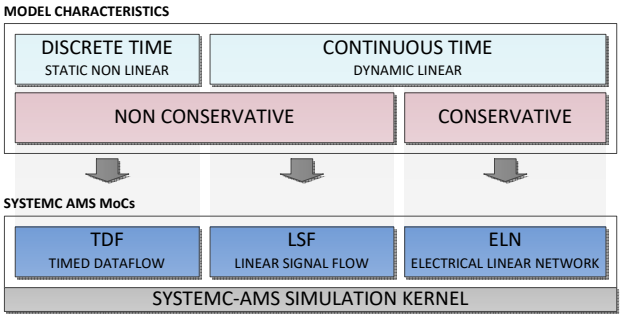


Figure 2. Main characteristics of the SystemC-AMS abstraction levels.

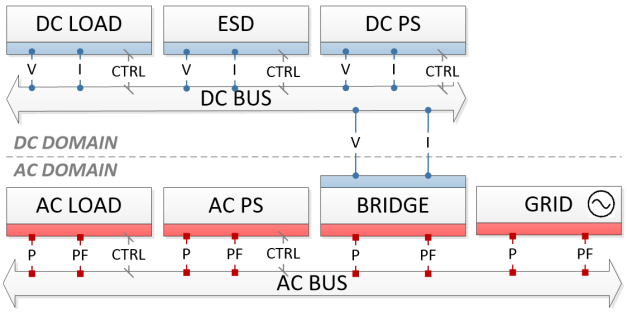


Figure 3. Generic CPEES architectural template with model-specific connections. For the sake of readability, AC interfaces use the (P, PF) interface, that may be replaced by (V, I), depending on the chosen modeling style.

Flow (LSF) through a library of pre-defined primitive modules (e.g., integrators or delay, bottom-right corner). Circuit descriptions can be modeled as *Electrical Linear Network* (ELN) components through the instantiation of predefined linear primitives (e.g., resistors or capacitors, top-right corner). ELN is the only conservative MoC, and it ensures that energy conservation laws are satisfied.

The competitive advantage of SystemC-AMS is that all these different abstraction levels *co-exist in a single simulation environment that relies on a single simulation kernel*; the latter handles both timed events and equations that require a continuous solver, and it ensures correct conversion of the signals between domains, as sketched by Figure 2.

SystemC-AMS is chosen as the reference language for the proposed framework for a number of reasons. The presence of multiple MoCs allows covering a wide range of domains using a single language; moreover, SystemC-AMS natively provides converters between MoCs, thus guaranteeing correctness and hiding synchronization details when different MoCs are simulated concurrently. SystemC-AMS also supports multiple time steps in one single run, so that each component can have its own individual time resolution.

Such flexibility with respect to MoC, time resolution, and abstraction level makes SystemC-AMS a perfect candidate for simulating CPEES, where very heterogeneous components (i.e, cyber or physical) are present, and different degree of accuracy might be desired for different components. Last but not least, as it will be shown in the results section, the native management of synchronization and the use of a single simulation kernel (i.e., no co-simulation) result in excellent simulation performance.

3 Modeling and Simulation Framework

The goal of CPEES simulation is to trace power flow, i.e., power production, distribution, and consumption. The multi-layer framework of [13] made the first effort to formalize the energy flows in CPEES; the key element was the definition of a standardized architecture, with a common bus and a precise taxonomy of components, their interfaces, and semantics. However, [13] focused on small-size electronic systems and did not support neither the modeling the environment or physical quantities (e.g., mechanical components) nor the alternating current (AC) electrical domain. This work, while being focused only on the power flow and not considering other quantities, it generalizes its scope to support all kinds of components included in CPEES, plus the physical domain.

Figure 3 shows the template of a CPEES used in the proposed framework; as in [13], a bus-based architecture is used for scalability, mimicking the structure of a typical computing system. Three main "actors" are envisioned: loads (acting as energy consumers), Power Sources (PS, acting as energy generators), and Energy Storage Device (ESD). Blocks involved in energy exchange are instances of these three types of elements.

The "energy bus" represents a well-defined power (voltage) level, and the connections among blocks occur through the bus. Therefore, every block has its own bus *adapter* that allows to safely connect a block to it (in figure, the shaded blocks between the various elements and the bus). This structure closely mimics the real electrical connections, where the bus is a common connector, and the adapters are power converters. There is however a fundamental difference between this intuitive view of the bus as a connector: the "energy bus" manages the distribution within the system (either as an ideal conductor or with some power loss), and its operations are managed by a *policy* that monitors the components to determine the optimal flow of power. For instance it could monitor the State-of-Charge (SOC) of an ESD to determine whether it can provide energy or it has to be charged.

As the target is realistic CPEES, Figure 3 actually includes two power domains, namely DC and AC, each with its own bus. Components do interface only to one given domain, and the two domains are connected through a block denoted as *Bridge*. This is again consistent with the real electrical connection: when an AC bus and a DC bus co-exist, there is a clear separation between the two domains, and the *Bridge* is a bi-directional AC/DC converter. Finally, the template includes the grid as a special component on the AC domain. The template of Figure 3 exemplifies also the interfaces of the various modules, that will be described in the next section.

3.1 Interfaces

The interfaces of the various types of blocks fundamentally differ based on whether they are DC or AC components. Figure 3 summarizes the main differences for the block interfaces in the two domains.

3.1.1 *DC modules*

In the DC domain we can represent current and voltage as values that change over time. Power is thus represented as

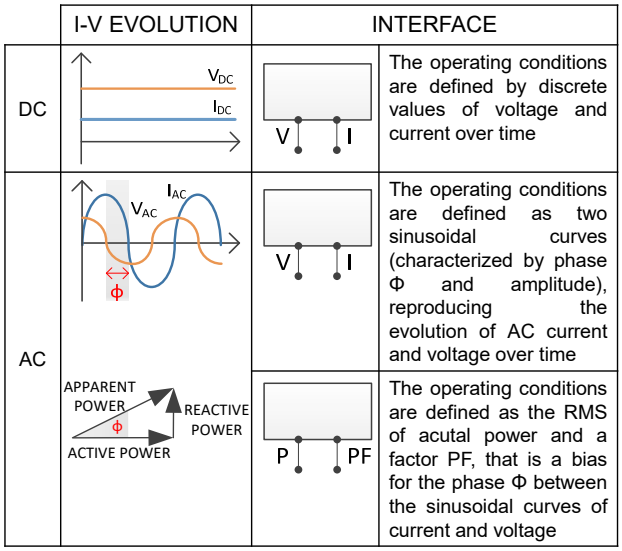


Figure 4. Comparison of AC and DC interfaces.

a pair of values (I, V) that can be represented in SystemC-AMS by using standard signals and ports. Voltage and current are kept as distinct signals, as many DC elements operate at different voltages (e.g., a battery pack output vs. a 5V DC load).

3.1.2 AC modules

In the AC domain, electric current periodically reverses direction according to a sinusoidal behavior. This complicates AC representation in a functional language such as SystemC-AMS, which only supports sinusoidal curves as sources of non-conservative data (e.g., the `sin_src` LSF block). To solve this issue, we propose two different interfaces for the AC domain, depending on the desired level of accuracy with respect to the true sinusoidal behavior:

- The first option is to keep the same interface as for the DC domain (i.e., current and voltage signals), and to reproduce thus the sinusoidal behavior of current and voltage. This will imply extending the standard blocks defined by SystemC-AMS for the modeling of sinusoidal signals, as will be explained in Section 3.1.2. The sinusoidal curves will naturally preserve the major characteristics of AC current and voltage: amplitude and phase. Depending on the type of component, the two sinusoidal curves will be out of phase by a degree ϕ. Both phase ϕ and amplitude of the sinusoidal curves (current in particular) may vary over time.
- The second option is to abstract the sinusoidal behavior, and to preserve only an aggregated value. This is achieved by using the Root Mean Square (RMS) values of the sinusoidal curves: in this way AC power, voltage and current can be represented by a single value that evolves over time, as in the DC domain. RMS current and voltage are however not sufficient to model power in the AC domain: the existence of a phase ϕ between current and voltage implies that some power is not transferred but rather wasted. AC power is thus made of: active power, i.e., power that performs work, and reactive power, i.e., power dissipated due to a phase ϕ between current and voltage (bottom of Figure 4). The (vectorial) sum of the two is called *apparent power*, that is the power that must be taken into account during simulation. Power of AC components in this second representation is thus represented by (i) the RMS of active power over time P, and (ii) the power factor $PF = \cos\phi$. The latter must be represented through an explicit port as the PF is typically not constant, as it may vary for a given AC load (e.g., due to different operations, for instance in a washing machine). For the AC domain it is not necessary to decouple voltage and current as in the AC bus the RMS amplitude of the voltage is normally standardized (e.g., 110, 220, or 380V) and current is therefore implicitly derived from power.

Interface signals will obviously have a direction depending on functionality: loads will receive power, PSs will generate power, whereas ESDs and the grid will be bidirectional as they can serve both functions. Converters will inherit the directions from the blocks they are connecting.
Notice finally that all components (but the grid) have additional control/status ports that are not involved in the power flow; these ports might be driven by energy management policies that decide the power flow based on the overall system state. As an example, one such port for an ESD will be used to control the direction of the power flow (charge/discharge) based on SOC of ESD, whereas for a load it could represent the possibility of disabling it.

3.1.3 Interface modeling in SystemC-AMS

Each component of the system is mapped onto a SystemC *module*, and the system topology formalized in Figure 3 is reflected by the connections between such components. Component ports are mapped onto SystemC TDF ports (`sca_tdf::sca_in` or `sca_tdf::sca_out`) for performance reasons, as TDF generates the least amount of events to be managed. "Data" ports (those carrying power information, i.e., I, V, P, and PF) are of type `double`, while control information (status/commands) are either `bool` or `int`.

3.2 Models

Once the interface has been defined, SystemC modules must be populated with the implementation of the components. Each CPEES component is associated with a model that reproduces its dynamics depending on the environment or on system conditions.
To determine the corresponding mapping onto SystemC-AMS constructs, our framework categorizes models according to two-dimensional space (Table 1): the first dimension concerns the domain (cyber vs. physical), whereas the second one is relative to the underlying simulation semantics. Each simulation semantics will correspond to a different modeling style, and to the mapping onto a specific SystemC-AMS MoC (introduced in Section 2.2). To determine how a model should be implemented, it is thus necessary to understand its classification according to Table 1; e.g., if the model is a circuit (-equivalent) or purely functional, different SystemC-AMS constructs will be used for its implementation. Note that, for a given model of a component, the classification according to Table 1 is unambiguously determined : if the model includes a mix of two modeling

Table 1
Models classification of CPEES components, some examples for each type of semantics for models with the relative references.

	SEMANTICS		
	Functional	**Signal Flow**	**Circuit-equivalent**
DOMAINS	Discrete time models, e.g., functions, polynomials, equations.	Continuous time, including integration, derivative and delay operators.	Electrical circuit models with energy conservation, including, e.g., resistors and capacitors.
Cyber	Management policies implemented by the bus to monitor power flow, enable power sources and/or energy storage devices, and control the loads [13].	–	–
Physical	Abstract the complex physical and mechanical dynamics as equations and functional models [21], [22].	Reproduce the behavior of a physical system as signals and high level primitives, such as state space equations, integrators and delays [23], [24].	Reproduce physical behaviors through ELN primitives that emulate the dynamics of non-electrical phenomena [25], [26].
SystemC-AMS	TDF (Section 3.2.1)	LSF (Section 3.2.2)	ELN (Section 3.2.3)

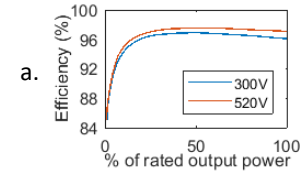
styles, then the model must be divided in two sub-blocks, each mapped to SystemC-AMS separately.

The Section will provide three examples of implementation in SystemC-AMS of models, each corresponding to one column of Table 1. The first three subsections reflect the columns of Table 1. The last subsection is devoted to the modeling of AC, that requires a separate discussion.

3.2.1 Functional Models

Functional models can be reproduced by *implementing the function in C++ or as a digital process*. As an example of functional model we use a DC/AC converter (inverter). This is an interesting case as power conversion requires non-linear devices like diodes that are not currently supported by SystemC-AMS. Although it is not possible to model the detailed circuit implementation in SystemC-AMS, it is probably not necessary to have such a level of detail for a component that fundamentally only *adapts* power levels. Therefore, we could abstract the operations of the inverter in terms of its efficiency, i.e., of ratio between the generated AC power w.r.t. the input DC power, that is usually available from datasheets [27], [28]. This information is sometimes given as a plot of efficiency vs. the ratio of the input DC power and the output rated power (Figure 5.a). We can therefore build a functional model by fitting the curves to a polynomial with two inputs (ratio of input DC power and rated power P and the operating voltage V) and use this equation as a model, as shown in Figure 5.b.

Figure 5.c shows the SystemC-AMS implementation of the functional model of the inverter. Given that the module contains only a functional model, it is implemented as a SystemC-AMS TDF model (line 1, denoted by the keyword `SC_TDF_MODULE`), as this reduces the activation overhead: TDF modules are in fact scheduled statically at fixed time steps, thus avoiding the cost of determining what ports changed value. The module has two ports P and PF on the AC side, and two ports V and I for the DC side. The `initialize()` method (line 11) is invoked at the beginning of the simulation to reset all values and to set the activation timestep of the current module. The `processing()` method (line 6) is invoked at discrete time steps, and it describes the inverter behavior over time: it calculates the efficiency (η) with the empirical equation in Figure 5.b (line 8), and then derives real power (line 9).




```
1. SCA_TDF_MODULE (inverter){
2.   sca_tdf::sca_in<double> I, V, PF;
3.   sca_tdf::sca_out<double> P;
4.   double eta, p, real, …;
5. };
6. inverter:processing(){
7.   p = I.read() * V.read();
8.   eta = 0.88 + 0.74*p – 2.7*exp(p,2) + 4.8*exp(p,3)
           – 4.17*exp(p,4) + 1.39*V.read();
9.   P.write((p * eta)/(1 + PF.read()));
10. }
11. inverter:initialize(){ …}
```

c.

Figure 5. Efficiency curves for an inverter and corresponding functional model implemented in SystemC-AMS.

3.2.2 Signal Flow Models

Signal flow models represent a physical system as signal values that propagate in one direction and that are elaborated by linear elements, e.g. gain, Laplace functions, integrators. Such constructs can be directly mapped onto the corresponding LSF primitives, that are instantiated and connected in a way that reproduces signal propagation.

As an example, Figure 6 shows the high level structure of a wind turbine (a), a subset of its equations (b), the corresponding SystemC-AMS LSF system (c), and a few lines of the corresponding code (d). The difference between the angular speed of the turbine rotor and of the generator rotor is mapped onto a `sca_lsf::sca_sub` primitive (lines 5-8), while the integrator estimating the angle Θ is mapped onto a `sca_lsf::sca_integ` primitive (lines 9-11).

3.2.3 Circuit Equivalent Models

Circuit equivalent models are implemented as a network of SystemC-AMS ELN primitives, instantiated and connected as in their circuit specification.

As an example, we choose a circuit equivalent model of a battery [26]. The circuit on the right of Figure 7 is implemented by mapping its elements to ELN primitives: e.g., current source I_B is instantiated as an ELN

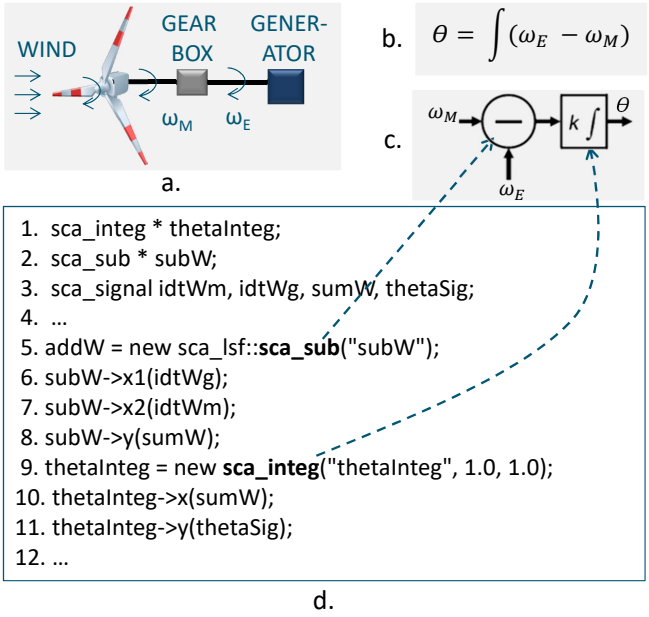


Figure 6. Example of signal flow model implemented in SystemC-AMS.

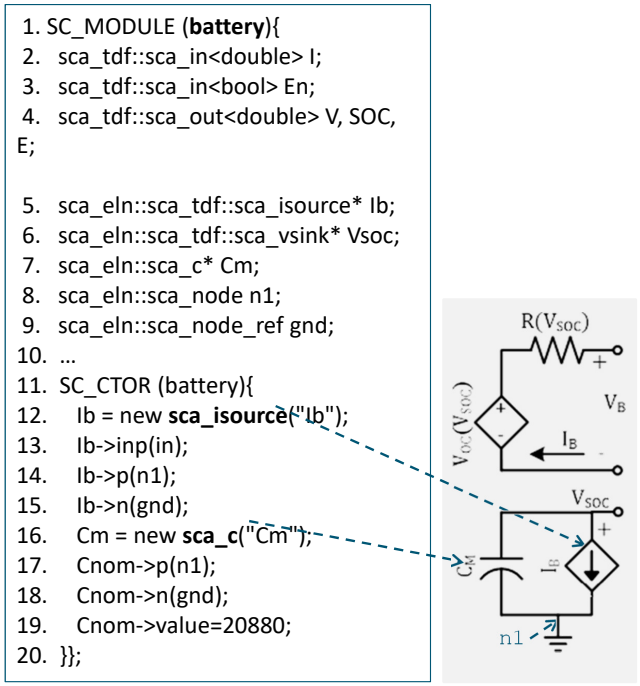


Figure 7. Example of circuit model implemented in SystemC-AMS, by mapping circuit elements to ELN primitives.

`sca_eln::sca_isource` object (lines 11-13), and capacitor C_M as a `sca_eln::sca_c` object (lines 14-15). The connection between ELN primitives is achieved through the instantiation of nodes (e.g., node $n1$).

3.2.4 AC Models

The discussion on SystemC-AMS implementation of AC signals deserves a separate section. SystemC-AMS supports AC only partially: both the primitive to generate a sinusoidal signal (`sca_lsf::sca_source`) and the primitives to generate sinusoidal current and voltage (`sca_eln::sca_vsource` and `sca_eln::sca_isource`) assume phase and amplitude to be fixed throughout the simulation, which is not realistic for AC simulation.

This limitation turns out to be an opportunity to represent the AC domain with different simulation accuracy/speed tradeoffs, by either abstracting the sinusoidal behavior of AC signals, or extending the SystemC-AMS support for these type of signals. We thus propose three levels of details for AC modeling

- *abstract AC modeling*, by representing the AC behavior as discrete values of active power and power factor over time;
- *sinusoidal AC modeling*, by representing AC signals as pure sinusoidal curves with varying amplitude and phase;
- *accurate AC modeling*, as in the previous item, but reflecting more closely the dynamics of AC signals.

3.2.4.1 **Abstract AC modeling**: The first AC modeling option consists of abstracting the AC signals and approximate them as discrete values over time, as in the DC domain. This corresponds to the interface (P, PF), that exports *actual* power and the power factor. This corresponds to a strong simplification of AC signals, as sinusoidal curves are represented by the evolution over time of their RMS and phase ϕ, represented by $PF = \cos\phi$. In this way, the sinusoidal nature of AC current and voltage is totally lost.

Nonetheless, it is important to observe that RMS and phase are the typical quantities measured by meters and made available for appliances [29], [30]. More detailed models (such as the precise electrical components of a load) are almost never available for single appliances, let alone at the level of an entire home or a micro-grid. Thus, this abstraction is reasonable for system-level design and exploration of CPEES.

The AC signals can thus be generated by a TDF module, that writes in output the values of P and PF (e.g., loaded from files) with a chosen sampling frequency, as in the top code fragment of Figure 8.

3.2.4.2 **Sinusoidal AC modeling**: AC current and voltage can be modeled as sinusoidal signals if the values of amplitude and phase over time are known. The voltage curve typically has fixed amplitude in the AC domain (i.e., 110, 220, or 380V, represented by V_{ref}) and the phase is 0 (as the phase is calculated with respect to the voltage curve itself). Conversely, current values may vary over time, and it is thus necessary to have traces of both amplitude and phase over time. In case this information is available as the RMS of actual power and the PF, it is possible to derive this information as follows:

- The amplitude of the sinusoidal curve can be derived from the RMS of power and the amplitude of voltage: $P \cdot \sqrt{2}/V_{ref}$;
- The phase is instead derived as $\arccos(PF)$.

Note that a change of amplitude and phase (and of P or PF) implies a change in the configuration of the sinusoidal curve: this can not however be expressed directly with SystemC-AMS primitives (e.g., `sca_lsf::sca_source`), as these do not allow to vary amplitude and phase of sinusoids over time.

```
1. SCA_TDF_MODULE (abstractAC){
2.   sca_tdf::sca_out<double> P, PF;
3.   ifstream inp, inpf;
4.   double pvalue, pfvalue;
5. };
6. abstractAC :processing(){
7.    inp >> pvalue; inpf >> pfvalue;
8.    P.write(pvalue);
9.    PF.write(pfvalue);
10. } ...
```


```
1. SCA_TDF_MODULE (sinusoidalAC){
2.   sca_tdf::sca_out<double> V, I;
3.   ifstream inp, inpf;
4.   double p, pf;
5.   int t, freq, ampl;
6. };
7. sinusoidalAC:processing(){
8.    inp >> p; inpf >> pf;
9.    V.write(Vconst*sin(2*M_PI*freq*t);
10.   ampl = p*sqrt(2)/Vconst;
10.   I.write(ampl*sin(2*M_PI*freq*t + acos(pf));
11. } ...
```

Figure 8. Simple example of abstract (top) and sinusoidal (bottom) modeling of AC.

For this reason, we extended SystemC-AMS by defining a module that supports the generation of sinusoidal curves with varying amplitude and phase. This is achieved by defining a TDF module that computes the equation for sinusoidal curves:

$$\text{amplitude} \cdot \sin(2 \cdot \pi \cdot \text{frequency} \cdot t + \text{phase})$$

the equation is the same as the one used by `sca_lsf::sca_source`, but amplitude and phase may thus vary over time:

$$V = V_{ref} \cdot \sin(2 \cdot \pi \cdot \text{frequency} \cdot t)$$

$$I = (P \cdot \sqrt{2}/V_{ref}) \cdot \sin(2 \cdot \pi \cdot \text{frequency} \cdot t + \arccos PF)$$

The resulting interface is thus (V, I) as any other DC component; this solution allows one to reproduce the dynamic processing of alternating (sinusoidal) signals in SystemC-AMS. The bottom code fragment of Figure 8 shows an example of this modeling style.

Although this modeling style restores the continuous-time semantics of voltage and current, every sample is computed independently of the others; therefore, when a change of phase occurs, the transition introduces a discontinuity in the sinusoidal curves. This is not the real behavior of AC currents and voltages, for which phase transitions occur smoothly. A solution for this issue is proposed in the next section.

3.2.4.3 **Accurate AC modeling**: This latter modeling style allows solving the issues just mentioned and aims at reproducing the smooth transitions due to phase and/or amplitude changes. The only solution to reproduce such an accurate behavior is to *make AC explicit as a sinusoidal voltage source*, where any change in amplitude and phase of current is generated by the activation of separate circuit branches, that can be connected and disconnected over time by means of switches. An example is shown in Figure 9, where the three branches represent three conceptual electrical loads with different characteristics (e.g. different reactive components) that can be activated at different times by acting on the corresponding switch. Due to the different electrical elements of the branches, each one will have a different impact on the total current absorbed from the voltage source, both in terms of transient behavior and at steady-state. The time constants of such electrical elements ensure a smooth transition whenever changes of amplitude and phase occur. The circuit can be reproduced by using SystemC-AMS ELN primitives (as explained in Section 3.2.3; some mappings are shown in Figure 9). Note that the circuit terminals are left open, as this circuit constitutes a load of a larger CPEES.

It is however important to note that modeling an AC component at this level of detail requires information about its characteristics in terms of reactive behavior (the amount of inductive or capacitive behavior, if any). This information is typically not available unless measurements are carried out on the actual devices [31]–[33], and is thus generally not consistent with the system-level semantics of the proposed approach, that rather focuses on the design phase and on providing a long-term estimation of the behavior of the CPEES. The characteristics of this modeling approach will be further discussed in the experimental section.

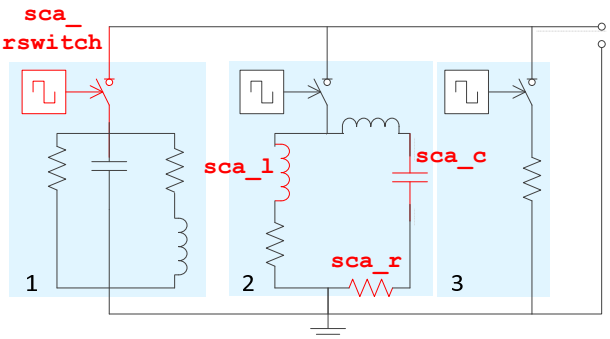


Figure 9. Example of circuit modeling different AC modes: the three branches are activated at different times and generate different amplitude and phase of the AC sinusoidal curves.

3.3 SystemC-AMS Simulation

The support for multiple levels of abstraction of SystemC-AMS allows the simultaneous presence of heterogeneous components: all descriptions (TDF, LSF or ELN) can be used in a single simulation run.

The execution flow of SystemC-AMS simulation is classified into three phases, summarized in Figure 10.

The initial phase is called *elaboration*: the SystemC-AMS kernel builds the module hierarchy and the data structures necessary to handle simulation, including the activation condition of each module, i.e., the activation time step of TDF modules, and the list of input signals for SystemC processes and for ELN and LSF modules. Note that SystemC-AMS provides signal converters between the three abstraction levels, thus guaranteeing a correct and efficient conversion between them.

During the elaboration phase, the SystemC-AMS simulation kernel also builds the equation system generated by

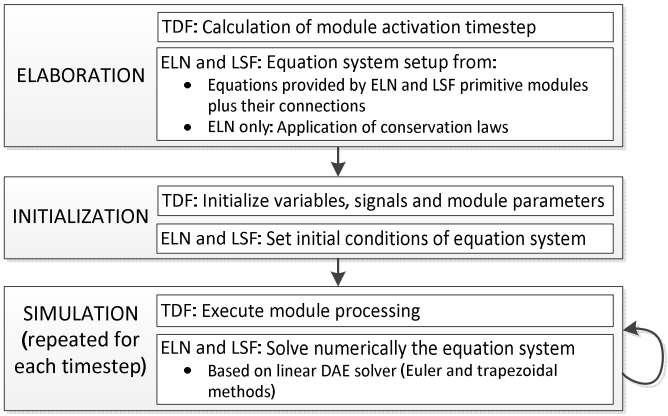


Figure 10. Execution flow of SystemC-AMS simulation as managed by the simulation kernel.

ELN and LSF equations, as derived from the instantiated primitive modules, their connections and the application of conservation laws.

The next phase is *initialization*: the SystemC-AMS kernel sets module parameters, initializes system components (for TDF) and sets the equation initial conditions (ELN and LSF).

The last phase *simulation* is the core of SystemC-AMS simulation: in each simulation cycle, the SystemC-AMS kernel determines the next event to be triggered and the corresponding execution queue, it numerically solves the corresponding equations modeling system behaviour over time by using lightweight numerical methods (i.e., backward Euler and trapezoidal methods with optimization methods like LU decomposition and Woodbury formulas).

It is worth emphasizing that the choice of TDF for ports and synchronization allows efficient simulation: it allows to build a static activation sequence of the modules, that is used to run the entire simulation; this reduces to the minimum any synchronization and management cost, and it hides any conversion overhead from the designer. Additionally, connected elements may run at different time steps: it is enough to set the rate between them, so that SystemC-AMS decouples their activation and automatically buffers signal values.

4 Analysis of AC Modeling

In order to demonstrate the characteristics of the different modeling styles for the AC domain, we propose a simple case study where the AC load is alternatively modeled with one of the three approaches in Section 3.2.4. This will allow to reason in terms of accuracy w.r.t. the AC dynamics, possible approximations and simulation overhead.

To this extent, we simulated the AC circuit in Figure 9 for a six hours span, with the three different levels of detail:

- the individual components of the circuit are accurately simulated at the electrical level in what we call the accurate modeling;
- the other two levels are reproduced by extracting the RMS and phase for each 1s windows, that are then used to build P and PF traces for the abstract model and the sinusoidal waves for the sinusoidal model.

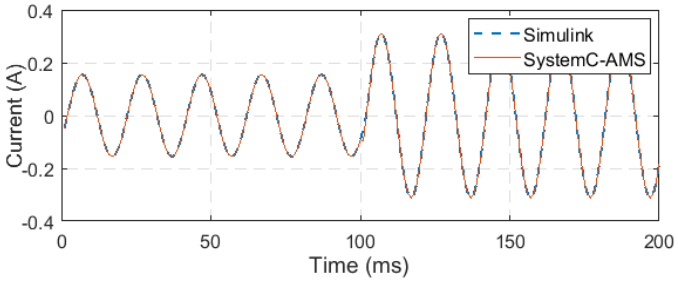


Figure 11. Snapshot of simulation of the accurate AC implementation of the circuit in Figure 9 simulated in Simulink (dashed) and SystemC-AMS (solid) with only an amplitude change.

All such simulations are then compared w.r.t the *accurate* Simulink implementation, to have a measure of accuracy and of relative simulation speed. The outcomes are reported in Table 2. We adopted SystemC-AMS 2.1 and Matlab 2018a. Simulations were run on a server with Intel Xeon 2.40GHz CPU (16 cores, 2 threads each), a 128GB RAM, and Ubuntu 18.04.1.

4.1 Analysis of accuracy

To have an overall measure of accuracy, we used the error on the overall estimated energy (Column *Total Energy*), but also three different indicators for comparing the Simulink and SystemC-AMS traces (Column *Power Trace*) [34]:

- the coefficient of determination R^2;
- the Legates coefficient of efficiency;
- the Willmott's index of agreement.

All these three indicators use higher values to indicate better accuracy (maximum value is 1).

The *accurate AC model* implemented in SystemC-AMS exhibits a high level of accuracy w.r.t. the corresponding Simulink, with only a 0.7% difference in the total estimated energy. This is evident also from the indicators in Table 2, that exhibit very high accuracy compared to Simulink (values are very close to 1).

To have a visual proof of accuracy, Figures 11 and 12 show two snapshots of simulation for the current curves.

Figure 11 shows an amplitude change due to switch configuration at time 100ms: both before and after time 100ms SystemC-AMS (solid) shows a high fidelity w.r.t. Simulink (dashed). The amplitude change (caused by the resistor in branch 3 of Figure 9) generates a small error, due to a more sudden reaction of SystemC-AMS. This difference is due to the different solvers used by the two tools.

At time 1000ms switch configuration is changed again, this time modifying both amplitude and phase of the sinusoidal curve. As evident from Figure 12, the change is propagated gradually on the sinusoidal curve, until stable behavior is achieved around time 1200ms. This is an effect of the inductance and the capacitance present in the activated branch 1 in Figure 9. It is important to note that SystemC-AMS follows well Simulink evolution, despite a small error occurring in the first two sinusoidal periods.

When moving to *sinusoidal AC modeling*, the fidelity of simulation w.r.t. accurate Simulink is lowered. This is evident from Figure 13: the transient behavior (already reported in Figure 12 for the same time interval, solid) cannot be

Table 2
Accuracy and simulation speed of SystemC-AMS w.r.t. Simulink in different AC modeling levels.

Level of AC Modelling	Implementation Methodology	Accuracy					Simulation time (s)	Speedup (×)
		Total Energy		Power Trace				
		Value (Wh)	Diff (%)	R	LCE	WIA		
Accurate	Simulink	1,313.9	0.0	1.0000	1.0000	1.0000	1,513.42	1.00
	SystemC-AMS	1,304.4	0.7	0.9999	0.9942	0.9971	141.32	10.71
Sinusoidal	Simulink	1300.9	1.0	0.9999	0.9940	0.9970	1,023.97	1.48
	SystemC-AMS	1,300.9	1.0	0.9999	0.9940	0.9970	106.22	14.24
Abstract	Simulink	1,300.9	1.0	N/A			2.73	554.37
	SystemC-AMS	1,300.9	1.0	N/A			0.21	7,206.76

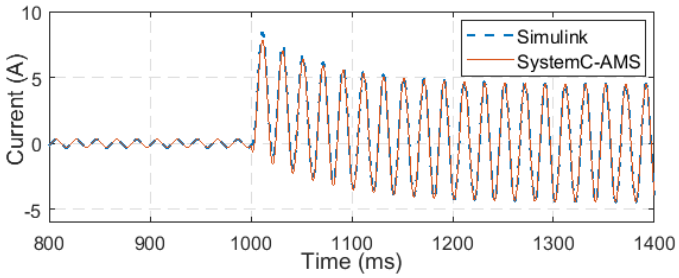


Figure 12. Snapshot of simulation of the accurate AC implementation of the circuit in Figure 9 simulated in Simulink (dashed) and SystemC-AMS (solid) with both amplitude and phase change.

reproduced by a pure sinusoidal curve (sinusoidal AC modeling, dashed), that changes amplitude and phase abruptly. This generates a sudden shift of the first sinusoidal wave generated at time 1000ms (due to the phase change), while amplitude coincides again only around time 1200ms. This implies a larger error between the two modeling styles. However, as shown in Table 2, the relative error on the total energy consumed by the circuit is only 1.0%. It is important to note that the error is not due to SystemC-AMS, but is rather replicated also by Simulink when the same modeling style is adopted: the error is thus not caused by the language, but rather by the adopted modeling style. This is clear from Table 2: since the values of three indicators and of the total energy consumption are exactly the same for the two rows related to Sinusoidal AC modelling (i.e., Simulink and SystemC-AMS).

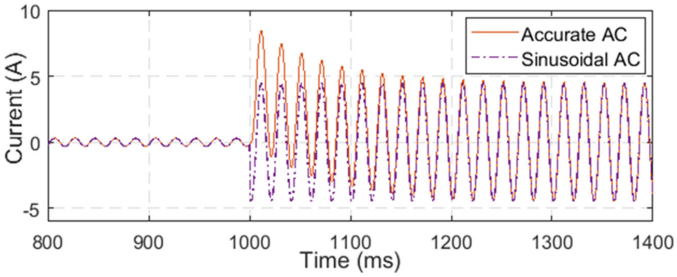


Figure 13. Snapshot of simulation of the accurate AC implementation and of the sinusoidal AC implementation of the circuit in Figure 9 simulated SystemC-AMS.

The *abstract AC modeling* style is a byproduct of the sinusoidal AC curves. Instead of using amplitude and phase to reconstruct the sinusoidal curve, such values are represented as aggregate values: the RMS of the curve and the power factor. However, the accuracy will be, by definition, identical to the case of sinusoidal AC curves (since those curves are simply generated using the P and PF values extracted for each 1s period), and will thus suffer from the same inaccuracies in case of phase changes. This is shown by the fact that the total energy error for the abstract modeling is the same as for the sinusoidal case in Table 2. The values of the three statistical performance indicators cannot be computed for this modeling style, since each 1s window of sinusoidal power trace is replaced by a single P and PF pair: it is thus not possible to compare the sinusoidal behavior with one single sample.

4.2 Analysis of simulation speed

In order to investigate the simulation speed achieved by the different AC modeling levels, we built one system in both Simulink and SystemC-AMS, which simply includes one load, one inverter and one power source. The load part represents the circuit of Figure 9, modeled using one of the three different AC modeling levels in each simulation run. The power source is a battery pack, chosen big enough to ensure 6 hours of simulation. The battery pack and the inverter are modeled with lightweight models (Sections 3.2.3 and 3.2.1), so to highlight the contribution of load modeling on simulation time.

We conduct 6 hours simulation in both Simulink and SystemC-AMS by using the three different AC modeling levels. We used the `tic toc` commands to record the simulation time of Simulink, and the `time` command for estimating the simulation time of SystemC-AMS. Simulation times are calculated as the average over 50 simulations to eliminate the influence of other tasks running on the server. Simulation time is reported in Table 2.

Simulation speed does not improve evidently when moving from the accurate to the sinusoidal modeling style: both of them conduct simulation by using a 1ms time step; in other words, although the sinusoidal AC modelling does not model actual electrical elements in the load, it does not achieve a very large speed-up as it still has to reproduce frequent samples of the AC sinusoidal curves. Furthermore, in both modes the SystemC-AMS implementation is always 10 times faster than the corresponding Simulink implementation.

When using the abstract AC modeling level, the simulation speed dramatically improves (more than three orders of magnitude) since the time step becomes 1s and there is no electrical network in the simulation. This does not imply a higher error in energy estimation, as already discussed in the previous section: the abstract modeling style is thus a winning simulation speed-accuracy trade off when long simulations are required, e.g., to have an estimation of the CPEES evolution over year-long time windows.

5 CPEES Case Study

In order to demonstrate the effectiveness of the proposed framework, we show its application on a case study similar to the CPEES described in [35]: it includes a wind turbine, a photovoltaic (PV) array, a battery pack, a common DC bus, a grid-interface inverter and various AC loads. Figure 14 shows the structure of the case study mapped to the template of Section 3, while Table 3 provide the details about the various components, the corresponding models and the simulation setup.

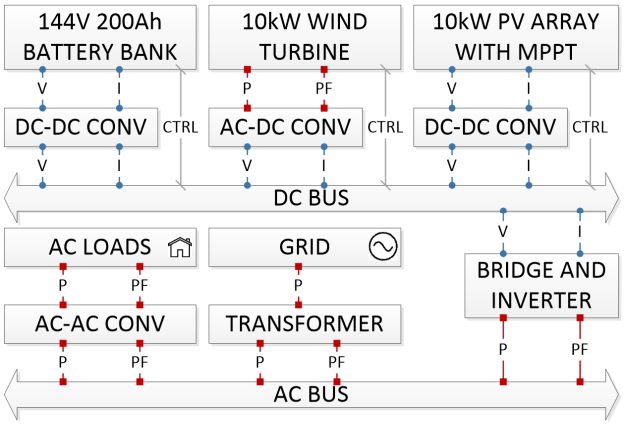


Figure 14. Test case study used in our validation [35].

6 Simulation Results

To evaluate the effectiveness of the proposed framework, we compared a SystemC-AMS implementation of the above case study with the corresponding Simulink, and compared the relative accuracy and performance. We implemented the system in SystemC-AMS 2.1 and Matlab 2018a, respectively; simulations were run with a fixed time step of 1 ms on a server with Intel Xeon 2.40GHz CPU (16 cores, 2 threads each) and 128GB RAM, with Ubuntu 18.04.1.

The Simulink and the SystemC-AMS descriptions have been implemented at the same level of abstraction, i.e., for each component we adopted the same kind of model, and AC loads use the "abstract" AC modeling style (RMS power and PF) rather than the sinusoidal behavior. This ensures that the modeling complexity is comparable, and that no artificial overhead is introduced on either side.

The adoption of abstract AC models is motivated by the analysis reported in Section 4: the computational complexity of the accurate AC models would make unfeasible simulations over a day-long or year-long horizon, like the ones carried in this section and in Section 7. AC modeling is indeed just a part of the big picture, while the general objective is a fast simulation environment for the analysis of an entire CPEES (including energy storage and renewables), which should privilege speed with respect to accuracy. Nevertheless, we have shown that an accurate AC modeling is possible, at the cost of sacrificing simulation speed.

6.1 Simulation Results

Figure 15 depicts system evolution on one example week for environmental traces relative to the NREL Solar Radiation Research Laboratory in Colorado. The figure highlights that the proposed framework allows to trace very different quantities (e.g., the dynamics of an electro-mechanical device such as the wind turbine or the internal evolution of the battery pack) in a single simulation run.

In order to appreciate the details of the simulation, Figure 16 zooms into a shorter 12-hour period. As shown in Figure 16.(c), five representative "regions" can be identified in the 12-hour simulation, with different relations between produced and consumed power. From 2:00am to about 6:00am (I), total power generation is typically larger than the load demand, and, as battery pack SOC is within the 10%–90% operation range, the extra generated power is used to charge the batteries, and thus there is no interaction with the grid. A second interval (II) from about 6 to 7 am shows that total power generation is still larger than the load demand; however, with the battery pack already charged at 90%, the additional generated power is sold to the grid, as shown by the red area in (c). From about 7am to around 10 am (III) the load demand exceeds the generated power, thus discharging the battery pack, as visible in (b). As total power generation is still modest, when the pack reaches 10% SOC it is necessary to purchase power from the grid (IV) for a short time (in the specific example between around 10:10am and 10:25am, the blue area in (c)). Then the load demand decreases and the generation increases so we start re-charging the pack (V).

6.2 Comparison against Simulink

6.2.1 Accuracy Comparison

For the assessment of the accuracy of the SystemC-AMS simulation against Simulink, we did a trace-by-trace comparison of the power consumption of the individual components of the testcase. As the errors are very small, we summarize them in a table rather than showing the resulting waveforms, on which the errors would not be visible. Table 4 reports the average and maximum errors of the power traces of each system components; results clearly show that the proposed SystemC-AMS framework can track very accurately all the components: the largest maximum error ϵ_{max} is for the wind turbine and is less than 0.5%, while the average errors are in all cases negligible.

6.2.2 Performance Comparison

To evaluate simulation performance, we compared Simulink and SystemC-AMS on different lengths of simulated time. We used the same way described in section 4 to calculate the simulation times of Simulink and SystemC-AMS. Table 5 reports simulation times and the speedup of SystemC-AMS over Simulink, for different simulated times (i.e., from half a day to 6 days). From the Table, it is evident that SystemC-AMS is significantly faster than Simulink, with an almost constant speedup around 262 to 265 times, regardless of the simulation length. The difference of simulation time is explained by the difference in terms of solver and implementation of the models: the presence of a mechanical model forces Simulink to use the $ode14x$ solver, as Simulink contains both dynamic and algebraic equations even after Simcape model simplification, thus implying a quite heavy overhead. On the other hand, SystemC-AMS uses a lightweight first order solver based on Euler's method.

Table 3
Details of the CPEES case study.

Component	Description
PV Array	The PV system comprises of a PV array for a total rated power of 10kW and a DC-DC converter. We built the model of one SunPower A300 [36] module (54.7V and 5.49A rated voltage and current), and then scaled it up to the size of the PV array (30 modules). The module model was built using the curves provided in the A300 datasheet with the method of [22], i.e., empirical expressions of I and V as a function of irradiance and temperature. The model is fully functional and it is thus implemented as a TDF module in SystemC-AMS.
Wind Turbine	The wind turbine has a power rating of 10kW. As its power production is affected also by the complex laws controlling the mechanical evolution, we adopted a mechanical model similar to [24], which was split it in two sub-modules: one for the TDF part, implementing the aerodynamic model and most of the drive train and the induction generator; the other for the LSF part, devoted to the constructs requiring a signal flow approach as exemplified in Figure 6. The model takes into account a number of physical and mechanical aspects, including rotor torques, tip speed ratio, a simple blade pitching mechanism and the impact of inertia and of gear ratio in the process of transforming mechanical power into electrical power.
Battery Pack	The battery pack includes 2,400 Li-Ion cells wth 3.4Ah nominal capacity and 3.7V rated voltage (NCR18650B). We first built a circuit-equivalent model [26] of a single battery cell and scaled it up to the size of pack using a 40-series, 60-parallel configuration. The resulting model has been implemented by mapping the linear circuit elements onto ELN primitives, as exemplified in Figure 7.
AC Loads	The AC loads reproduce the power consumption of the appliances of 15 houses to represent a residential community. Traces are taken from the UK Domestic Appliance Level Electricity dataset (UK-Dale) [37], that provides traces of power factor, apparent power and active power with a 1 second time step. Traces have been condensed in a single AC load component implemented in TDF.
Converters	DC-DC converters are modeled in terms of their conversion efficiency as a function of the input voltage, output voltage and current, by using the model in [38]. For the inverters, we considered that their efficiency tends to be constant and almost independent on input power, whenever this is at least 15% of rated power and are thus modeled as a constant efficiency ranging from 92% to 97% depending on the connected component.
Grid	The role of the grid here is only to keep track of the power imbalance between demand and supply. This allows us to abstract from any low-level detail, and to consider the grid as an ideal voltage source with virtually infinite power and energy. The grid is thus implemented as a simple functional TDF block, that receives in input over time the amount of power requested from or returned to the grid.
Buses	Both buses are considered as ideal charge transfer interconnects with a 430V for the DC bus and 380V for the AC bus.
Cyber Policies	The DC bus is provided with an energy management policy similar to the one proposed in [39]. AC loads are satisfied by the PSs whenever possible, and the battery pack is used to compensate whenever necessary (until its state of charge reaches a minimum of 10%). If the sum of energy stored in the battery pack and the power generated from the PSs cannot satisfy the AC loads, the houses purchase energy from the grid. Otherwise, if the demand of the AC loads is less than the total power generation of the PSs, the unused power is used to charge the battery pack until it reaches 90% SOC, and then it is sold back to the grid.
Environment	The traces of irradiance and wind speed have been downloaded from the NREL datasets [40]. As the time resolution of irradiance and wind speed traces is different from those of the load. We adopted the methodology proposed in [13] to solve the issue of different time resolutions.

Table 4
Accuracy of proposed framework w.r.t Simulink.

Component	ϵ_{max} (%)	ϵ_{avg} (%)
PV Power	0.29	1.05e-05
WT Power	0.46	1.08e-04
I_{batt}	0.1362	0.0110
V_{batt}	0.0875	0.0102
SOC_{batt}	0.0963	0.0107

Table 5
Simulation time and Speedup vs. Simulink (1s time step).

Simulation (Days)	Simulink (s)	SystemC-AMS (s)	Speedup (X)
0.5	116.81	0.445	262.49
1	233.21	0.882	264.41
2	456.92	1.726	264.73
3	682.02	2.586	263.73
4	921.19	3.492	263.80
5	1,165.66	4.387	265.71
6	1,388.35	5.243	264.80

However, the adoption of different solvers alone is not enough to explain such a high simulation speedup. To further investigate the simulation complexity of Simulink and SystemC-AMS, we thus focused on the organization of the simulation and the content of each of the two frameworks. The SystemC-AMS simulation framework is organized in 36 SystemC-AMS modules. All three Model of Computations (MoCs) in SystemC are adopted in our simulation: 4 ELN primitives for building the electrical equivalent model of the battery, 17 LSF primitives for implementing wind turbine mechanical model and 15 TDF modules for all other EES components. The effective TDF synchronization adopted for inter-module communication ensures faster simulation evolution and information propagation in the EES. On the other hand, the complexity of the implemented Simulink framework is much higher: it includes 628 blocks from Simulink and Simscape libraries. Such a large number of blocks illustrates that the accurate simulation involved heterogeneous components and requires a lot of computation. According to the profile report generated by Simulink, the heaviest computational blocks are from the wind turbine mechanical model, that account (on average) for 14.7% of simulation time. Other blocks with a heavy effect on simulation speed are 6 ActionPort blocks, 30 Simscape electronics-related blocks, 8 lookup tables, and 5 integration blocks. Additionally, 15 PS-Simulink and Simulink-PS converters influence the overall simulation speed. Last but not least, the whole EES is organized in 64 sub-systems, which add an overheard at initialization time. These numbers show that the simulation speedup is achieved not only thanks to the native characteristics of the language, but also through a well-designed structure of the SystemC-AMS framework. This is confirmed by comparing Table 5 with Table 2. In fact, the latter refers to a simpler circuit where the impact of framework organization is less relevant. In that case,

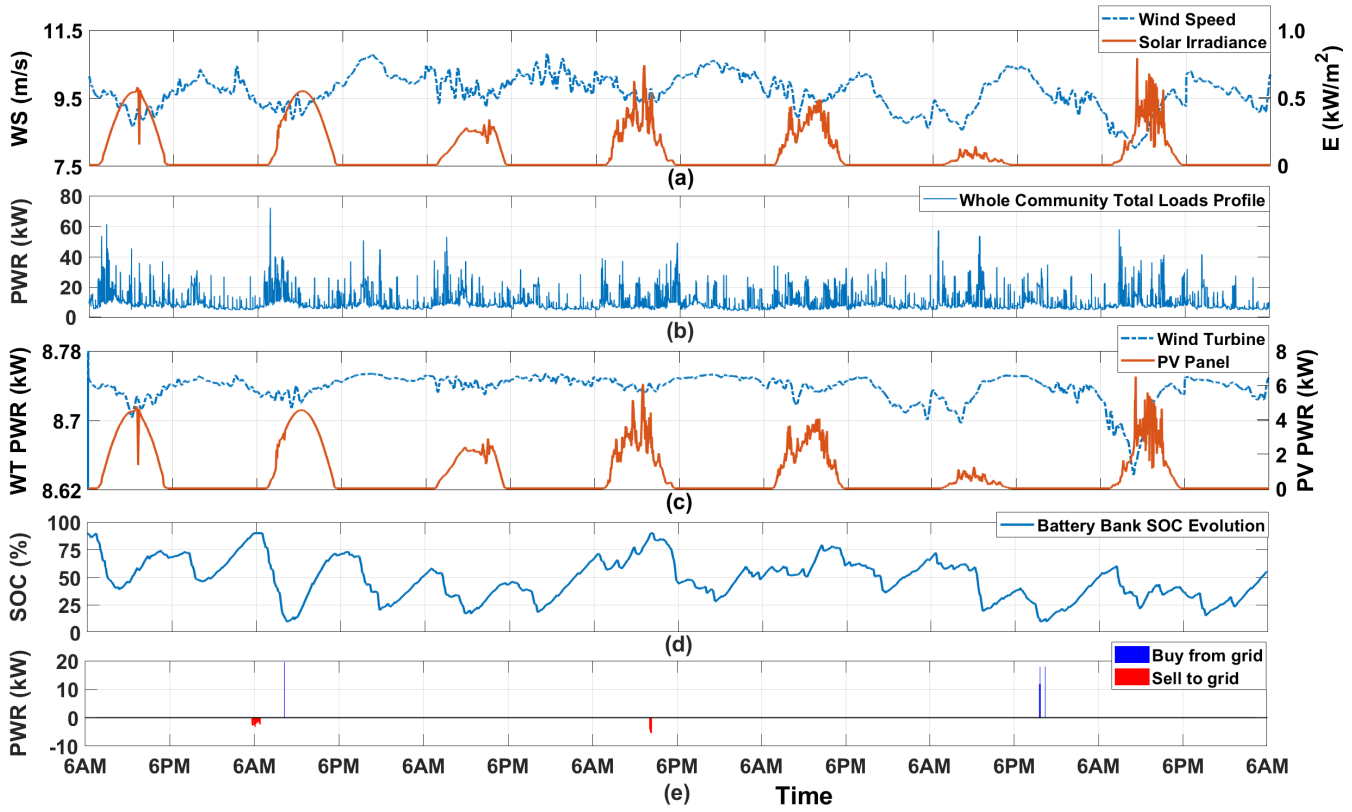


Figure 15. One-week long simulation of the CPEES. Irradiance and wind profiles (a); load consumption of the whole residential community (b); the power generated by the wind turbine and the PV array (c); SOC of the battery pack (d); the power flow with respect to the grid (e).

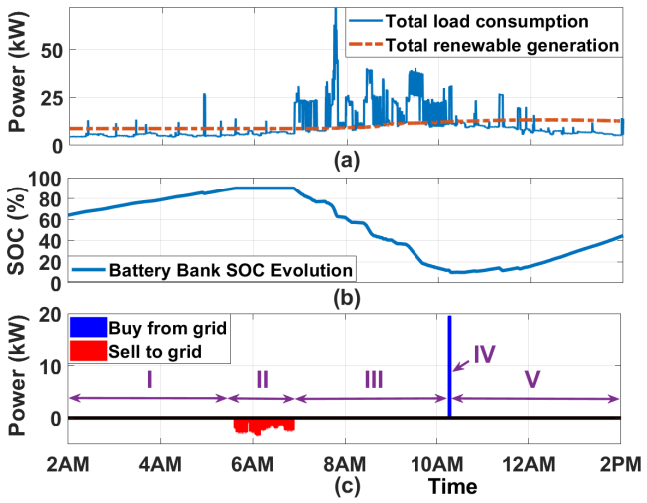


Figure 16. Zoom-in of some of the quantities plotted in Figure 8 for a 12-hour period: Total power consumption and sustainable generation (a); battery pack SOC (b); power flow to/from the grid (c), with emphasis on five periods involving different power flow balances.

the speedup achieved by SystemC-AMS with respect to Simulink for the same abstract modeling style was around 10-13x, which can be charged to the language and compiler efficiency. Conversely, Table 5 reports a much larger speedup, which is motivated by the aforementioned differences in the two frameworks.

As an additional element of analysis, we verified the impact of simulation time step on speedup (Figure 17). The data show that the speedup of SystemC-AMS obviously decreases for decreasing time steps, where Simulink does fewer calls to the solver over a given amount of time. However, this decrease is not particularly marked: even with a 10s time step our framework still guarantees a speed-up of about 250X. For a 1ms time step the speed-up reaches about 370X, and Simulink cannot even manage to complete the 5- and 6-day simulation due to memory space limitations.

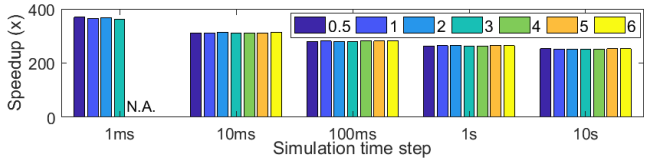


Figure 17. Simulation speedup of SystemC-AMS w.r.t. Simulink for different simulation time steps (x-axis) and simulation lengths (in days, different colors).

7 Design Space Exploration (DSE)

Thanks to its computational efficiency, our framework allows running Design Space Explorations (DSE) over significantly long time horizons. In this section, we present two DSE experiments on the CPEES of Figure 14. The first one carries out an exhaustive exploration of different configurations to determine the one with the largest profit; the second one adds a constraint on the initial investment.

7.1 DSE Setup

We consider the amount of power sources and of energy storage as parameters of the DSE. For the PV array and

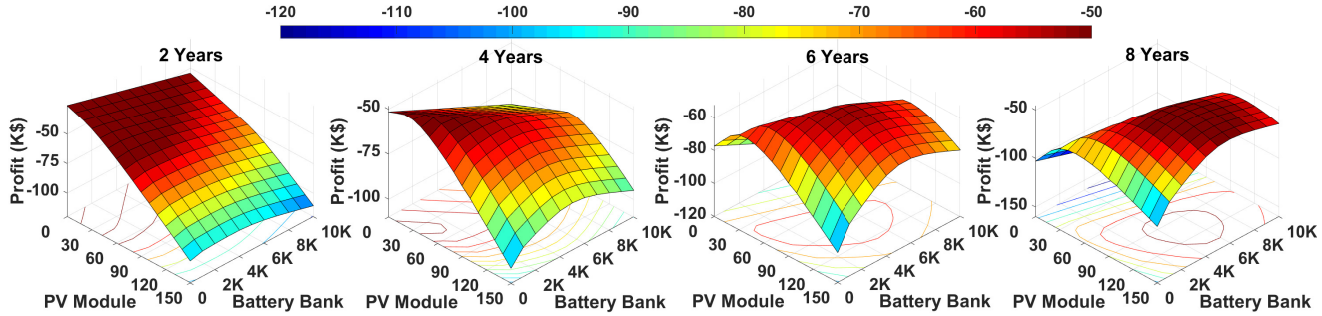


Figure 18. Total profit of different configurations for Case 1 and for different lengths of simulated time.

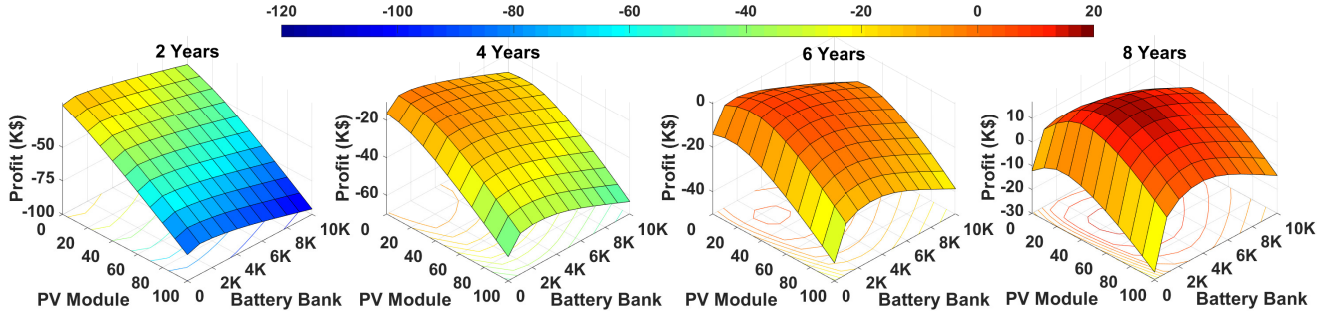


Figure 19. Total profit of different configurations for Case 2 and for different lengths of simulated time.

the battery pack, which are intrinsically modular, this corresponds to the number of PV modules and of battery cells; conversely, the wind turbine is considered either as included or removed. Table 6 indicates the cost information of these three elements. The total costs (device cost + installation cost) of wind turbine and PV array are $\$1.95/W$ and $\$2.25/W$, respectively, and their average operation and maintenance (O&M) cost is \$15/kW/Year [41]. We set the total cost of battery pack to $\$200/kWh$ [42] (one NCR18650 battery cell costs $\approx \$2.5$), and its O&M cost is \$10/kW/Year.

Table 6
Cost and lifetime of CPEES components.

Component	Cost [\$]	O&M Cost [\$/kW/Year]	Lifetime [Years]
Wind Turbine	19,500	15	20
PV Module	675	15	20
Battery Cell	2.5	10	8

For the electricity cost, we used the selling/buying prices reported in [43] (Table 7).

Table 7
Electricity Prices for different times of the day.

Price Category	Value (\$/kWh)	Time span
Buying F1	0.220	10am-3pm 6pm-9pm
Buying F2	0.215	7am-10am 3pm-6pm 9pm-11pm
Buying F3	0.200	11pm-7am
Selling	0.030	all day

The total profit of the CPEES is defined by Equation 1. We do not consider possible government incentives in our profit analysis due to the significant differences in local policies. Therefore, our profit analysis can be treated as a conservative one.

$$Profit = (P_{own} \times p_{buy}) + (P_{extra} \times p_{sell}) - (P_{grid} \times p_{buy}) - C_{capital} - C_{o\&m} \quad (1)$$

where:

P_{own} = Generated power for own use
P_{extra} = Extra generated power sold to the grid
P_{grid} = Power bought from utility grid
p_{buy} = Electricity buying price
p_{sell} = Electricity selling price
$C_{capital}$ = Capital investment for all devices
$C_{o\&m}$ = Global O&M cost

7.2 Exhaustive Exploration

The first experiment is an exhaustive DSE to determine the configuration with the highest economic benefit. The ranges of the variables are set as follows: when the wind turbine is not present (Case 1), the number of PV modules varies from 0 to 150 in steps of 10; when present (Case 2), the number of PV modules varies from 0 to 100. In both cases, the number of battery cells varies from 0 to 10,000, in steps of 1,000. We use the same AC loads and the same traces of irradiance and wind speed used in Section 6.

The exploration results for Case 1 are shown in Figure 18. The x- and y-axis represent the various configurations, in terms of the number of PV modules and battery cells. The z-axis represents the profit, computed as in Equation 1, where negative values denote a loss. The profits of various configurations in these 3D plots change from flat to convex surfaces when prolonging simulations up to 8 years. Interestingly, the results show that, in absence of incentives (which could lower the initial investment cost) there is no profitable configuration that can be obtained by any configuration of PV array and battery pack, even after 8 years of operations. Notice that the profit would further worsen after the eighth year, due to the cost of battery pack replacement.

When including the wind turbine (Figure 19), although the shapes of the curves are similar to Case 1, there exist some configurations that can lead to a profit after the sixth year. The main reason why the wind turbine brings such a benefit

is that it complements the absence of power generation by PV array during the night or during cloudy or rainy days, providing a better coverage of energy generation in particular during peak hours, which are not generally the most irradiated.

It is worth emphasizing that this analysis required year-long simulations, made possible only thanks to the fast simulation speed of the proposed framework: simulation of 1 year only took about 145s on average. Considering the 250-350x speedup discussed before, this would have required about a half day of simulation in Simulink.

Table 8
Highest profit configuration for different years.

Year	PV	Batt.	Profit[$]	Year	PV	Batt.	Profit[$]
1	0	0	-19,030	7	30	3,000	8,400
2	0	1,000	-16,260	8	40	3,000	17,220
3	0	1,000	-13,140	9	40	2,000	19,960
4	0	1,000	-10,010	10	50	2,000	28,730
5	10	2,000	-5,710	11	50	3,000	37,970
6	30	3,000	380	12	60	3,000	48,110

Table 8 lists the optimal points of the surfaces obtained by the above exploration at each year boundary and for 1 to 12 years horizon, with the relative configuration of PV modules and battery cells. Interestingly, the optimal solution corresponds to different configurations in different years. As expected, the profit increases with the length of the observation interval; notice however that this includes the cost of battery replacement after the eighth year, so the actual quantification is not trivial. In particular, it can be observed how it is not convenient to excessively increase the size of the battery pack, as the replacement cost will adversely affect the profit.

7.3 Exploration with Fixed Capital Cost

In the previous exploration there was no bound on the initially invested capital and only the maximization of the profit was sought. An alternative scenario is one where an initial investment is decided upfront and used a constraint. The exploration variables are the same as before, but in this case they will be constrained by the cost boundary. Based on the conclusion we drew in the previous analysis, we consider configurations that include one wind turbine; the actual available capital for PV modules and battery cells is therefore \$30,500, i.e., \$50,000−\$19,500 (the wind turbine cost). Table 9 lists 16 configurations with \$30,500 capital cost.

Table 9
Different configurations with same capital cost.

Config. No.	PV	Battery	Config. No.	PV	Battery
1	0	12,200	9	24	5,720
2	3	11,390	10	27	4,910
3	6	10,580	11	30	4,100
4	9	9,770	12	33	3,290
5	12	8,960	13	36	2,480
6	15	8,150	14	39	1,670
7	18	7,340	15	42	860
8	21	6,530	16	45	50

As a further variable, we selected two different locations: one in Arizona (dry and windy weather, with up to 90% sunny days) and the other in Oregon (with a mild wet climate).

Table 10
Optimal configurations at different years for each location.

	Oregon		Arizona	
Year	Config	Profit (K$)	Config	Profit (K$)
1	13	-43.674	12	-42.044
2	13	-36.545	12	-34.088
3	13	-31.024	12	-26.132
4	13	-22.602	12	-17.931
5	13	-18.373	12	-10.112
6	13	-8.483	12	-1.993
7	13	-5.723	12	4.291
8	13	1.584	12	14.299
9	15	1.994	13	15.199
10	14	8.134	13	23.549

Figure 20 shows the design exploration results for a time horizons from 1 to 10 years. Both surfaces exhibit a ditch after year 8 due to the cost of replacing the battery pack. The points with largest profit are significantly different for the two location: $\$8,134$ in Oregon vs. $\$25,230$ in Arizona, respectively. Both faster wind speed and higher irradiance in Arizona cause a much higher profit than in Oregon. Thus, the break-even of CPEES occurs at different times: after the eighth year in Oregon and after the seventh year in Arizona.

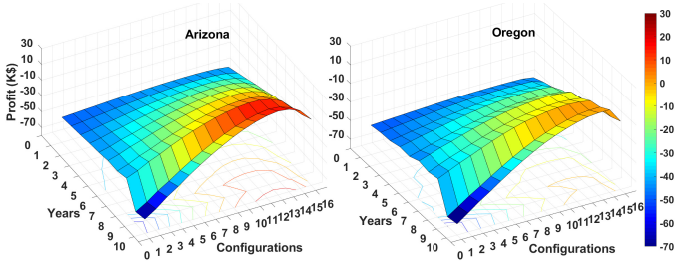


Figure 20. Profit statistics over ten years of the configurations listed in Table 9 at different locations.

Clearly, the optimal configurations are also different for the two locations. Table 10 shows the profit of the optimal configuration at each year for both locations. Interestingly, the optimal solution changes depending on the target horizon. With a eight year horizon, the best Oregon configuration is #13 (36 PV modules combined with 2,430 battery cells), while in Arizona is #12 (33 PV modules with 3,240 battery cells). The difference is due to the fact that in Oregon the lesser energy harvested by power sources is mostly consumed by the loads rather than stored in the battery, and a larger battery is not useful; conversely, in Arizona the availability of harvested energy motivates the use of larger batteries and fewer PV modules. Intuitively, the battery replacement event shifts the optimal solution to configurations with larger PV arrays and smaller battery packs: after the eighth year, the optimal configurations become #14 for Oregon (with 1,670 battery cells) and #13 for Arizona (with 2,480 battery cells).

Needless to say, running all these experiments over such long time intervals is possible only thanks to the extremely fast simulation speed of our framework.

8 Conclusions

An accurate assessment of the power flow in a CPEES and its relative economic implications requires accurate and efficient simulation tools that are also flexible in supporting different types of models. This paper proposed a SystemC-AMS framework that features all these requirements, allowing an efficient simulation of CPEES in which components are possibly represented with very different levels of details and of time semantics. We demonstrate the flexibility and high computing efficiency of our modeling and simulation framework by running different design space exploration experiments that span time horizons longer than 10 years while requiring only minutes of simulation time, still keeping excellent accuracy vs. state-of-the-art simulation tools.

References

[1] P. Lopion, P. Markewitz, M. Robinius, and D. Stolten, "A review of current challenges and trends in energy systems modeling," *Renewable and sustainable energy reviews*, vol. 96, pp. 156–166, 2018.

[2] E. Bellan, "JRC annual report 2016," https://ec.europa.eu/jrc/en/publication/annual-reports, pp. 1–38, 2016.

[3] G. Fulli, M. Masera, A. Spisto, and S. Vitiello, "A change is coming: How regulation and innovation are reshaping the European Union's electricity markets," *IEEE MPE*, vol. 17, no. 1, pp. 53–66, 2019.

[4] A. Maffei, S. Srinivasan, D. Meola, G. Palmieri, L. Iannelli, O. H. Holhjem, G. Marafioti, G. Mathisen, and L. Glielmo, "A cyber-physical systems approach for implementing the receding horizon optimal power flow in smart grids," *IEEE TSUSC*, vol. 3, no. 2, pp. 98–111, 2018.

[5] Y. Li, J. Si, S. Ma, and X. Hu, "Using energy-aware scheduling weather forecast based harvesting for reconfigurable hardware," *IEEE TSUSC*, vol. 4, no. 1, pp. 109–117, 2019.

[6] H. Xu, X. Jin, F. Kong, and Q. Deng, "Two level colocation demand response with renewable energy," *IEEE TSUSC*, 2019.

[7] H.-K. Ringkjøb, P. M. Haugan, and I. M. Solbrekke, "A review of modelling tools for energy and electricity systems with large shares of variable renewables," *Elsevier RSER*, vol. 96, pp. 440–459, 2018.

[8] L. Liu, H. Sun, C. Li, T. Li, J. Xin, and N. Zheng, "Exploring highly dependable and efficient datacenter power system using hybrid and hierarchical energy buffers," *IEEE TSUSC*, 2018.

[9] S. Yue, D. Zhu, Y. Wang, M. Pedram, Y. Kim, and N. Chang, "Simes: A simulator for hybrid electrical energy storage systems," in *Proc. of IEEE ISLPED*, 2013, pp. 33–38.

[10] Y. Kim, D. Shin, M. Petricca, S. Park, M. Poncino, and N. Chang, "Computer-aided design of electrical energy systems," in *Proc. of ICCAD*. IEEE, 2013, pp. 194–201.

[11] J. M. Molina, X. Pan, C. Grimm, and M. Damm, "A framework for model-based design of embedded systems for energy management," in *Proc. of IEEE MSCPES*, 2013, pp. 1–6.

[12] M. A. Al Faruque and F. Ahourai, "A model-based design of cyber-physical energy systems," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014, pp. 97–104.

[13] S. Vinco, Y. Chen, F. Fummi, E. Macii, and M. Poncino, "A layered methodology for the simulation of extra-functional properties in smart systems," *IEEE TCAD*, vol. 36, no. 10, pp. 1702–1715, 2017.

[14] J. Leonard, R. Hadidi, and J. C. Fox, "Real-time modeling of multi-level megawatt class power converters for hardware-in-the-loop testing," in *Proc. of IEEE EDST*, 2015, pp. 566–571.

[15] B. Palmintier, B. Lundstrom, S. Chakraborty, T. Williams, K. Schneider, and D. Chassin, "A power hardware-in-the-loop platform with remote distribution circuit cosimulation," *IEEE TIE*, vol. 62, no. 4, pp. 2236–2245, 2014.

[16] P. Palensky, E. Widl, and A. Elsheikh, "Simulating cyber-physical energy systems: Challenges, tools and methods," *IEEE TSMC*, vol. 44, no. 3, pp. 318–326, 2013.

[17] M. D. Ilic, L. Xie, U. A. Khan, and J. M. Moura, "Modeling of future cyber–physical energy systems for distributed sensing and control," *IEEE TSMCA*, vol. 40, no. 4, pp. 825–838, 2010.

[18] A. Banerjee, J. Banerjee, G. Varsamopoulos, Z. Abbasi, and S. K. Gupta, "Hybrid simulator for cyber-physical energy systems," in *Proc. of IEEE MSCPES*, 2013, pp. 1–6.

[19] P. Palensky, A. van der Meer, C. Lopez, A. Joseph, and K. Pan, "Applied cosimulation of intelligent power systems: Implementing hybrid simulators for complex power systems," *IEEE MIE*, vol. 11, no. 2, pp. 6–21, 2017.

[20] C. Unterrieder, M. Huemer, and S. Marsili, "SystemC-AMS-based design of a battery model for single and multi cell applications," in *Proc. of PRIME*, 2012, pp. 1–4.

[21] N. Chang, D. Baek, and J. Hong, "Power consumption characterization, modeling and estimation of electric vehicles," in *Proc. of IEEE ICCAD*, 2014, pp. 175–182.

[22] S. Vinco, Y. Chen, E. Macii, and M. Poncino, "A unified model of power sources for the simulation of electrical energy systems," in *Proc. of ACM GLS-VLSI*, 2016, pp. 281–286.

[23] B. Vernay, A. Krust, G. Schröpfer, F. Pêcheux, and M.-M. Louerat, "SystemC-AMS simulation of a biaxial accelerometer based on mems model order reduction," in *Proc. of IEEE DTIP*, 2015, pp. 1–6.

[24] K. Ohyama and T. Nakashima, "Wind turbine emulator using wind turbine model based on blade element momentum theory," in *Proc. of IEEE SPEEDAM*, 2010, pp. 762–765.

[25] K. Caluwaerts, D. Galayko, and P. Basset, "SystemC-AMS heterogeneous modeling of a capacitive harvester of vibration energy," in *Proc. of IEEE WAMS*, 2008, pp. 142–147.

[26] Y. Chen, E. Macii, and M. Poncino, "A circuit-equivalent battery model accounting for the dependency on load frequency," in *Proc. of IEEE DATE*, 2017, pp. 1177–1182.

[27] *SE4K–SE10K datasheet*, SolarEdge, 2017, www.solaredge.com/sites/default/files/se-three-phase-inverter-datasheet.

[28] *A-301/302-150 series datasheet*, Meanwell, 2016, www.meanwell-bg.com/files/M2017H1/A300-150-SPEC.PDF.

[29] "Openzmeter: An efficient low-cost energy smart meter and power quality analyzer," *MDPI Sustainability*, no. 11, 2018.

[30] M. Pipattanasomporn, M. Kuzlu, S. Rahman, and Y. Teklu, "Load profiles of selected major household appliances and their demand response opportunities," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, pp. 742–750, 2014.

[31] P. Meehan, C. McArdle, and S. Daniels, "An efficient scalable time-frequency method for tracking energy usage of domestic appliances using a two-step classification algorithm," *MDPI Energies*, vol. 7, pp. 7041–7066, 2014.

[32] National Instruments, "What is LabVIEW?" https://www.ni.com/en-us/shop/labview.html, 2019.

[33] Allegro MicroSystems, "ACS712: Fully integrated, hall-effect-based linear current sensor IC with 2.1 kVRMS voltage isolation and a low-resistance current conductor," https://www.allegromicro.com/en/Products/Sense/current-sensor-ics, 2019.

[34] C. A. Gueymard, "A review of validation methodologies and statistical performance indicators for modeled solar radiation data: Towards a better bankability of solar projects," *Renewable and Sustainable Energy Reviews*, vol. 39, pp. 1024 – 1034, 2014.

[35] S. K. Kim, J. H. Jeon, C. H. Cho, J. B. Ahn, and S. H. Kwon, "Dynamic modeling and control of a grid-connected hybrid generation system with versatile power transfer," *IEEE TIE*, vol. 55, no. 4, pp. 1677–1688, 2008.

[36] SUNPOWER, *300 solar panel*, https://www.energymatters.com.au/images/sunpower/sunpower-300.pdf.

[37] J. Kelly and W. Knottenbelt, "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes," *Scientific Data*, vol. 2, no. 150007, 2015.

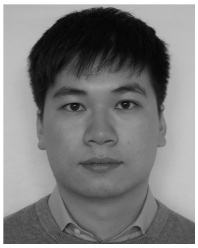
[38] S. Park, Y. Wang, Y. Kim, N. Chang, and M. Pedram, "Battery management for grid-connected PV systems with a battery," in *Proc. of ISLPED*, 2012, pp. 115–120.

[39] N. Liu, X. Yu, C. Wang, C. Li, L. Ma, and J. Lei, "Energy-sharing model with price-based demand response for microgrids of peer-to-peer prosumers," *IEEE TPWRS*, vol. 32, no. 5, pp. 3569–3583, 2017.

[40] National Renewable Energy Laboratory, "Measurement and Instrumentation Data Center," https://midcdmz.nrel.gov/.

[41] International Renewable Energy Agency, "Renewable power generation costs in 2017," https://www.irena.org/publications/2018, 2017.

[42] M. Safoutin, J. McDonald, and B. Ellies, "Predicting the future manufacturing cost of batteries for plug-in vehicles for the US environmental protection agency (EPA) 2017–2025 light-duty greenhouse gas standards," *WEVJ*, vol. 9, no. 3, p. 42, 2018.
[43] S. Magnani, L. Pezzola, and P. Danti, "Design optimization of a heat thermal storage coupled with a micro-chp for a residential case study," *Energy Procedia*, vol. 101, pp. 830–837, 2016.



Yukai Chen (M'15) received the M.Sc. degree in computer engineering and the Ph.D. degree in computer and control engineering from the Politecnico di Torino, Turin, Italy, in 2014 and 2018, where he is currently a Postdoctoral Research Fellow. His current research interest includes computer-aided design for integrated circuits and cyber physical energy systems, with particular emphasis on the modeling and simulation of extra-functional properties.

Sara Vinco (M'09) is Assistant Professor with tenure track at Politecnico di Torino, Turin, Italy. She received the Ph.D. degree in computer science from the University of Verona, Verona, Italy, in 2013. Her current research interests include energy efficient electronic design automation and techniques for the simulation and validation of heterogeneous embedded systems and cyber-physical production systems.

Daniele Jahier Pagliari (M'15) received the M.Sc. and Ph.D. degrees in computer engineering from Politecnico di Torino, Torino, Italy, in 2014 and 2018, respectively. He is currently an Assistant Professor in the same institution. His research interests include computer-aided design of digital systems, with particular emphasis on low-power optimization and approximate computing.

Paolo Montuschi (M'90-SM'07-F'14) is a Full Professor with the Department of Control and Computer Engineering and a member of the Board of Governors with Politecnico di Torino, Torino, Italy. His research interests include computer arithmetic, computer graphics, and intelligent systems. He is a Fellow of the IEEE, Life Member of the International Academy of Sciences of Turin, and of Eta Kappa Nu, the Honor Society of IEEE.

Enrico Macii (SM'02-F'05) is a Full Professor of Computer Engineering with the Politecnico di Torino, Torino, Italy. He holds a PhD degree in computer engineering from the Politecnico di Torino. His research interests are electronic digital circuits and systems, with a particular emphasis on low-power consumption aspects, energy efficiency, sustainable urban mobility, clean and intelligent manufacturing. He is a Fellow of the IEEE.

Massimo Poncino (SM'12-F'18) is a Full Professor of Computer Engineering with the Politecnico di Torino, Torino, Italy. His current research interests include several aspects of design automation of digital systems, with emphasis on the modeling and optimization of energy-efficient systems. He received a PhD in computer engineering and a Dr.Eng. in electrical engineering from Politecnico di Torino. He is a Fellow of the IEEE.