POLITECNICO DI TORINO
Repository ISTITUZIONALE

Modeling Cyber-Physical Production Systems with SystemC-AMS

Original
Modeling Cyber-Physical Production Systems with SystemC-AMS / Fraccaroli, Enrico; Vinco, Sara. - In: IEEE TRANSACTIONS ON COMPUTERS. - ISSN 0018-9340. - ELETTRONICO. - 72:7(2023), pp. 2039-2051. [10.1109/TC.2022.3226567]

Availability:
This version is available at: 11583/2973763 since: 2022-12-12T10:31:36Z

Publisher:
IEEE

Published
DOI:10.1109/TC.2022.3226567

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

30 April 2024

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

Modeling Cyber-Physical Production Systems with SystemC-AMS

Enrico Fraccaroli, *Member, IEEE,* Sara Vinco, *Member, IEEE,*

Abstract—The heterogeneous nature of SystemC-AMS makes it a perfect candidate solution to support Cyber-Physical Production Systems (CPPSs), i.e., systems that are characterized by a tight interaction of the cyber part with the surrounding physical world and with manufacturing production processes. Nonetheless, the support for the modeling of physical and mechanical dynamics typical of production machinery goes far beyond the initial application scenario of SystemC-AMS, thus limiting its effectiveness and adoption in the production and manufacturing context. This paper starts with an analysis of the current adoption of SystemC-AMS to highlight the open points that still limit its effectiveness, with the goal of pinpointing current issues and to propose solutions that could improve its effectiveness, and make SystemC-AMS an essential resource also in the new Industry 4.0 scenario.

Index Terms—SystemC-AMS, SystemC simulation kernel, cyber-physical production system, digital twin, non linear systems, mechanical models, schedulability, DAE solver, multi-discipline.

1 Introduction

Initially designed for modeling and simulating interacting mixed analog and digital functional subsystems, the Analog and Mixed-Signal (AMS) extension of SystemC, SystemC-AMS, has been adopted over time in several heterogeneous domains, including extra-functional and continuous-time ones. Table 1 reports some recent works that use SystemC-AMS in non-standard contexts, ranging from power and mechanical simulation to biological modeling: this proves that the impact of SystemC-AMS went far beyond the initial expectations.

The features that made SystemC-AMS a winning solution in such heterogeneous contexts are: its support for multiple Models of Computation (MoCs), that ensures *flexibility* to heterogeneous domains, and its adaptability to different abstraction levels, thus covering from system level (with the data-flow paradigm) down to digital and analog Hardware (HW) descriptions and signal flow constructs. Additionally, SystemC-AMS has a user-friendly C++-based syntax, supported by the standardization of *predefined primitives*. This allows a non-experienced user to model descriptions in domains such as frequency analysis and linear circuits. Such features made SystemC-AMS one of the reference languages for the development of virtual platforms, where it is used for implementing systems integrating the Software (SW) under development with accurate models of the HW and with physical components composing the underlying platform and environment [1], [2].

These aspects make SystemC-AMS a promising candidate in the context of Cyber-Physical Production Systems (CPPSs), i.e., systems where the cyber part (including SW, HW and electronic parts) is tightly interwoven with mechanical parts, mechatronic subsystems (i.e., sensors and actuators) and with the dynamics of the surrounding environment [18], [19]. In this scenario, a SystemC-based infrastructure would allow a holistic implementation of the different aspects of the CPPS, through the adoption of SystemC for the cyber part, SystemC-TLM for communication and early SW development, and SystemC-AMS for the dynamic, mechanic and physical subset of the system. The adoption of a common infrastructure would ease integration issues and allow the effective modeling of feedback loops between the different parts of the system (e.g., production machinery evolution and control SW). This solution would lead de facto to the construction of a SystemC-based simulatable model of the CPPS, thus allowing its validation and evaluation in varying application scenarios and configurations.

Even if the modeling of production machinery and of mechanical components goes well beyond its initial scope, SystemC-AMS could thus play an important role and be extremely useful in the construction of simulatable models where the heterogeneous nature of SystemC-AMS would find its maximum expression. Nonetheless, SystemC-AMS

Table 1: SystemC-AMS extra-functional application domains.

Domain	Application	References
Mechanical	MEMS and accelerometers	[3], [4]
	Fluidic systems	[5]
	Sensors	[6], [7]
	Automotive	[8], [9]
Electrical	MEMS and accelerometers	[3], [4]
	Sensors	[6], [7]
	Circuit equivalent model	[10], [11], [12]
Power	Direct current domain	[10], [13], [14], [15]
	Alternating current domain	[14]
Thermal	Circuit-equivalent models	[11], [13]
Others	Chemical and biological	[16], [17]

- E. Fraccaroli is with the Department of Computer Science, University of Verona, Verona, Verona 37129, Italy. E-mail: enrico.fraccaroli@univr.it.
- S. Vinco is with the Department of Control and Computer Engineering, Politecnico di Torino, Torino 10129, Italy. E-mail: sara.vinco@polito.it.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

currently is one step back w.r.t. its competitors, as its adoption in the context of CPPS and digital twin development is mostly limited to the cyber part, with ad-hoc cosimulations built to support the production and manufacturing subsystem [20].

This work analyses the *current status of SystemC-AMS from the perspective of CPPS development*. The analysis is based both on a detailed study of the SystemC-AMS simulation kernel and on a number of case studies that will highlight different aspects of the problem.The adoption of SystemC and SystemC-AMS is widespread and well established for modeling of the *cyber* part. Thus, the focus of this paper is mostly on the *physical* and mechanical subset of a CPPS: such domains go beyond the initial scope of SystemC-AMS, and thus their support still hides several limitations. The highlighted issues span across different levels of seriousness and affect different aspects of SystemC-AMS, ranging from scheduler-related and solver-related issues, caused by constraints on the schedulability or solvability of the systems, to issues related to the current implementation of the standard and possible extensions. The analysis starts from case studies and solutions proposed at state-of-the-art to make a point on CPPS support. Our goal is thus not to provide novel solutions per se, but to rather provide the reader with an analysis of the current issues of SystemC-AMS in the CPPS perspective, corresponding possible viable solutions (either available in the literature or proposed by this work), and draw a picture of the possible future evolution of SystemC-AMS.

The paper is organized as follows. Section 2 explains what CPPSs are and recaps the main features of SystemC-AMS, in terms of supported MoCs and of simulation infrastructure. Section 3 presents a set of reference case studies, either developed by the authors or taken from the literature. Section 4 provides a more thorough description of the adoption of SystemC-AMS in heterogeneous systems. Section 5 can be considered the core of this work, as it focuses on the discussion of the issues posed by the CPPS context and highlighted by the case studies, to propose currently available solutions and possible future improvements. Finally, Section 6 draws our conclusions.

2 Background

This section offers an explanation of what CPPSs are, and an introduction to the SystemC-AMS language and to its organization and simulation kernel.

2.1 Cyber-Physical Production System

Cyber-Physical Systems (CPSs) are systems of collaborating computational entities which are in intensive connection with the surrounding physical world and its on-going processes. They provide and use, at the same time, data-accessing and data-processing services [21]. Cyber-Physical Production Systems (CPPSs) apply this concept to the manufacturing context, so that the physical aspects being modeled include also mechanical, mechatronic and production processes with complex dynamics that must be monitored, controlled and influenced both adaptively and intelligently. Thus, The main feature of CPPSs is its multi-disciplinary nature, in a sense that they require to bridge mechanics, electronics, engineering, control and computation [22]. Their inherently complex nature, affects also the development and simulation frameworks, that must span across multiple domains and consider them not separately but rather jointly, as their integration, interaction and inter-dependency are crucial.

As highlighted by the survey in [22], this complex design process can be effectively supported only by multi-discipline modelling and simulation frameworks that span across multiple domains. They must provide designers support for validating, evaluating and optimizing their design in a single modeling environment. On the one hand, such framework must be cross-multi-discipline, including support for different physical domains, modeling strategies and time models, i.e., continuous-time (e.g., dynamics of physical components) and discrete-time (for digital computing components). On the other hand, the frameworks must be capable of handling different time rates: different physical domains are indeed characterized by different time constants (e.g., nanoseconds for digital components, milliseconds for power, seconds for temperature [13]), and the simultaneous presence of such multiple time scales must be handled with care, not to make the simulation too slow and heavy.

2.2 SystemC-AMS

SystemC has been introduced in the early 2000s to answer to the need of HW-SW co-design imposed by embedded systems: on one hand, it reproduces the typical behaviors and constructs of HW description languages; on the other, the adoption of C++ as base language eases the integration of SW as complex as operating systems [23].

2.2.1 Models of Computation (MoC)

The evolution of technology made systems tightly connected to the environment and "smart", i.e., capable of sensing, actuating and reacting to the evolving conditions surrounding the system itself. This required the definition of its AMS extension, targeting interacting mixed analog and digital functional subsystems.

SystemC-AMS provides different MoCs to cover a wide variety of domains (Figure 1):

- *Timed Data-Flow (TDF)* models data processing systems: processes are activated with a precise frequency, elaborate input tokens and produce output tokens. This allows to schedule processes statically, by considering their activation time step and producer-consumer dependencies.
- *Linear Signal Flow (LSF)* supports the modeling of continuous time behaviors through a library of pre-defined primitive modules (e.g., integration, or delay), each associated with a linear equation.
- *Electrical Linear Network (ELN)* models electrical networks through the instantiation of predefined linear primitives, e.g., resistors or capacitors, where each primitive is associated with an electrical equation.

The characteristics of the three MoCs are deepened in Section 4, where their adoption is exemplified on examples and discussed in detail.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

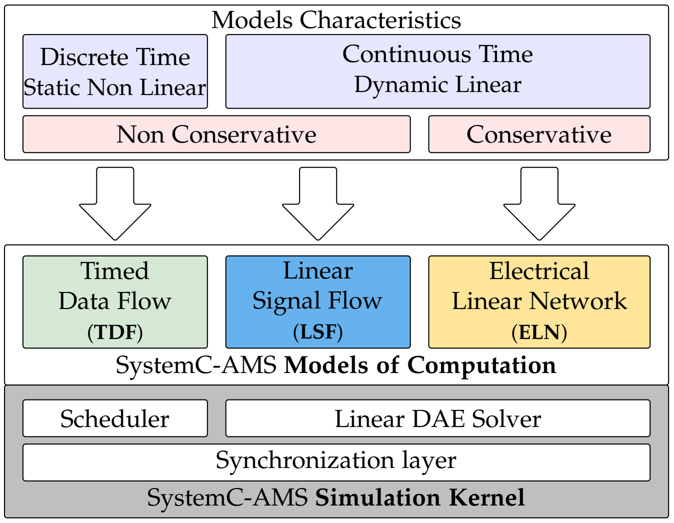


Figure 1: SystemC-AMS simulation kernel and supported MoCs.

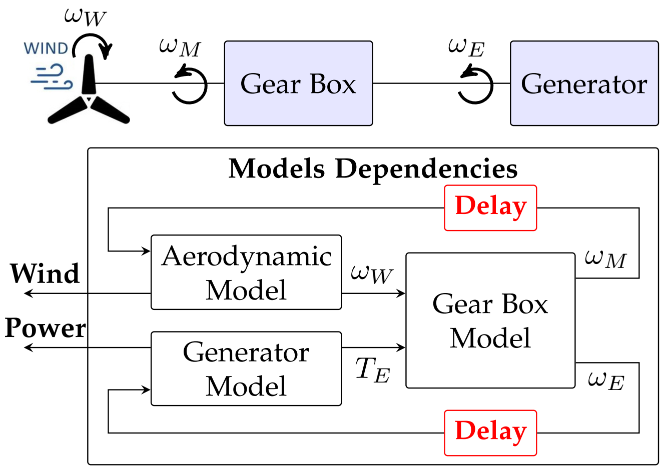


Figure 2: Structure of the wind turbine model (ω is angular speeds, T is torque, subscript W stays for wind, M for mechanical and E for electrical).

2.2.2 SystemC scheduler extension for AMS simulation

The SystemC language uses an event-based architecture, where a centralized scheduler controls the execution of processes based on events, i.e., synchronizations, time notifications or signal value changes. The SystemC-AMS simulation kernel enriches the standard SystemC kernel with three additional blocks (Figure 1).

A *TDF scheduler* groups all TDF modules in clusters of interconnected modules, and builds a static schedule of each cluster depending on time step, activation rate and dependencies of the modules.

A *linear Differential-Algebraic Equation (DAE) solver* is used to handle ELN and LSF descriptions: it analyzes the ELN and LSF instantiated primitives to derive the underlying equations, that are solved to determine system state at any simulation time. The solver uses lightweight numerical methods, i.e., backward Euler and trapezoidal methods, combined with optimization techniques, e.g., Lower-Upper (LU) decomposition and Woodbury formulas, to speed up matrix factorization, to provide a reasonable level of accuracy and guarantee at the same time good simulation performance [24], [25].

Finally, a *synchronization layer* uses the activation time step of each module, primitive and cluster to insert the execution of SystemC-AMS items in the standard SystemC simulation flow. The synchronization layer stores the static schedule of a cluster of components (i.e., modules or primitives) built by the TDF scheduler as a *list of pointers to the cluster components*:

- cluster activation period is determined by the module with largest time step;
- the other modules are then listed by reflecting their producer-consumer dependencies, and they may be replicated a number of times, depending on their activation time step: if a module is activated multiple times in a cluster period (i.e., the cluster period is a multiple of its time step), one activation entry is listed for any time that the module should be activated.

This ensures a lightweight modeling of the static schedule, as the activation list allows the scheduler to simply activate all cluster modules in the correct order at simulation time with no additional runtime scheduling overhead.

Recently, SystemC-AMS has been extended with the design environment COSIDE, developed by COSEDA [26], with the goal of easing the adoption of SystemC-AMS and of speeding up the modeling of analog and mixed-signal hardware and software systems. COSIDE includes support for modeling and verification, through libraries of predefined components, automatic code generation, a graphical interface, and Universal Verification Methodology (UVM) test-bench generation and assertion-based mixed-signal verification support.

3 Reference Case Studies

This section introduces a set of case studies that required a non-conventional modeling approach, both developed by the authors or taken from the literature. The choice fell on those case studies that better highlighted the potentialities and the limitations and open issues of SystemC-AMS.

3.1 Wind turbine

The first case study is a wind turbine, part of a more complex CPPS reproducing an electrical energy system, that includes also houses (demanding power), a battery (feeding the system when wind power production can not provide enough energy), and a connection to the grid, to occasionally buy or sell power. Such case study requires covering very heterogeneous domains: as depicted in Figure 2, the wind turbine model includes environmental quantities (i.e., the wind activating the blades), mechanical and aerodynamic equations (to convert to mechanical power, modeled by the gear box), and electric power generation (encapsulated by the generator). The model of the wind turbine used in this work is based on [27], that reports in full detail the underlying equations.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

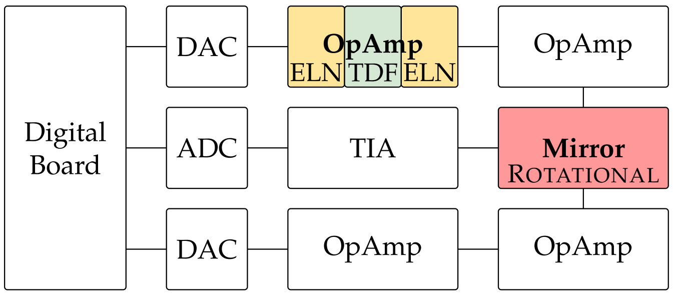


Figure 3: Structure of the pico-projector, with MoCs used to build the non-linear Operational Amplifier (OpAmp) and presence of conservative non-electrical domains.

3.2 Digital Pico-Projector

The second case study is a pico-projector[1], a Micro-Opto-Electro-Mechanical System (MOEMS) implemented by combining Micro-Electro-Mechanical System (MEMS) and micro-optics technologies. The pico-projector modulates three laser sources, whose light converges into a single beam deflected by a micro-mirror. The mirror is moved by drivers, in accordance to the decisions taken by a digital board (Figure 3). The signals provided by the digital board are transmitted to the circuitry controlling the mirror through a pair of two-stage OpAmps. The main issue of this component is caused by the presence of non-linear behaviors of the following components: the two-stage OpAmps, the micro-mirror, and the TransImpedance Amplifier (TIA). Another issue is the heterogeneity of the domains involved: digital for the board, rotational (linear and acceleration) for the mirror, and electrical for the other blocks. Let us look at the amplifier on top of Figure 3, that exemplifies how its behaviour has been achieved by combining together different MoCs, to circumvent some of the language limitations. The component is interfacing with the Digital-to-Analog Converter (DAC) and the subsequent OpAmp through an electrical interface written in ELN, while the internal non-linear behaviour is implemented by using a TDF module.

3.3 Direct Current (DC) Motor

A DC motor is a typical component of a CPPS, and it is characterized by a high degree of heterogeneity, caused by the simultaneous presence of electrical and rotational kinematic modeling. An example of DC motor implemented in SystemC-AMS has been proposed in [8]: the case study includes an electro-mechanical subsystem with a DC motor that manipulates throttle position, an Electronic Control Unit (ECU) that determines the required throttle position given the current state of the vehicle, and a control board. The focus here is restricted to the DC motor subsystem: Figure 4 exemplifies its structure and highlights at the same time the heterogeneity of domains involved.

3.4 Diode

Diodes are key components of a variety of systems, ranging from circuits to photovoltaic modules, sensors and electro-mechanical components. The most critical characteristic of diodes is their well-known non-linear behaviour (characterized by a very steep exponential current-voltage curve), that collides with the limitations of SystemC-AMS. Linearized or Piecewise Linear (PWL) implementations proposed in the literature provide appreciable accuracy only with very small time steps (in the order of nanoseconds), that heavily affect the simulation speed of the surrounding system. A solution to the SystemC-AMS implementation of diodes has been proposed by [28], and its structure can be summarized by the simplified circuit of Figure 5. The diode is divided into two parts, implemented in ELN and TDF. The electrical part contains a TDF-controlled resistor and two inductors that connect the two pins of the resistor to the output pins (i.e., p and n). The TDF part instead controls the resistor value based on the potential drop between the the two pins of the controlled resistor. The issues related to the diode's behaviour are more thoroughly discussed later on in the paper.

1. Design provided by STMicroelectronics in the context of SMAC (SMArt systems Codesign) European Project (FP7-ICT-2011-7-288827).

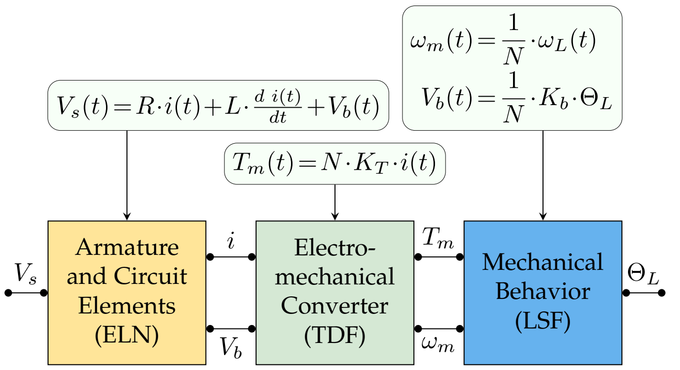

$\omega_m(t) = \frac{1}{N} \cdot \omega_L(t)$
$V_b(t) = \frac{1}{N} \cdot K_b \cdot \Theta_L$
$V_s(t) = R \cdot i(t) + L \cdot \frac{d\,i(t)}{dt} + V_b(t)$
$T_m(t) = N \cdot K_T \cdot i(t)$
V_s
Armature and Circuit Elements (ELN)
i
V_b
Electro-mechanical Converter (TDF)
T_m
ω_m
Mechanical Behavior (LSF)
Θ_L

Figure 4: Structure of the DC motor, with the corresponding partitioning into SystemC-AMS MoCs.

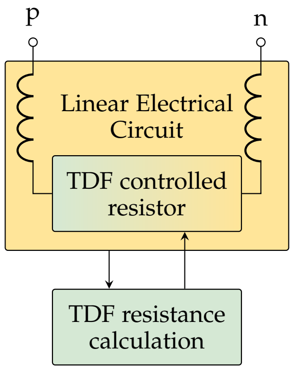


Figure 5: Structure of the diode, the piece-wise linear behaviour is given by the control structure which changes the resistor value.

4 Multi-domain modeling with SystemC-AMS

SystemC-AMS does not formalize how to adopt its MoCs in domains other than the functional one or the electrical one. As a consequence, works in the literature going beyond such domains focus on the specific challenges given by the target application scenario. This section tries to recap the main solutions adopted and to provide a uniform view of

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

the modeling strategies given the kind of description. The main dimensions that have to be considered to identify the kind of description are:

- The *level of abstraction*, which includes time management (i.e., discrete time or continuous time) and the level of detail of the description (e.g., based on equations or on transfer functions).
- The *adherence to conservation laws*, specifically the attention that has to be devoted to conservation laws is a direct consequence of the lack of definition of conservative domains other than the electrical one.

The electric domain has indeed a devoted abstraction level, ELN, that directly applies conservation laws: whenever the designer is describing an electric network, he can choose between a more abstract model, based, e.g., on equations that must explicit conservation laws, or mapping directly onto ELN constructs, that implicitly apply Kirchhoff's conservation laws. For any other kind of conservative description, the designer has to explicit also those dependencies that originate by the intrinsic conservation laws of the domain of interest. This is a limitation of SystemC-AMS w.r.t. other AMS languages like Verilog-AMS [29]. Verilog-AMS is a standard modeling language, primarily used to design analog circuits, which allows defining nets of different *disciplines*. A discipline associates *potential* and *flow* natures (e.g., voltage, current, force, position, velocity, etc.) for conservative system. Verilog-AMS bases the analog behaviour of conservative systems on the generalized Kirchhoff's Potential and Flow Laws (KPL and KFL).

$$\lambda = \frac{\omega_M \cdot R}{v}$$

$$\gamma = \frac{\lambda}{1 - \lambda \cdot c_1}$$

$$C_p = c_2 \cdot \left(\frac{c_3}{\gamma} - c_4 \right) \cdot e^{\frac{c_5}{\gamma}}$$

$$T_w = \frac{\frac{\Pi}{2} \cdot C_p \cdot \rho \cdot R^2 \cdot v^3}{\omega_M}$$

ω_M	Rotor Angular Velocity
R	Blades Radius
v	Wind Speed
λ	Tip Speed Ratio
C_p	Power Coefficient
ρ	Air Density
T_w	Rotor Torque
c_i	Turbine-Specific Coeff.

```
1  SCA_TDF_MODULE (aerodynamic) {
2    sca_tdf::sca_out<double> Tw;
3    sca_tdf::sca_in<double> v;
4    sca_tdf::sc_in<double> Wm;
5    ...
6  };
7  void aerodynamic::initialize() {
8    Tw.set_timestep(1, sc_core::SC_MS);
9    ro = 1.225;
10 }
11 void aerodynamic::processing(){
12   lambda = varWm * radius / wind.read();
13   gamma = lambda/(1-0.035*lambda);
14   Cp = 0.5*(116/gamma-5)*exp(-21/gamma);
15   Tw.write(
16     (PI / 2) * Cp * ro * pow(radius,2)
17     * pow(v.read(), 3) / Wm.read()
18   );
19 }
```

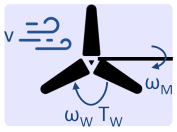


Figure 6: Aerodynamic model of the wind turbine (top) and corresponding SystemC-AMS TDF code (bottom).

4.1 Discrete time models

This class of models represents component evolution as a function, a polynomial, an algebraic model or even a waveform of values over time. This kind of models is not restricted to functionality description (e.g., standard Register-Transfer Level (RTL) models of digital HW), but it rather includes abstracted non-functional models (e.g., Peukert's model, reproducing battery dynamics through equations) [5], [10] and discretized or linearized implementations of non-linear models (e.g., obtained via model order reduction or through abstraction [7], [30]). This solution has been adopted for the pico-projector case study, where the non-linear and PWL behaviors of the OpAmp components have been implemented as a set of equations controlled by conditions. It is important to remember that a discrete time model must incorporate all necessary equations, as conservation laws do not apply.

Discrete time models perfectly fit the TDF MoC. Figure 6 exemplifies this for the aerodynamic model of the wind turbine: the model is described as equations (top) that determine the rotor torque depending on the angular velocity of the rotor, blade radius and the characteristics of the wind turbine. The model is encapsulated into the `processing()` function, that is repeatedly executed over time at any activation of the component (lines 10–15). The `initialize()` method allows to set the activation time step of the current module, thus tuning how often the function shall be executed (lines 6–9). It is important to note that the IEEE 1666.1-2016 standard extended TDF with the possibility to dynamically change time step and data rate of processes, to handle more efficiently sporadically changing signals and behaviors where frequencies, time steps and data rates change over time. However, the driver function adjusting dynamically the time step must be written by the designer, and it strictly depends on both the desired level of accuracy and the kind of component (e.g., accurate to the clock cycle for RTL models, while models of physical phenomena like temperature or power consumption might update in the order of the tens of milliseconds).

4.2 Dynamic systems

Dynamic systems represent a physical system that evolves over time by preserving a certain notion of memory, i.e., output values depend both on inputs and on system status in the previous time instants. Dynamic systems are modeled in different ways, ranging from differential and algebraic systems of equations to transfer functions. SystemC-AMS restricts the scope to linear dynamic models: it provides specific LSF constructs for linear elements, e.g. gains and integrators, transfer functions in the frequency domain (used for mechanical, chemical and fluidic systems [5], [16]) and state-space equations (that allow the modeling of MEMS, power models and mixed-technology systems [7], [31]). Such constructs can be mapped onto the corresponding LSF primitives, that are instantiated and connected in a way that reproduces signal propagation.

As an example, Figure 7 shows the equations deriving the angle θ between the wind turbine rotor and the gen-

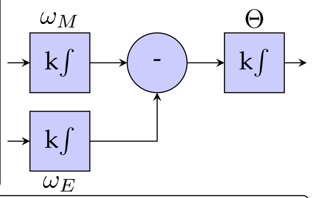
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

$$\Theta = \int (\omega_E - \omega_M)$$

$$\omega_M = \int massmodel(T_M, T_W)$$

$$\omega_E = \int massmodel(T_E, T_M)$$

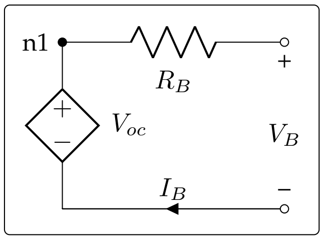


```
sca_lsf::sca_integ
    Wm("Wm", 1.0, 2.10001),
    Wg("Wg", 1.0, 2.2192116114),
    Theta("Theta", 1.0, 1.000268);
sca_lsf::sca_sub SubW("subW");
...
Wm.x(...);
Wm.y(idtWm);
Wg.x(...);
Wg.y(idtWg);
SubW.x1(idtWg);
SubW.x2(idtWm);
SubW.y(sumW);
Theta.x(sumW);
Theta.y(thetaSig);
```

Figure 7: Equations to derive the angle θ between the wind turbine rotor and the generator rotor from their angular velocities and their mass model (left) and corresponding LSF model (described in the block diagram notation, right, and as SystemC-AMS code, bottom).

$$V_{OC} = V_B + R_B \cdot I_B$$

V_{OC}	Open Circuit Voltage
R	Internal Resistance
V_B	Battery Voltage
I_B	Current Demand



```
1  SC_MODULE (battery){
2    sca_tdf::sca_in<double> I;
3    sca_tdf::sca_out<double> SOC;
4    sca_eln::sca_vsource Voc;
5    sca_eln::sca_r Rb;
6    sca_eln::sca_node n1;
7    ...
8    SC_CTOR (battery) :
9        Voc("Voc"),
10       Rb("Rb") {
11     Voc.p(n1);
12     Rb.n(n1);
13     ...
14   }
15 };
```

Figure 8: Circuit model of a battery (left) and corresponding implementation in SystemC-AMS (right), by mapping circuit elements to ELN primitives.

erator rotor from their angular velocities. The difference between the angular speed of the turbine rotor and of the generator rotor is mapped onto a `sca_lsf::sca_sub` primitive (round primitive), while the integrators are mapped onto `sca_lsf::sca_integ` primitives (square primitives).

As shown in the previous section, the pico-projector presents a non-linear behaviour inside its operational amplifiers. Part of their internal behaviour is implemented in the frequency domain with Laplace transfer functions. In SystemC-AMS Laplace transfer functions are implemented by using the primitives `sca_lsf::sca_ltf_nd` and `sca_lsf::sca_ltf_zp`, which are the numerator-denominator form and the zeros-poles form respectively.

4.3 Circuit models

The ELN MoC describes the electrical domain by providing a set of predefined circuit elements. A circuit model can thus be straightforwardly implemented by connecting such primitives: each primitive will define one (or more) circuit equations, that will be enriched by the application of Kirchhoff's conservation laws. Note that this modeling style has been used over time to describe also circuit-equivalent models, i.e., models that emulate the behavior of a non electrical component through an equivalent electrical circuit. This approach applies to a wide range of domains, including power simulation [10], thermal simulation [11], fluidic systems [5] and MEMS [4], but requires deep knowledge of both domains.

An example is provided in Figure 8, that shows a circuit equivalent model of the battery [32] included in the energy system case study. The circuit is implemented by mapping its elements to ELN primitives: e.g., voltage source V_{OC} is instantiated as an ELN `sca_eln::sca_vsource` object (lines 9-10), and resistor R_B as a `sca_eln::sca_r` object (lines 11-12).

4.4 Mapping complex models to modeling styles

To determine the correct implementation of a component, it is necessary to understand its characteristics in terms of time management, presence of conservation laws and kind of constructs adopted (e.g., integrative, state space equations). This allows to easily map the component model onto one of the three aforementioned modeling styles, and to adopt the suitable SystemC-AMS MoC for its implementation. Note that a model may contain a mix of such modeling styles, e.g., combine circuit elements with signal flow elements or equations. In this scenario, the model must be divided in sub-blocks, each mapped to separate SystemC-AMS modules following the corresponding modeling style.

A clear example is given by the DC motor presented in Section 3.3. Figure 4 clearly exemplifies its heterogeneous nature: the armature circuit of the DC motor and the corresponding circuit elements are mapped to ELN, the mechanical behavior is implemented as LSF primitives, and TDF is used for connection and to implement the electromechanical converter behavior. This process has been applied also to the simulatable model of the energy system, as clear from the former subsections, and to the pico-projector (Figure 3): the amplifier has been implemented by combining together ELN for interfacing with the DAC and the subsequent OpAmp, while the internal non-linear behaviour is implemented by using TDF.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

5 Issues and possible future improvements

This Section discusses the main issues we encountered when modeling the non-standard systems in SystemC-AMS, with a focus on the example case studies listed in Section 3. The issues can be organized in five classes, depending on the affected part of SystemC-AMS and on the seriousness and impact on the standard:

A. *scheduler-related* issues, caused by constraints on the schedulability of systems;
B. *solver related issues*, attributed to the current linear DAE solver;
C. *MoC-related* issues, that are specific of each MoC;
D. *implementation* issues, caused by the current implementation of the SystemC-AMS simulation kernel;
E. *extensions* that would improve the effectiveness of SystemC-AMS.

5.1 Schedulability issues

Section 2.2 explained that the basic concept of SystemC-AMS scheduler is the construction of islands (called *clusters*) of interconnected modules and primitives that can be scheduled statically to ensure effective insertion in the SystemC simulation cycle, thus speeding up simulation. The assumption is that no cyclic dependency occurs, and that it is always possible to determine a sequence of activations of the modules and primitives, possibly allowing different activation rates and thus a certain degree of buffering. However, this synchronization strategy may encounter a number of schedulability issues.

5.1.1 *Enhanced support for schedulability analysis*

The most typical one is caused by *cyclic dependencies*, typical of TDF modules. Cyclic dependencies are quite easy to find: the wind turbine presented in Section 3.1 hides cyclic dependencies between its main components, that mutually influence each other as an effect of the drive-train dynamics (as highlighted by the arrows in Figure 2); the DC motor includes a cyclic dependency between its electrical part and its mechanical part. Most of such dependencies are easy to solve by adding a delay (i.e., `set_delay()` function in TDF, or `sca_lsf::sca_delay` primitive of LSF) [8], [33]. However, when the SystemC-AMS system becomes large, with heterogeneous models working at different time steps and rates, the source of error is often difficult to identify. SystemC-AMS discovers such issues only at runtime, when the synchronization layer tries to build the schedule of the components: a schedulability error aborts simulation, and is notified together with a list of the chained modules and primitives that created the issue. This may be very difficult to debug, above all when clusters and execution chains include a large number of items: the reported list is indeed often not complete, thus reducing the visibility of the possible sources of error.

A different kind of synchronization issues is generated by the integration with the SystemC scheduler: TDF clusters may indeed generate *causality issues*, when events generated by TDF are in the past with respect to the current advance of time determined by the SystemC kernel. An interesting analysis on this issue has been proposed by [34].

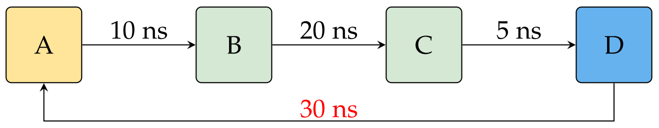


Figure 9: An example of a graphical representation that could help debugging in the presence of scheduling problems. The annotation above edges represents the rate of each output port. This example shows how this representation can help pinpoint the schedulability issue.

Both these classes of issues may be solved by providing support for *enhanced static schedulability analysis*. As an example, the work in [35] proposes a static analysis of TDF modules, that identifies schedulability issues both in the TDF and the continuous time parts and proposes possible solutions for their scheduling. [34] instead proposes a pre-simulation symbolic analysis to identify schedulability issues, implemented by mapping SystemC-AMS constructs on Petri nets. Providing tools like the aforementioned ones, and even integrating them in the COSIDE framework, would allow the user to quickly identify the sources of errors and to construct a feasible schedule with minor updates to the system.

The dynamic scheduler poses new problems from the point of view of error detection and schedulability analysis in general. The scheduler has some trouble conveying in a human-friendly way the necessary information for pinpointing the issue: the current debugging features simply list the modules and primitives that are part of a cluster that generates schedulability issues, with no additional information about the time steps or the rates that generate the issues. One way the scheduler might help the user would be to provide a graphical representation of the current schedule, like the one in Figure 9. Specifically, SystemC-AMS should provide a graphical representation of the relations between the processes annotated with their time step, and for each class, it should provide the list of activation chain. An hypothetical representation in shown in Figure 9. This graphical representation could contain other information useful for finding the scheduling problem, the example of Figure 9 shows only the output ports rates, which sometimes might be enough. Tools and algorithms like `dot` from the `graphviz` library might be used to generate the directed graphs for the graphical representation.

5.1.2 *Supporting dynamic systems*

Some schedulability issues are however impossible to solve by simply adding explicit delays or carefully looking at module settings. This is the case of *dynamic systems*, that are widely adopted for the modeling of mechanical and physical components. Dynamic systems are modeled through systems of equations (both algebraic and differential) that must be evaluated simultaneously, rather than sequentially. This creates dependency loops that can not simply be broken with a delay: at any time step, it is indeed necessary to re-evaluate the system of equations a number of times, until convergence is reached. Advanced dynamic system simulators, like Simulink stateflow, are equipped with algebraic loop solvers, that requires no user intervention [36].

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

$$
\begin{aligned}
\varphi_{DS} &= I_{DS} + I_{DR} \\
\varphi_{QS} &= I_{QS} + I_{QR} \\
I_{DS} &= \varphi_{QS} \\
I_{QS} &= \varphi_{DS} \\
\varphi_{DR} &= I_{DS} + I_{DR} \\
\varphi_{QR} &= I_{QS} + I_{QR} \\
\frac{d\varphi_{DR}}{dt} &= \varphi_{QR} \cdot \omega_E + I_{DR} \\
\frac{d\varphi_{QR}}{dt} &= \varphi_{DR} \cdot \omega_E + I_{QR} \\
T_E &= \varphi_{QR} \cdot I_{DR} + \varphi_{DR} \cdot I_{QR}
\end{aligned}
$$

ω_E	Angular Velocity
φ	Magnetic Flux
T_E	Electrical Torque
DS	Stator Direct
QS	Stator Quadrature
DR	Rotor Direct
QR	Rotor Quadrature

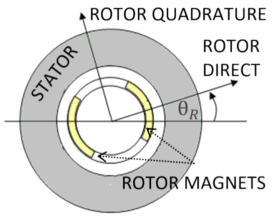


Figure 10: Equations of the wind turbine generator causing cyclic dependency.

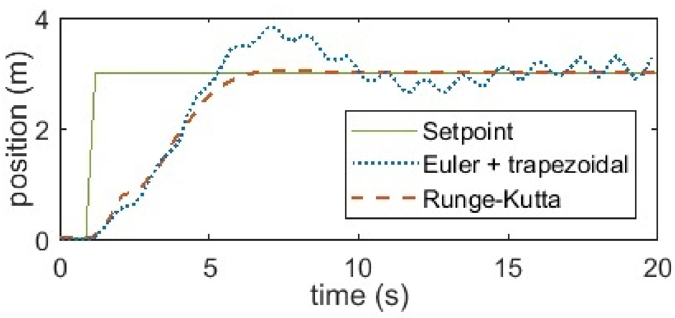


Figure 11: Example of SystemC-AMS solver instability on a marginally stable physical system (modified from [24]).

SystemC-AMS does not provide support for this kind of scenarios: any simulation time step executes modules and primitives exactly once, with no notion of stability and convergence. This prevents the construction of complex physical models.

An example is provided by the wind turbine, whose generator falls in this category. The generator is described by the set of equations in Figure 10, that clearly contains a cyclic dependency between magnetix fluxes (for sake of readability, all constants and negative signs are removed, the original equations can be found in [27]). It is clear that the cyclic dependencies can not simply broken with a delay, as they are intrinsic of the model: the only solution to implement such model in SystemC-AMS was thus to *solve symbolically* the set of equations with a symbolic solver capable of dealing with systems of linear equations [37].

However, the adoption of a symbolic solver is far from trivial for a non experienced user. The possible solutions to cover dynamic systems with SystemC-AMS in a more organic way are two: the extension to other models of computation (like the bond graph and the block diagram formalisms [3], that do support algebraic loops but have never been formalized in SystemC-AMS), or an extension of the solver to manage algebraic loops, as done by competitor tools. Such solutions would ease the modeling of dynamic systems, making the presence of unbreakable algebraic loops transparent to the user.

5.2 Solver issues

The current implementation of the SystemC-AMS DAE solver is based on a combination of implicit backward Euler and trapezoidal methods, extended with optimization techniques (e.g., Lower-Upper decomposition and Woodbury formulas) [24], [25]. The choice of such light-weighted methods determines a good compromise between accuracy and simulation speed, when compared with simulators supported by more complex solvers (e.g., HotSpot, based on 4-th order Runge-Kutta [11], and Simulink, supported by a library of solvers of increasing complexity [14]). However, this choice has severe limitations when modeling physical and dynamic systems: such systems are often stiff and marginally stable, and thus become numerically unstable for simpler numerical methods (unless the step size is extremely small, e.g., $1ms$ in case of the wind turbine) [24].

A clear example of this issue is the crane proposed in [24], whose position is tracked in Figure 11: the authors clearly prove that the SystemC-AMS solver (dotted) never reaches stability, while the adoption of a higher order numerical solver, like 4-th order Runge-Kutta (dashed), allows to quickly stabilize the system to the setpoint (solid), even if no code modification has been applied.

These examples highlight that, when moving to physical and dynamic systems, it becomes necessary to extend SystemC-AMS with additional higher-order solvers. Such solvers should be transparent to the user, i.e., the user should be able to seamlessly choose the solver at compilation time, to both cover a wider range of models and to provide different accuracy-simulation speed trade offs.

5.3 MoC related issues

Some issues are intrinsic of each MoC, even if they are tightly connected to characteristics of the scheduling management or of the solver adopted by SystemC-AMS. For this reason, they are treated separately in this section.

5.3.1 *TDF issues*

The Annex 2 of the SystemC-AMS standard extended TDF with the possibility of *varying the time step at run time*. This feature is extremely useful when modeling control systems (that constitute a relevant portion of production equipment), as it allows to modulate the granularity of simulation and to avoid over- or under-sampling. However, the driver function adjusting dynamically the time step must be written by the designer [38]. This might be very challenging, as detecting the correct time step requires detailed knowledge of time evolution and of the case study of interest. On one hand, changing the time step too often impacts on simulation time, as it requires to recompute the schedule of clusters and to reinitialize the underlying solver matrices. On the other, leaving the time step unchanged may lead to signal oversampling, thus wasting computation time without generating meaningful data for the simulation, or undersampling, with the effect of missing potential crucial events. Additionally, note that any modification of the time

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

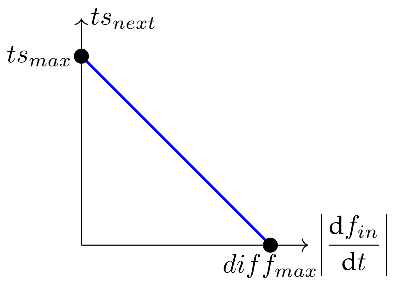


Figure 12: Graphical representation of the linear equation proposed in [38] to compute the time step ts_{next}: the x coordinate corresponds to the difference value of the input signal with respect to the time $\frac{df_{IN}}{dt}$ (maximum difference in amplitude $diff_{max}$), the y coordinate corresponds to the time step over time (maximum time step ts_max).

```
1  void change_attributes() {
2    double dy = in_val - in_prev;
3    double dx = t_val - t_prev;
4    double ts_next;
5    double diff_max = amp_max / ts_max;
6    if (dx == 0.0)
7      ts_next = ts_min;
8    else
9      ts_next = ts_max * (1 - (fabs(dy / dx) / diff_max));
10   // Check that ts_next is inside the boundaries.
11   ts_next = std::min(ts_max, std::max(ts_min, ts_next));
12   // Set the maximum sampling period between two
13   // consecutive samples.
14   set_max_timestep(ts_next, sc_core::SC_SEC);
15 }
16 void processing() {
17   // Reading and writing port values ...
18   change_attributes();
19 }
```

Listing 1: Time step controller implementation as proposed in [38]. The values of $diff\_max$, ts_max and ts_min and the linear relationship in line 9 stricly depend on the system being modeled.

step introduces a non negligible overhead, due to the necessary re-initialization (or refactorization) of the underlying solver matrices [25].

The work presented in [38] provides a possible implementation of the time step controller, which computes the new maximum time step so that if the change of the input signal with respect to time $\frac{df_{IN}}{dt}$ is increasing, the time step is decreased to ensure that all the variations of the signal will be sufficiently sampled. This behavior corresponds to a linear equation used to estimate the time step over time, as depicted in Figure 12. The corresponding code is shown in Listing 1. The critical aspects lie in the values chosen to define the relationship between time step and value difference over time (i.e., the angular coefficient of the linear equation), and the maximum allowed thresholds for time and value variations allowed in the model. All such values strictly depend on the system being modeled, and thus can not be generalized.

An example of a possible positive impact of a sound implementation of dynamic time step is *threshold crossing detection*, i.e., the capability of detecting the exact time stamp at which an input signal crosses a threshold value [38]. Note that this feature is crucial to support non-linear and piecewise linear constructs, and is supported by all other AMS languages (i.e., Verilog AMS and VHDL-AMS). Threshold crossing can be implemented with a careful management of the dynamic time step feature: when the input signal approaches the threshold, the time step must become smaller, so to determine much more exactly the crossing instant (e.g., in [38] the time step varies between $1.3ms$ and $0.1ms$ to accurately detect threshold crossing of a simple sinusoidal signal). Leaving such a crucial feature in the hands of the user is dangerous, and the effectiveness of SystemC-AMS would improve by embedding time step adjustment in the synchronization layer.

5.3.2 LSF issues

The LSF MoC has two main issues that become particularly critical in presence of physical and mechanical components. The first problem is tied to the initialization of the integrative constructs (i.e., the `sca_lsf::sca_integ` primitive). Integration converges over time only if the correct initial condition is set, thus requiring an effort from the designer to determine the correct value (default is 0). From Figure 7 it is possible to see that this may require up to ten digit precision: any minor modification (e.g., stopping at the ninth digit) would cause a divergence and stuck the integrator construct at `NaN`, with very little tracing support. This issue happens also in other simulation environments (e.g., Simulink), as it is intrinsic of the integrative problem. However, other simulators provide solvers for initial value problems of ordinary differential equations, that suggest a correct initial condition with no effort from the designer. Additionally, the fixed time step strategy of SystemC-AMS and the adoption of simple solvers to numerically solve the integral make the problem even more critical, and leave far less debugging information to the designer.

This problem could be partially diminished by *dynamically adjusting the time step* during the simulation, so that finer ticks could help detecting when the divergence occurs. If the change in the monitored signal with respect to time is increasing, then the next time step should be of decreasing size to ensure that all the variations of the signal will be sufficiently sampled. While, when the rate of change of the signal is lower, the simulation time step could be increased, resulting in a lower number of simulation points that should be computed. This can lead to very effective results in terms of simulation speed and accuracy, as it allows avoiding oversampling and in some cases also divergence. However, as mentioned in Section 5.3.1, the management of the dynamic time step is currently left entirely to the designer, thus resulting extremely challenging, and a very detailed knowledge of the case study is required.

5.3.3 ELN issues

The primitives defined for the ELN MoC suffer from severe limitations, that make its adoption in realistic industrial case studies complex.

The current implementation of ELN does not support *non-linear components*, e.g., diodes and transistors. To overcome this limitations, different solutions have been proposed in the literature to describe non-supported models, like diodes and ideal switches [39], [40], [41]. However,

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

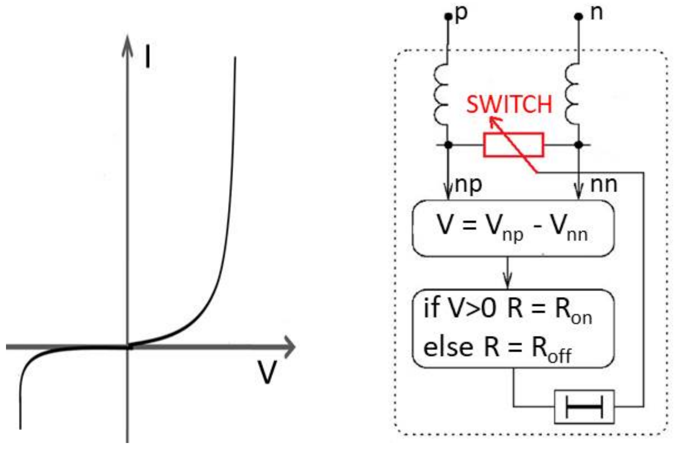


Figure 13: Diode characteristic curve (left) and SystemC-AMS implementation proposed by [40] (right).

these solutions are custom implementations, that in some cases modify critical parts of the SystemC-AMS infrastructure, like the construction of the underlying system matrices or the adoption of multiple solvers. As such, these solutions are subject to errors and to inefficient computation. A clear discussion of the problem is provided by [40], that presents the modeling of an energy harvester, including a diode. The behaviour of the diode is briefly explained in Section 3.4 and its general structure is depicted in Figure 5. The paper of Caluwaerts et al. [40] explores different implementations of the diode, and identifies the one based on a controlled switch as the most promising one. Figure 13 shows more in details how the switch is implemented inside an instance of a TDF component, through a TDF-controlled `sca_rswitch`. However, the authors highlight that the instantaneous switching provided by the SystemC-AMS primitive disturbs the state vector of the circuit. To solve this issue, they had to introduce an artificial capacitor, to maintain voltage evolution within reasonable levels. This proves that extending ELN support to non-linear (or at least piecewise-linear) elements would be crucial to enhance the effectiveness of SystemC-AMS, and that it can not be simply left to the intuition of the users.

Another major limitation of the ELN level is that it supports the Direct Current domain, while it does not support the *Alternate Current (AC) domain*, that is nonetheless crucial for modeling large scale production and energy systems (that may even operate in tri-phase). Sinusoidal curves are indeed supported only as sources of non-conservative data (e.g., the `sin_src` LSF block), thus not applying the conservation laws and the typical AC behaviors, e.g., the phase that may occur between current and voltage curves (that determines a possibly significant waste of power). To overcome this limitation, the authors of [14] introduced two possible ways to model AC curves in SystemC-AMS. Once again, this is little more than a custom solution, that should be engineered and introduced in the SystemC-AMS environment to ensure correct and efficient computation.

5.4 Implementation issues

Two issues linked to system management and scheduling are not caused by the standard per se, but rather by an inefficient memory usage in the Proof of Concept implementation of SystemC-AMS.

5.4.1 *Construction of static schedules*

As mentioned earlier in the paper, the SystemC-AMS scheduler heavily relies on the construction of static schedules, that are saved as activation lists reflecting the static order of execution of each component. Such an activation list has a maximum size, that can not be exceeded without incurring in memory blocking errors (e.g., end of stack size). If system size is limited, and modules work at the same time step (or with few variations), everything works fine, and actually this implementation of the scheduler is one of the reasons for the fast simulation of SystemC-AMS.

However, in some scenarios, it is necessary to integrate modules working at very different time scales: a valid example is the work in [13], that runs TDF modules with time steps that vary from *100ns* to *1s*, reflecting the dynamic properties of the different system components (i.e., battery chemical reactions are far slower than a processor clock rate), thus implying a difference of 7 orders of magnitude.

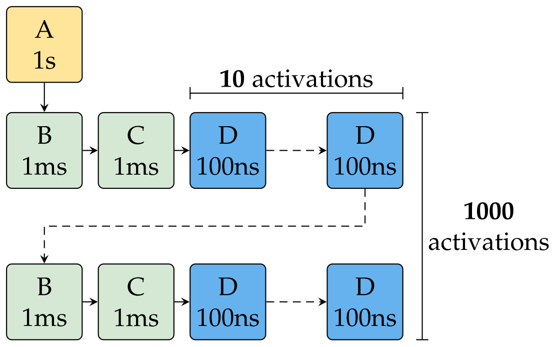


Figure 14: Example of scheduling issue: if the modules in the system execute at heterogeneous time rates, the activation list constructed by the TDF scheduler may be very long, despite of its periodicity. A more compact representation, making explicit the relationship between time steps, would impact on run time but allow the simulation of larger systems.

The top of Figure 14 abstracts the problem by showing the corresponding activation list: cluster schedule would contain one module working at *1s* (A), followed by 1000 entries to the modules working at *1ms* (B and C), each interleaved with the 10 entries to the module working at *100ns* (D). This quickly leads to a crash of the synchronization layer. Nonetheless, the decoupling at different time steps is a key advantage of SystemC-AMS, and it would be even more relevant in the context of plant or machinery modeling, as different system components evolve at very different speeds (e.g., fine grain mechanical models versus gross grain models of the energy subsystem or of thermal evolution). It might thus be necessary to re-think the SystemC-AMS scheduler, e.g., by allowing a dynamic reconstruction of the static schedule from activation queues

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

of smaller periods (e.g., a list of all modules that must be activated every 1ms, points to an entry summing up the 10 activation of the modules working at 100ns, as shown on the bottom of Figure 14). This solution would increase scheduling overhead at run time, but it would allow the simulation of larger systems with more realistic features.

5.4.2 Primitive instantiation at initialization time

Another issue ascribable to the current implementation of SystemC-AMS occurs at initialization time, during the construction of the matrices used by the solver to determine system state over time. The *initialization of system matrices* makes an intensive use of recursive calls over LSF and ELN primitives (e.g., to initialize system matrices), thus making computation unfeasible for large systems, due to stack overflows. To overcome this issue, artificially increasing stack size is only a palliative. The impact is heavy on descriptions that require a large number of primitives, as in the case of the thermal simulator proposed in [11], where the maximum achieved granularity of the thermal model is determined by the memory limitations of the SystemC-AMS solver for the instantiation of resistor and capacitor ELN primitives, and not by flaws or intrinsic limitations of the model itself. Thus, the future versions of the SystemC-AMS library should bear in mind memory usage as a constraint, and re-design synchronization algorithms to avoid extensive use of recursive calls.

5.5 Further extensions

The last two issues are rather nice-to-have features, that would ease the job of the designer and enhance SystemC-AMS simulation.

5.5.1 Multi-discipline systems

Section 4.3 clarified that the only conservative domain supported by SystemC-AMS is the electrical one: any other kind of conservative description must be either reduced to the electrical domain (e.g., through a circuit equivalent model, or by relying on the analogies between domains), or implemented as equations that explicit also the intrinsic conservation laws of the domain of interest.

Nonetheless, it is easy to notice that the majority of designs cited in Section 4 contains components belonging to different disciplines: the wind turbine and the DC motor include mechanical equations, while the pico-projector uses rotational and rotational acceleration constructs to allow the evaluation of its control SW removing the ripple effect. Let us now focus on the *mirror* component of the pico-projector. It is an excellent example of multi-discipline component since it was originally implemented in Verilog-AMS by combining together the `electrical` and the `rotational_omega` disciplines. Specifically, *rotational_omega* uses the angular acceleration as potential nature, and the angular force as flow nature. Internally, the mirror is divided into two stages: the first stage is electrical and receives the signals from the two OpAmps components; then, the electrical part is connected to the electro-mechanical one, implemented by using the two rotational disciplines.

A viable solution for extending SystemC-AMS to non-electrical domains components could be the adoption of the *bond graph* MoC [42]. A bond graph represents the energy flow between generalized elements, defined by their effort (voltage for the electrical domain) and flow (current) [43]. The physical quantity represented by effort and flow, and the corresponding unit measure, strictly depend on the modeled domain, as outlined in Table 2[2]. Bonds between elements define causality constraints, and allow to compute system state by propagating equations and relations between quantities. The bond graph MoC is adopted by a number of simulators, including Simulink and Modelica, thanks to its effective support for multiple domains; additionally, the analysis of the bonds allows to easily detect model flaws, e.g., algebraic loops or ill-posed models.

Table 2: Bond graph effort and flow elements for different domains (with unit measure)

DOMAIN	EFFORT	FLOW
ELECTROMAGNETIC	Current (A)	Voltage (V)
MECHANICAL LINEAR	Velocity (m/s)	Force (N)
MECH. ANGULAR	Angular velocity (rad/s)	Torque $(N \cdot m)$
HYDRAULIC	Volume flow rate (m^3/s)	Pressure (Pa)

A preliminary implementation of bond graphs in SystemC-AMS has been proposed over a decade ago, together with an analysis of how the SystemC-AMS simulation infrastructure should be extended to include a bond graph solver [42]. The increasing relevance of non-electrical conservative domains for plant modeling makes bond graph an even more crucial solution: thus, its definition could contribute to an effective extension of SystemC-AMS in the direction of modeling physical and dynamic systems.

5.5.2 SystemC-AMS parallelization

Many works in the literature proposed approaches to parallelize SystemC by exploiting the intrinsic parallelism of RTL, and at times also of TLM, to achieve faster simulation through the adoption of parallel schedulers or by moving event-driveness to separate threads [44]. The work in [45] highlighted a common limitation of all such approaches: their integration inside of the SystemC kernel would be impossible, as they do not follow its semantics and APIs, or even completely remove the kernel for the sake of faster simulation.

The structure of SystemC-AMS offers additional ways to achieve parallelization, without distorting the building blocks of SystemC. At this point, it is clear that SystemC-AMS simulation relies on system partitioning into clusters of highly interconnected modules and primitives. Such clusters have few dependencies on each other, and their producer-consumer dependencies are clearly defined by the TDF semantics, that does not allow contention (e.g., multiple writers). Thus, it would be possible to foresee a

2. To effectively support multiple domains, the elements are extended to generalized displacement (charge for the electrical domain), generalized momentum (flux), resistance (resistance), inertance (inductance) and compliance (capacitance).

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

parallel extension of the SystemC kernel, where clusters are executed independently, reducing synchronization only at the borders. Despite of being a promising development of the standard, this direction has not been explored yet.

6 Conclusion

The paper started from the analysis of the current state of the art and from a set of heterogeneous case studies to make a point on the current status of SystemC-AMS with respect to the modeling of CPPSs. The analysis highlighted issues with different levels of severity: some have to be considered to allow an effective modeling of physical and mechanical behaviors (e.g., solver extension, support for dynamic systems and non-linear behaviors), while others are nice-to-have, that would ease the modeling from the perspective of a non-expert user or extend the expressiveness of the language (e.g., multi-domain support, improved memory usage). The discussion in the paper provided pointers to current custom solutions and possible future developments and improvements. The issues discussed in this work can help prioritizing which issues should be addressed first for improving SystemC-AMS, to make it an effective resource also in the Industry 4.0 automation scenario.

References

[1] K. Kang, S. Park, B. Bae, J. Choi, S. Lee, B. Lee, and J.-B. Lee, "Seamless SoC Verification Using Virtual Platforms: An Industrial Case Study," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, mar 2019, pp. 1204–1205.

[2] S. Vinco, V. Guarnieri, and F. Fummi, "Code manipulation for virtual platform integration," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2694–2708, sep 2015.

[3] F. Cenni, O. Guillaume, M. Diaz-Nava, and T. Maehne, "SystemC-AMS/MDVP-based modeling for the virtual prototyping of MEMS applications," in *2015 Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*. IEEE, apr 2015, pp. 1–6.

[4] S. Vinco, M. Lora, and M. Zwolinski, "Conservative behavioural modelling in systemc-AMS," in *Proceedings of 2015 IEEE Forum on Specification & Design Languages FDL*. IEEE, 2015, pp. 1–8.

[5] V. Fernandez, E. Wilpert, H. Isidoro, C. B. Aoun, and F. Pêcheux, "SystemC-MDVP modelling of pressure driven microfluidic systems," in *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*. IEEE, jun 2014, pp. 10–13.

[6] E. Markert, M. Dienel, G. Herrmann, and U. Heinkel, "SystemC-AMS assisted design of an inertial navigation system," *IEEE Sensors Journal*, vol. 7, no. 5, pp. 770–777, may 2007.

[7] B. Vernay, A. Krust, G. Schropfer, F. Pêcheux, and M.-M. Louerat, "SystemC-AMS simulation of a biaxial accelerometer based on MEMS model order reduction," in *2015 Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*. IEEE, apr 2015, pp. 1–6.

[8] X. Pan, C. Zivkovic, and C. Grimm, "Virtual prototyping of heterogeneous automotive applications: Matlab, SystemC, or both?" in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, jan 2019, pp. 544–549.

[9] B. Fernandez-Mesa, L. Andrade, and F. Perrot, "Electronic system level design of heterogeneous systems: a motor speed control system case study," in *2019 17th IEEE International New Circuits and Systems Conference (NEWCAS)*. IEEE, jun 2019, pp. 1–4.

[10] S. Vinco, A. Sassone, F. Fummi, E. Macii, and M. Poncino, "An open-source framework for formal specification and simulation of electrical energy systems," in *Proceedings of the 2014 international symposium on Low power electronics and design - ISLPED '14*. ACM Press, 2014, pp. 287–290.

[11] Y. Chen, S. Vinco, E. Macii, and M. Poncino, "SystemC-AMS thermal modeling for the co-simulation of functional and extra-functional properties," *ACM Transactions on Design Automation of Electronic Systems*, vol. 24, no. 1, pp. 1–26, jan 2018.

[12] V. Tran, P. Tisserand, F. Pêcheux, and A. Pinna, "Towards the simulatable specification of a highly customisable SystemC AMS alternator model in its multi-domain environment," in *2016 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE, jun 2016, pp. 1–4.

[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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1702–1715, oct 2017.

[14] Y. Chen, S. Vinco, D. J. Pagliari, P. Montuschi, E. Macii, and M. Poncino, "Modeling and simulation of cyber-physical electrical energy systems with SystemC-AMS," *IEEE Transactions on Sustainable Computing*, 2020.

[15] M. Radpour and S. M. Sayedi, "A SystemC model of energy harvesting CMOS digital pixel sensor," in *2016 4th International Conference on Robotics and Mechatronics (ICROM)*. IEEE, oct 2016, pp. 192–195.

[16] F. Cenni, E. Simeu, and S. Mir, "Macro-modeling of analog blocks for SystemC-AMS simulation: A chemical sensor case-study," in *2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, oct 2009, pp. 211–214.

[17] F. Pêcheux, M. Madec, and C. Lallement, "Is SystemC-AMS an appropriate "promoter" for the modeling and simulation of bio-compatible systems?" in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, may 2010, pp. 1791–1794.

[18] B. Vogel-Heuser, S. Wildermann, and J. Teich, "Towards the co-evolution of industrial products and its production systems by combining models from development and hardware/software deployment in cyber-physical systems," *Production Engineering*, vol. 11, no. 6, pp. 687–694, oct 2017.

[19] S. J. Oks, M. Jalowski, A. Fritzsche, and K. M. Möslein, "Cyber-physical modeling and simulation: A reference architecture for designing demonstrators for industrial cyber-physical systems," *Procedia CIRP*, vol. 84, pp. 257–264, 2019.

[20] M. Barnasconi, "SystemC and digital twin: good match or not?" SystemC Evolution Day, https://www.accellera.org/news/events/systemc-evolution-day-2019, 2019.

[21] L. Monostori, "Cyber-physical production systems: Roots, expectations and r&d challenges," *Procedia CIRP*, vol. 17, pp. 9–13, 2014.

[22] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche, "Design, modelling, simulation and integration of cyber physical systems: Methods and applications," *Computers in Industry*, vol. 82, pp. 273–289, 2016.

[23] H. Posadas, J. A. Adamez, E. Villar, F. Blasco, and F. Escuder, "RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model," *Design Automation for Embedded Systems*, vol. 10, no. 4, pp. 209–227, dec 2005.

[24] P. A. Hartmann, P. Reinkemeier, A. Rettberg, and W. Nebel, "Modelling control systems in SystemC AMS - benefits and limitations," in *2009 IEEE International SOC Conference (SOCC)*. IEEE, sep 2009, pp. 263–266.

[25] C. Reuther and K. Einwich, "A SystemC AMS extension for controlled modules and dynamic step sizes," in *Proceedings of 2012 IEEE Forum on Specification & Design Languages FDL*, Sep. 2012, pp. 90–97.

[26] COSEDA Technologies, "COSIDE - the design environment for heterogeneous systems," https://www.coseda-tech.com, 2020.

[27] J. Martinez, "Modelling and control of wind turbines," *Imperial College London*, 2007.

[28] K. Caluwaerts and D. Galayko, *Heterogeneous and Non-linear Modeling in SystemC-AMS*. Springer Netherlands, 2009, pp. 113–128.

[29] R. Narayanan, N. Abbasi, M. Zaki, G. Al Sammane, and S. Tahar, "On the simulation performance of contemporary ams hardware description languages," in *2008 International Conference on Microelectronics*, 2008, pp. 361–364.

[30] T. Machne, Z. Wang, B. Vernay, L. Andrade, C. B. Aoun, J.-P. Chaput, M.-M. Louerat, F. Pêcheux, A. Krust, G. Schropfer, M. Barnasconi, K. Einwich, F. Cenni, and O. Guillaume, "UVM-SystemC-AMS based framework for the correct by construction design of MEMS in their real heterogeneous application context," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, dec 2014, pp. 862–865.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3226567

[31] C. Zhao and T. J. Kazmierski, "An extension to SystemC-A to support mixed-technology systems with distributed components," in *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, mar 2011, pp. 1–6.

[32] Y. Chen, E. Macii, and M. Poncino, "A circuit-equivalent battery model accounting for the dependency on load frequency," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, mar 2017, pp. 1177–1182.

[33] A. Vachoux, C. Grimm, and K. Einwich, "Extending SystemC to support mixed discrete-continuous system modeling and simulation," in *2005 IEEE International Symposium on Circuits and Systems*. IEEE, 2005.

[34] L. Andrade, T. Maehne, A. Vachoux, C. Ben Aoun, F. Pêcheux, and M. Louerat, "Pre-simulation symbolic analysis of synchronization issues between discrete event and timed data flow models of computation," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE Conference Publications, March 2015, pp. 1671–1676.

[35] D. Genius, R. Porto, L. Apvrille, and F. Pêcheux, "A tool for high-level modeling of analog/mixed signal embedded systems," in *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS - Science and Technology Publications, 2019, pp. 435–442.

[36] The MathWorks, Inc., "Algebraic loop concepts," https://it.mathworks.com/help/simulink/ug/algebraic-loops.html, 2020.

[37] C. Bauer, A. Frink, and R. Kreckel, "Introduction to the GiNaC framework for symbolic computation within the C++ programming language," *Journal of Symbolic Computation*, vol. 33, no. 1, pp. 1–12, 2002.

[38] L. L. Andrade Porras, T. Maehne, M.-M. Louërat, and F. Pêcheux, "Time Step Control and Threshold Crossing Detection in SystemC AMS 2.0," in *Huitième colloque du GDR SOC-SIP du CNRS*, Lyon, France, Jun. 2013, p. 3.

[39] F. Cenni, S. Scotti, and E. Simeu, "SystemC AMS behavioral modeling of a CMOS video sensor," in *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*. IEEE, oct 2011, pp. 380–385.

[40] K. Caluwaerts, D. Galayko, and P. Basset, "SystemC-AMS heterogeneous modeling of a capacitive harvester of vibration energy," in *2008 IEEE International Behavioral Modeling and Simulation Workshop*. IEEE, sep 2008, pp. 142–147.

[41] L. Gil and M. Radetzki, "SystemC AMS power electronic modeling with ideal instantaneous switches," in *Proceedings of 2014 IEEE Forum on Specification & Design Languages FDL*, vol. 978-2-9530504-9-3. IEEE, oct 2014, pp. 1–8.

[42] T. Maehne, A. Vachoux, and Y. Leblebici, "Development of a bond graph based model of computation for SystemC-AMS," in *2008 Ph.D. Research in Microelectronics and Electronics*. IEEE, jun 2008, pp. 77–80.

[43] S. Bliudze, S. Furic, J. Sifakis, and A. Viel, "Rigorous design of cyber-physical systems," *Software & Systems Modeling*, vol. 18, no. 3, pp. 1613–1636, dec 2017.

[44] B. Haetzer and M. Radetzki, "A comparison of parallel systemc simulation approaches at RTL," in *Proceedings of 2014 IEEE Forum on Specification & Design Languages FDL*. IEEE, oct 2014.

[45] R. Domer, "Seven obstacles in the way of standard-compliant parallel SystemC simulation," *IEEE Embedded Systems Letters*, vol. 8, no. 4, pp. 81–84, dec 2016.

Enrico Fraccaroli is a postdoctoral research fellow at the Department of Computer Science of the University of Verona since May 2019. He received his Ph.D. degree in computer science from the University of Verona, Italy, in May 2019. His research interests are the development of new methodologies for the efficient simulation and functional safety evaluation of embedded platforms composed of analog, digital and network components.

Sara Vinco is Assistant Professor with tenure track at the Department of Control and Computer Engineering of Politecnico di Torino (Italy) since 2017. She received the Ph.D. degree in computer science from the University of Verona (Italy) in 2013. Her current research interests include energy efficient electronic design automation and techniques for the simulation and validation of cyber-physical production systems.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Politecnico di Torino. Downloaded on December 12,2022 at 10:29:54 UTC from IEEE Xplore. Restrictions apply.