Handling causality and schedulability when designing and prototyping cyber-physical systems

Rodrigo Cortés porto, Daniela Genius, Ludovic Apvrille

Noname manuscript No.
(will be inserted by the editor)

Handling Causality and Schedulability when Designing and Prototyping Cyber-Physical Systems

Rodrigo Cortés Porto · Daniela Genius · Ludovic Apvrille

Received: date / Accepted: date

Abstract Cyber Physical Systems (CPS) are built upon digital and analog circuits, making it necessary to handle different Models of Computation (MoC) during their design and verification (e.g. by simulation). When designing these systems, an important aspect to consider is the causality between the different domains. For this, we introduce a new model-driven framework able to identify causality problems and to suggest a valid schedule between the analog and digital domains. Once a valid schedule has been computed, our framework can generate cycle & bit accurate virtual prototypes (in SystemC/SystemC AMS) from high-level SysML models.

Keywords Cyber-physical systems, Virtual prototyping, Co-simulation

1 Introduction

Model-driven techniques rely on high level models in order to assist engineers in designing embedded systems combining software and hardware aspects. Model transformations are now a common practice to generate verification or executable code from models at different levels of abstraction. Nonetheless, in most cases, these approaches are limited to the digital parts of systems, i.e.

R. Cortés Porto
LIP6, Paris Sorbonne University, Paris, France and Technical University of Kaiserslautern, Department of Electrical Engineering and Information Technology, Kaiserslautern, Germany

Daniela Genius
LIP6, Paris Sorbonne University, Paris, France

Ludovic Apvrille
LTCI, Télécom Paris, Institut Polytechnique de Paris, Sophia-Antipolis, France

to the design of embedded software and the hardware platform. Yet, many embedded systems contain sensors and actuators, such as in robotics, medical and automotive: the design of the whole system commonly relies on a combination of models and simulation techniques from different domains.

Our former work [20] explained how SysML models can be used to capture both digital and analog aspects, to generate analog/digital virtual prototypes, and to simulate these systems [14] using the TTool framework.

The present work focuses on algorithms that can identify valid schedules —if they exist— between analog and digital domains. While causality issues are usually detected only at simulation or execution time, we propose a way to statically detect them directly in high-level SysML models, thus before any simulation or executable code is generated. As a result, the design of such mixed systems (software, hardware, analog) is facilitated. Once defined, schedules are enforced by the used of delays, as explained in the paper.

Obviously, once a valid schedule has been identified, the designer is expected to use other verification techniques, such as simulation and model-checking —that are also integrated in TTool— to verify that the system still respects all other requirements, e.g. real-time constraints such as deadlines and real-time schedulability.

The next Section shows how similar work addresses schedulability and causality issues between domains. Then, Section 3 presents the bases of our contribution to address synchronization aspects in our framework. Section 4 and 5 detail our contribution and explain our algorithms. Section 6 compares our approach with the two closest related works, while Section 7 concludes the paper and draws perspectives.

2 Related Work

Embedded systems can be modeled in different Models of Computation (MoC), with different notions of concurrency and time [25]. Note that a Model of Computation only precises the properties of model execution, not one execution in particular.

Our approach combines two operational semantics. For the digital part of the system, we use a Discrete Event (DE) MoC, which models the operation of a system as a discrete sequence of events in time, changing the state of the system. For the analog part, we use a Dataflow model, detailed in Section 3.3.

Well established tools in analog/mixed signal design, like *Ptolemy II* [27] are based upon a Dataflow model. They address heterogeneous systems by defining several sub domains. Instantiating elements controlling the time synchronization between domains is left to the responsibility of designers.

Metro II [16] allows *Adaptors* for data synchronization as a bridge between the semantics of components belonging to different MoCs. The model designer still has to implement time synchronization. As a common simulation kernel handles all process execution, MoCs are not well separated.

Modelica [19] is an object-oriented modeling language for component-oriented systems containing e.g. mechanical, electrical, electronic and hydraulic components. Classes contain a set of equations that can be translated into objects running on a simulation engine. Yet, since time synchronization is not predefined, the simulation engine must manipulate objects in a symbolic way in order to determine an execution order between components of different MoCs. Linking simulation with different MoCs can be done by using e.g. the Functional Mockup Interface [10]. Yet, in our case, we need to take into account both DE semantics and a semantics for the analog part from the beginning, in the high-level models, because our aim is to compute a valid schedule **before** simulation.

Discrete Event System Specification (DEVS [12]) is a modular and hierarchical formalism which supports discrete events and continuous systems. Continuous functions can be described by differential equations. A dozen of platform implementations based on DEVS exist, ranging from Petri Net based over object oriented to Python based. DEVS is also used for timing model transformations [36]. However, it relies on a global homogeneous time in all parts of the model.

Zelus [7] is a synchronous language [8] like Lustre and Lucid synchrone, but is specifically conceived for hybrid systems. It combines Ordinary Differential Equations to express continuous time with synchronous data-flow equations. Zelus has an elaborate type system and an execution model that alternates between continuous phases and sequences of "run-to-completion" discrete actions. Instantaneous feedback loops are statically rejected. Zelus neither proposes a graphical interface nor the generation of a virtual prototype.

MARTE [17], a very popular UML profile for modeling embedded real-time systems, can model jitter and polymorphic time through its Time subprofile.

CCSL [31] is a language for specifying constraints on clocks. It relies on a model of time similar to the one used by MARTE, but clocks have no associ-

ated time scale: CCSL is exclusively event-based, but cannot specify an event occurring at an arbitrary time or with an arbitrary delay.

TESL [11] is inspired by CCSL [31]. It reconciles time scales in several models of a heterogeneous nature. TESL models causality between the occurrences of events, time advancing on different scales, and supports both event-driven and time-driven specifications. Like our approach, it uses timestamps. Like Zelus, TESL reasons with multiple clocks and takes into account periodicity and continuous values. TESL uses timed finite state machines and fixed-point iteration; its deterministic nature makes it less well adapted for specification on a high level of abstraction.

On the other hand, approaches which aim at full-system simulation combine a hardware model, with software mapped onto it and an operating system. UML/SysML based modeling techniques such as MARTE have been employed to model cyber-physical systems [34], but are scarcely used for low-level simulation. With very few exceptions such as [37], modeling tools based on the MARTE approach do not support full-system simulation.

In the simulation domain, many approaches are based on SystemC [24], with or without alteration of the simulation kernel, which was initially targeted only toward discrete systems simulation. Among the frameworks based on SystemC are HetSC [23], HetMoC [40] and ForSyDe [32], all having the disadvantage that instantiation of elements and synchronization control is totally left to designers.

SystemC-H [33] and SystemC-A [39] extend the SystemC simulation kernel. The former allows only one hierarchical level in the description of models. Execution of models is based on a master-slave relation: a modified DE kernel initializes and simulates the processes described by means of different MoC-specific kernels. The SystemC scheduler preserves the initialization, evaluate, update, delta and time notification phases, but some of them are modified. Synchronization mechanisms are not available among components defined under different models of computation. In SystemC-A, which allows hierarchy, the scheduler calls the analog kernel phases (iteration and verification) before SystemC scheduling. The independent analog kernel is able to synchronize with the DE simulation kernel.

SystemC AMS extensions [1] is a standard describing an extension of SystemC with AMS (Analog/Mixed Signal) and RF (Radio Frequency) features [38] and predefines several Models of Computation. In the scope of the project BeyondDreams [9], a mixed analog-digital systems proof-of-concept simula-

tor has been developed [18], based on the SystemC AMS extension standard. Unfortunately, causality issues may invoke errors at simulation time - the simulation is aborted with an error message.

Another simulator, Multi Domain Virtual Prototyping (MDVP) is proposed in the H-Inception project[22]. A major result of that work is that causality issues can be automatically checked *before* simulation [2]. The simulation phases of SystemC are not modified. MDVP comes with a full-system simulation based on SoCLib [35], a free library of hardware models written in SystemC, for the digital part and a proprietary SystemC AMS simulator for the analog part. However, scheduling and causality checking starts from SystemC AMS code, not from a SysML representation.

3 Context

The context of our work is twofold. First, a modeling and verification framework named TTool [4]. This tool supports both modeling and verification aspects, in particular the generation of a SystemC virtual prototype. Second, TTool can generate a SystemC AMS specification from the analog parts of models, as explained in this paper. Yet, as explained before, TTool is extended with the early detection of causality between domains. This results in proposing a schedule of operations between domains that can be applied to the SystemC AMS cluster. Finally, this section presents TTool and SystemC AMS which are at the root of our contribution.

3.1 TTool

TTool is a UML/SysML free and open-source software for model-based engineering and verification of embedded systems at different abstraction levels: functional, partitioning, software design and deployment [30]. From SysML diagrams, TTool can generate virtual prototypes for full-system simulation based on SoCLib. Until recently, only digital aspects could be considered in TTool. To each of the aforementioned abstraction levels is associated a dedicated view as shown in Figure 1.

The method follows these abstraction levels e.g. it explains how to take hardware/software partitioning decisions at a high level of abstraction and how to regularly (re)validate them during software development [21]. The blue

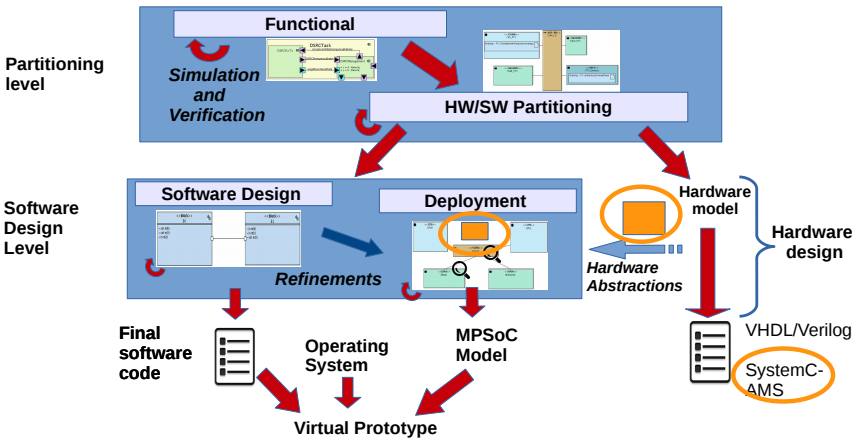

Fig. 1: Hardware/software partitioning and code generation for system-on-chip platforms

boxes represent the functional/partitioning and the software design/deployment levels in TTool.

Figure 1 shows, encircled in orange, the additions which were made in [20]. To the hardware model, we added the possibility to generate the SystemC AMS parts of the virtual prototype. They can be integrated in the deployment model (orange rectangles) along with the digital parts.

In both partitioning and deployment views, tasks must be allocated to processing elements (processors, hardware accelerators, ...), while communications between tasks are to be allocated to communication elements (e.g., buses and bridges) and to storage elements (e.g. memories).

An important advantage of TTool is that it offers an automated approach for formal verification and fast simulation. Formal verification is based on internal model-checkers, or external tools like UPPAAL [6].

TTool can also generate a SystemC specification from the deployment view. Such a specification can be simulated with a cycle & bit accurate simulator, using hardware models stemming from the *SoCLib* [35] public domain library targeting shared-memory *multiprocessor-on-chip system* architectures.

3.2 SystemC

SystemC is a system design modeling language based on C++, which adds libraries to address the modeling of both hardware and software systems.

3.2.1 Language features

SystemC includes important hardware oriented features such as a global discrete time model, concurrent execution of multiple processes supported by a cooperative multitasking model (scheduler), and hardware data types, supporting explicit bit widths for integer and fixed point quantities. SystemC supports hierarchy via modules. The communication and synchronization models are implemented by a set of mechanisms such as interfaces, ports and channels.

SystemC follows a block-oriented approach: it allows the representation of systems by a combination and interconnection of blocks and signals: blocks represent behaviors and can have multiple inputs and outputs; signals ensure the communication among blocks. Ports are connected to channels through interfaces, while processes describe the operation of the modules.

User-specific processes can be called by the Discrete-Event kernel during simulation. Two kinds of processes can be defined in SystemC: (1) methods, which are always executed from beginning to end; and (2) threads, which can suspend themselves during simulation using *wait* statements. These kinds of processes are also known as static processes because they are registered in the DE kernel before simulation.

3.2.2 Discrete Event Simulation Kernel

The simulation kernel provides the core features for the elaboration and simulation of models [24], relying on the *Discrete Event* (DE) MoC. Elaboration creates the data structures required to support the simulation semantics: it creates the module hierarchy, instantiates processes, bounds ports and channels, and sets the time resolution to be used (by default 1ps). Simulation runs the scheduler and deletes the data structures created during elaboration.

The scheduler controls the timing and the order for executing the processes. Execution is performed in five phases: (1) initialization, where all defined processes are entirely executed (methods) or until the first wait statement (threads); (2) evaluation, where each process ready to run is selected and its execution is resumed (this may cause new processes ready to run in the same phase); (3) update, where channels are updated thanks to the results of the evaluation phase; (4) delta notification, where are analyzed the notifications made during the previous two phases: if they should be executed in the current simulation time then, the evaluation phase is re-executed; (5) timed notifica-

tion, where the notifications are also evaluated: if they should not be executed in the current simulation time, such time is increased and then, the evaluation phase is re-executed. When no more notifications are present, execution is stopped.

3.3 SystemC Extensions for AMS

Timed Data Flow (TDF) is the (main) MoC of SystemC AMS for representing analog systems. TDF is based on the timeless Synchronous Data Flow (SDF) semantics [28]. SystemC AMS adds to SystemC support for signals where continuous data values are considered as signals and sampled with a constant time step. TDF maintains two important properties of the SDF formalism: the abilities to determine a static schedule, and a periodic execution.

A TDF *module* is described by an attribute representing a *time step* and a *processing function*. The time step is associated to a time period during which the processing function should be executed. The processing function corresponds to a mathematical function depending on both inputs and internal states. At each time step, a TDF module reads a fixed number of samples from each of its input ports, executes the processing function, and writes a fixed number of samples to each of its output ports.

TDF modules have the following attributes:

1. A Module time step (**Tm**) denotes the period during which the module is activated. A module is activated only if there are enough samples available at its input ports.
2. The Rate (**R**). Each module reads or writes a fixed number of data samples each time it is activated, annotated to the port as port rate.
3. Port time step (**Tp**) denotes the time interval between two operations (read or write).
4. A Delay (**D**) associated to a port is the **number of samples** which are added to this port at module initialization. A delay makes the port handle a fixed number of samples at each activation, and read or write them in the following activation of the port.

A *cluster* is a set of interconnected modules. Figure 2 shows a cluster in the System C AMS standard notation [1]. The DE modules X and Y are represented as white blocks, TDF modules A and B as gray blocks, TDF ports

as black squares, TDF converter ports as black and white squares, DE ports as white squares and signals as arrows.

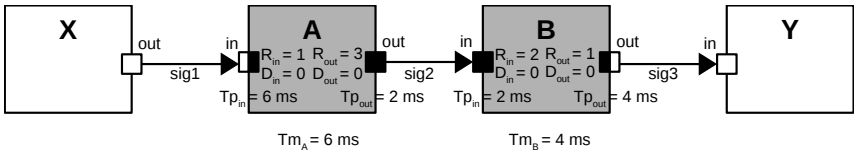

Fig. 2: TDF Cluster with two DE modules X and Y

The module time step of A is 6 ms, its input port time step is 6 ms, and port rate 1, respectively. Output port time step is 2 ms and rate 3. B has an input port time step of 2 ms and rate of 2, and module time step of 4ms, rate of 1 and time step of 4 ms in the output port, respectively. The input port of A and the output port of B are *converter ports* by which TDF modules interact with DE modules.

Since SystemC AMS is an extension of SystemC, and SoCLib is written in SystemC, the choice of extending TTool with SystemC AMS generation capabilities for integrating analog/mixed signal components was natural.

4 Detecting and solving schedulability issues

This section presents definitions and examples, before describing in detail each algorithm —the main contribution of the paper—, for determining (i) a valid schedule and (ii) detecting and resolving causality issues. We also show how the two are related and interdependent.

4.1 Schedulability definition

We now define what we mean *schedulability* and we give several examples to illustrate the schedulability concept.

Definition 1 Schedulability denotes the correct and static execution-order of TDF modules in one cluster, consistently with the data flow characteristics within a cluster. A cluster is **schedulable** if the module time step is consistent with the rate and time step of any port within a module.

The relation between time steps and rates is as follows. T_M denotes the module time step, T_{pi} and T_{po} respectively denote the input and output port time steps, R_{pi} and R_{po} respectively denote the input and output port rates:

$$T_M = T_{pi} \times R_{pi} = T_{po} \times R_{po}$$

Before the static schedule of a cluster can be computed, the *time steps* and *sampling rates* that are not indicated in the model need to be calculated.

It is not necessary to indicate all time steps and rates. One time step and one rate provided initially are sufficient. Then, the next time steps and rates are calculated by upstream and downstream propagation, as explained in [1]. The main point here is that the consistency of a cluster exclusively relies on the compatibility of port rate and delay values and is thus intrinsically independent of the initially selected time step.

4.2 Analysis

Since the TDF MoC is based on the SDF MoC, it computes the static schedule in a very similar way. Thus, TDF modules can be related to nodes of SDF graphs, while signals between ports of TDF modules can be related to arcs of SDF graphs. According to [29], the following elements can be used in order to compute a static schedule:

1. A topology matrix Γ, where the number of columns corresponds to the number of TDF modules, and where the number of rows correspond to the number of signals connecting the TDF modules. The entries of the matrix correspond to the rates of the ports of a module. For an an output port, the entry is positive. On the contrary, an input port has a negative rate value.

   Let us take an example with only TDF modules, and a feedback loop. Using the model depicted in Figure 3, the following topology matrix Γ is constructed. The first column corresponds to module **A** and the second column to module **B**. The first row represents the arc named **sig1** and the second row refers to **sig2**.

$$\Gamma = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

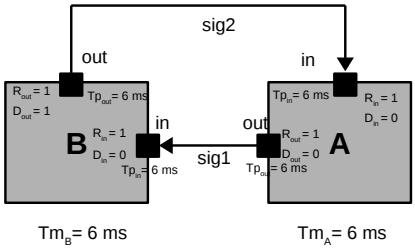

Fig. 3: TDF Cluster with feedback loop

2. A buffer vector $b(i)$ represents the data available in the arcs of the graph at time i. The size of the buffer is the number of arcs of the graph. Note that $b(0)$ represents the delays of the ports of the TDF modules since at time $i = 0$ the delay samples are available in the buffer (initial configuration). The following vector represents buffer $b(0)$ for the model of Figure 3. Once again, the first row corresponds to the delays generated for the arc named **sig1** and the second row for **sig2**. There is a delay of 1 in port **B.out**.

$$b(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

3. The column vector q represents the number of executions of a node. To computer these numbers, the Matrix Equation 1 needs to be solved for q: the smallest positive integer solution is selected.

$$\Gamma \cdot q = 0 \tag{1}$$

Using the model from Figure 3 and Equation 1, we have:

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} q_A \\ q_B \end{bmatrix} = 0$$

And solving for the smallest positive integer vector q, we have: $q_A = 1$ and $q_B = 1$, representing the number of executions for each module of the cluster.

Using the model from Figure 2, the following topology matrix Γ is constructed. It is formed by two columns representing the modules **A** and **B** and one row that represents the signal `sig2`.

$$\Gamma = \begin{bmatrix} 3 & -2 \end{bmatrix}$$

Since there are no delays, the buffer $b(0)$ is:

$$b(0) = \begin{bmatrix} 0 \end{bmatrix}$$

Using Equation 1, we have:

$$\begin{bmatrix} 3 & -2 \end{bmatrix} \cdot \begin{bmatrix} q_A \\ q_B \end{bmatrix} = 0$$

And solving for the smallest positive integer vector q: $q_A = 2$ $q_B = 3$. We thus have to schedule two instances of module A and three of B.

4.3 Algorithm for static scheduling

The sequential scheduling algorithm proposed in [29] is used to compute a static schedule of TDF clusters. Listing 1 shows an adaptation of this algorithm, which has been slightly modified in lines 12 and 13 to calculate the necessary delays that will solve scenarios where no schedule can be found. A node here represents a TDF module.

Listing 1 Sequential Scheduling algorithm (Adapted from [29]).

```
1:  procedure computeStaticSchedule
2:      Solve Matrix eq. for the smallest positive integer vector q
3:      Form an ordered list L of the nodes (α) of the model.
4:      for each node α do
5:          if α is runnable then
6:              Schedule α
7:          end if
8:      end for
9:      if each node α has been scheduled q_α times then
10:         STOP
11:     else if no node could be scheduled then
12:         Deadlock detected: calculate suggested delay to solve deadlock
13:         STOP_AND_RESTART
14:     else GO to 4
15:     end if
16: end procedure
```

The ComputeStaticSchedule procedure of this algorithm assumes that the topology matrix Γ is already built and the buffer vector $b(0)$ is already

filled with the delays of the ports at time 0. The first step of the algorithm shown in line 2 is to solve the Matrix Equation 1 for q and to provide the smallest positive integer solution for the equation. In line 3, an arbitrarily ordered list of the nodes of the TDF cluster is computed. Then, all runnable nodes are scheduled for execution as shown in lines 4 - 8. Note that in line 5, a node is considered runnable if it has not run q times and it will not cause buffer $b(i)$ to be negative. When a node runs, it consumes from the buffer the number of units that the rate of its ports specifies. In line 9, if each node has already been scheduled q times, then the algorithm stops. Otherwise, if no node could be scheduled during this cycle, in line 12 a deadlock is detected, which implies that a schedule could not be found because of feedback loops in the graph: delays have to be added in order to find a schedule.

Here, the suggested delay (D_{sug}) that will solve the deadlock can be calculated based on the last node α that tried to be scheduled and that made the buffer b_α to go negative. Equation 2 shows how D_{sug} is calculated. It uses the current delay D_{cur} of the port associated to the buffer and subtracts it from the value of buffer b_α. Indeed, since buffer b_α is negative, doing so will result in computing a new delay, which will make b_α to stop being negative.

$$D_{sug} = D_{cur} - b_\alpha \tag{2}$$

4.4 Example: inserting delays

Let us continue the example from section 4.2. Supposing that all delays of the TDF cluster from Figure 3 are zero. Buffer $b(0)$ would be:

$$b(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Neither Module A nor Module B are runnable because they would make buffer $b(0)$ go negative. Suppose that module A was the last node that tried to be scheduled and made buffer $b(0)$ go negative:

$$b(0) = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

Using Equation 2 we have:

$$D_{sug} = 0 - (-1) = 1$$

Hence, the suggested delay that needs to be added to the input port of Module A or the output port of Module B is 1.

Finally as stated in line 13, once a delay has been calculated, the scheduling algorithm from Listing 1 will run again to make sure that no other deadlocks exist. This algorithm will run iteratively, as shown in line 14, until all nodes have been scheduled q times.

Thus, the schedule for the example in Figure 2 is $ABABB$. For the example given in Figure 3, the schedule is AB.

5 Synchronizing time between DE and TDF

In [15], it is stated that the execution of the SystemC DE modules is blocked while the SystemC AMS simulation kernel executes. As a consequence, the DE simulation time (t_{DE}) does not advance at all, while the TDF simulation time (t_{TDF}) runs according to the time steps of the TDF modules and ports. On access to a TDF converter port, the SystemC AMS simulation kernel is interrupted and yields to the SystemC DE simulation kernel. This way, t_{DE} advances until it is equal to t_{TDF}. In general t_{TDF} runs ahead of the t_{DE}.

Definition 2 Causality is guaranteed, if the following Equation 3 always holds:

$$t_{\mathrm{TDF}} \geq t_{\mathrm{DE}} \tag{3}$$

.

The following synchronization operations may provoke causality issues.

1. *Periodic synchronization operation*: this operation occurs when the period of a TDF cluster has been completed. A causality check is done using Equation 3. The SystemC AMS simulation kernel is interrupted and yields to the SystemC DE simulation kernel. In consequence, the t_{DE} advances until it is equal to the t_{TDF}.
2. *Read synchronization operation*: this operation occurs when there is an access to an input converter port. A causality check is done using Equation 3. The SystemC AMS simulation kernel is interrupted and yields to the SystemC DE simulation kernel. In consequence, the t_{DE} advances until it is equal to the t_{TDF}.

3. *Write synchronization operation*: this operation occurs when there is an access to an output converter port. A causality check is done using Equation 3. But in this case, the SystemC AMS simulation kernel is not interrupted, hence the t_{DE} does not advance.

SystemC AMS may detect —but not solve— these problems during simulation, whereas SystemC MDVP can detect and solve them before simulation, but requires SystemC AMS code as an input. In [2], synchronization at converter ports is modeled with the help of Colored Timed Petri Nets [26] derived from the SystemC AMS code. Causality issues between TDF and DE MoC are then automatically checked. However, this analysis is performed from SystemC AMS code, whereas we propose a way to detect causality issues directly from SysML models.

5.1 Occurrence of time synchronization issues

In [13], possible scenarios are analyzed. Our finding was that only in two cases the SystemC AMS simulation kernel is interrupted to yield to the SystemC DE simulation kernel and in consequence t_{DE} advances:

1. When there is an access to a TDF input converter port and Equation 3 holds (read synchronization operation).
2. When one period of the TDF cluster has finished executing (periodic synchronization operation).

Time synchronization issues between DE and TDF MoCs thus can only occur when there was an access to an input converter port that advanced t_{DE} further than the t_{TDF} of the output converter port that is being currently accessed.

As a conclusion, time synchronization issues may only occur when there is a read synchronization operation before a write synchronization operation.

5.2 Timing analysis

Whenever there are multi-rate TDF clusters that interact with the DE domain using input/output converter ports, a time analysis including all the input and output converter ports is required. The interactions of each converter port of a TDF cluster with the DE domain cannot be considered independently, since

they maintain a time relation with each TDF converter port and their DE counterparts. The time relation is based on the static schedule of the TDF Cluster which is pre-computed in the same way as the static schedule of an SDF MoC [29], see Section 4. This means that once a static schedule has been found, the timestamps of the DE domain should be tracked after each execution of a TDF module that accesses a converter port.

The following equations derived from [2] show how to perform this DE timestamp tracking. Tm_{M} is the module time step of the TDF module **M**; Tp_{p} is the time step of the TDF converter port **p** of module **M**; D_{p} is the delay associated to this converter port and R_{p} is the rate associated to the converter port.

Here, j_{M} is the number of times that the TDF module **M** has been executed, considering that j_{M} is increased only when the number of samples indicated by the rates of the input and output ports have been consumed/produced and the module has finished executing. Finally, k represents the number of times that the converter port has been accessed within one activation of the module **M** - i.e. the sample number that has been produced or consumed.

Equation 4 shows how time step t_{DE} advances when there is an access to an input converter port.

$$t_{\mathrm{DE}} = (j_{\mathrm{M}} * Tm_{\mathrm{M}}) + ((k-1) * Tp_{\mathrm{p}}) - D_{\mathrm{p}} * Tp_{\mathrm{p}} \qquad k = [1...R_{\mathrm{p}}] \quad (4)$$

On the other hand, Equation 5 is used in order to determine if a future access to an output converter port will generate a causality problem. This equation determines the $t_{\mathrm{TDF}_{M.p}}$ of an access to an output converter port **p** of module **M**.

$$t_{\mathrm{TDF}_{M.p}} = (j_{\mathrm{M}} * Tm_{\mathrm{M}}) + ((k-1) * Tp_{\mathrm{p}}) + D_{\mathrm{p}} * Tp_{\mathrm{p}} \qquad k = [1...R_{\mathrm{p}}] \quad (5)$$

In order to fix a time synchronization issue that has occurred (Equation 3 does not hold), a delay has to be inserted into the TDF module's output converter port that is being currently accessed. The delay should be of at least the next integer value of the difference between the t_{DE} and the t_{TDF} of the port.

In general, the minimum required delay that has to be inserted in port **p** of module **M** ($D_{req_{M.p}}$) is defined by Equation 6. The difference between the two simulation times should be divided by the time step $Tp_{M.p}$ because the ports are accessed periodically based on this time step.

$$D_{req_{M.p}} = \left\lceil \frac{t_{DE} - t_{TDF_{M.p}}}{Tp_{M.p}} \right\rceil \tag{6}$$

Equation 6 should be used to suggest a delay to the author of the model. This is what SystemC AMS does, too, but one by one every time the model is simulated. Our algorithm will find all the delays required to fix causality issues, but it is up to the designer to decide whether he or she applies these delays, or tries to find another solution for his model.

5.3 Algorithm for detection of causality issues

As mentioned in section 5.1, time synchronization issues only occur when an access to an input converter port precedes an access to an output converter port. Thus, in order to validate that Equation 3 always holds, only read synchronization operations before write synchronization operations have to be analyzed.

For this purpose and based on the static schedule for one complete TDF cluster period, each time each module is executed, the minimum timestamp $t_{\mathrm{TDF}_{M.p}}$ of all the accessed output converter ports of the current executed module, calculated in Equation 5, should be greater or equal than the maximum timestamp (t_{DE}) of all the accessed input converter ports of previously executed TDF modules in the schedule. This is shown in Equation 7.

$$\begin{aligned} MIN\big(t_{\mathrm{TDF}_{M.p}}(m(o))\big) \geq MAX\big(t_{\mathrm{DE}}(n(i))\big) \quad & m = current\_executed\_module, \\ & o = [1...\#out\_converter\_ports], \\ & n = [first\_executed\_module...m], \\ & i = [1...\#in\_converter\_ports] \end{aligned} \tag{7}$$

If Equation 7 does not hold, then a delay calculated from Equation 6 should be inserted in the corresponding output converter port. It is important to mention that, even if these equations are only applied for the case when there is a read synchronization operation before a write synchronization operation, the same equations will work in all the other possible scenarios. Only the parameters i and o from Equation 7 should be adapted to the specific scenario that is being validated.

The following algorithm solves causality issues by iterating over additional delays and recomputing schedules until all causality issues are solved. The algorithm shown in Listings 2 and 3 makes use of the previously defined equations to detect time synchronization issues between the TDF and DE MoCs and to suggest an appropriate port Delay that can solve these causality problems.

Listing 2 Sub algorithm to detect time synchronization issues between TDF and DE

```
1: procedure walkThroughSchedule(static_schedule)
2:     max_t_DE ← 0
3:     j_M ← 0                                   ▷ For each module M
4:     for each module M in static_schedule do
5:         detectTimeSyncIssues(M.converterPorts, j_M, max_t_DE)
6:         j_M ← j_M + 1
7:     end for
8: end procedure
```

Listing 3 Algorithm to detect time synchronization issues

```
1: procedure detectTimeSyncIssues(converterPorts, j_M, max_t_DE)
2:     for each converterPort p in converterPorts do
3:         if p.origin = input then
4:             k ← R_p
5:             t_DE ← (j_M * Tm_M) + ((k − 1) * Tp_p) − D_p * Tp_p
6:             max_t_DE ← max(max_t_DE, t_DE)
7:         else if p.origin = output then
8:             k ← 1
9:             t_TDF_M.p ← (j_M * Tm_M) + ((k − 1) * Tp_p) + D_p * Tp_p
10:            if !(t_TDF_M.p ≥ max_t_DE) then
11:                D_req ← ⌈(max_t_DE − t_TDF_M.p)/Tp_p⌉
12:                CAUSALITY ERROR - STOP
13:            end if
14:        end if
15:    end for
16: end procedure
```

In the procedure walkThroughSchedule in lines 1-8, the global variable max_t_{DE} (initialized to 0 in line 2) is used to store the maximum calculated t_{DE} from all the accessed input converter ports. The variable j_M (number of times that module M has been executed) is local to each module M and is

initialized to 0 for each module M of the model at line 3. In lines 4-7, the procedure goes over the pre-computed static schedule list; for each module M in the schedule, it calls the DETECTTIMESYNCISSUES procedure in Listing 3, giving as parameters the list of all converter ports of the module M (*M.converterPorts*), and the variables j_M and max_t_{DE}. In line 6, it increments $\mathrm{J_M}$, meaning that the module M has been executed one more time.

The DETECTTIMESYNCISSUES procedure checks for time synchronization issues in the current scheduled module M. In lines 2-15, it goes over the list of *converterPorts*. In line 3, it checks whether the current port is an input converter port. Note that in line 4 the variable K is set to the value of the port rate R_p: according to Equation 7, we strive to obtain the biggest value of t_DE for that input converter port. Thus, in this case, it is useless to calculate the t_DE for previous samples (in case the port rate is greater than 1). Line 5 calculates t_DE using Equation 4. In line 6, the computed t_DE is compared against the previously stored value max_t_{DE}, in order to keep track of the maximum calculated t_DE from all the accessed input converter ports.

Line 7 checks whether the converter port is an output converter port. Notice that according to Equation 7, we are looking for the minimum $t_{\mathrm{TDF}_{M.p}}$ of all the accessed output converter ports of the current executed module. For that reason, the variable k is set to 1 in line 8, since calculating bigger values of $t_{\mathrm{TDF}_{M.p}}$ is useless in this case. Then, the $t_{\mathrm{TDF}_{M.p}}$ is calculated in line 9 using Equation 5.

Now that the minimum $t_{\mathrm{TDF}_{M.p}}$ has been computed, Equation 7 can be applied. This is done in line 10. And as explained before, if this equation does not hold, it means that a causality problem is present and a delay should be inserted in that output converter port to solve the problem. This delay is computed in line 11 using Equation 6. After this, the execution of the program is stopped because a time synchronization issue was found (line 12).

5.4 Illustrating example

We illustrate our proceeding by the example from Figure 2, applying the algorithms given in Listings 2 and 3.

Table 1 shows all the parameters to calculate t_DE, t_TDF and the detection of synchronization issues. Since the rates (R_p) of the converter ports of this model are equal to 1, the parameter k always has a value of 1: it is therefore not shown in the table.

Step	Executed Module/ port (M.p)	j_{M}	Tm_{M} (ms)	Tp_{p} (ms)	D_{p}	t_{DE} (ms) (Eq. 4)	max_t_{DE} (ms)	$t_{\mathrm{TDF}_{M.p}}$ (ms) (Eq. 5)	Causality Check (Eq. 7)
1	A.in	0	6	6	0	0	0	–	–
2	A.in	1	6	6	0	0	0	–	–
3	B.out	0	4	4	0	–	0	0	True
4	B.out	1	4	4	0	–	0	0	–
5	A.in	1	6	6	0	6	6	–	–
6	A.in	2	6	6	0	6	6	–	–
7	B.out	1	4	4	0	–	6	4	**False**

Table 1: Execution without delay, computation of DE and TDF simulation times

Below, a description of the execution and calculation of the simulation times of the model is presented. Remember from section 4.3 that the static schedule of this model is **ABABB**.

1. Starting from the WALKTHROUGHSCHEDULE procedure, module A is executed for the first time ($j_{\mathrm{A}} = 0$) according to the schedule. Procedure DETECTTIMESYNCISSUES is called. Module A has only one converter port, which is an input converter port. Since $R_{\mathrm{in}} = 1$, k is set to 1 as mentioned before. The t_{DE} is calculated and the max_t_{DE} is chosen as shown in Table 1. Since there is no access to an output converter port, the causality check is not performed.
2. There are no more converter ports, so the procedure DETECTTIMESYNCISSUES finishes and j_{A} is incremented.
3. Next, module B is executed for the first time ($j_{\mathrm{B}} = 0$) according to the schedule. Procedure DETECTTIMESYNCISSUES is called. Module B has only one converter port, which is an output converter port. Hence k is set up to 1, and $t_{\mathrm{TDF}_{B.out}}$ is calculated. Then the causality check using Equation 7 is performed. In this case $max\_t_{\mathrm{DE}} = 0$ ms and $t_{\mathrm{TDF}_{B.out}} = 0$ ms so there is no causality problem.
4. Module B has no more converter ports, so DETECTTIMESYNCISSUES finishes and j_{B} is incremented.
5. Now, module A is executed for the second time ($j_{\mathrm{A}} = 1$). The procedure DETECTTIMESYNCISSUES is called. For its only input converter port the t_{DE} is calculated and max_t_{DE} is chosen.
6. There are no more converter ports, so the procedure DETECTTIMESYNCISSUES finishes and j_{A} is incremented.

7. Then, module B is executed for the second time ($j_{\mathrm{B}} = 1$). The procedure DETECTTIMESYNCISSUES is called. For the output converter port, the $t_{\mathrm{TDF}_{B.out}}$ is calculated. As it can be seen in Table 1, since the max_t_{DE} of all previously accessed input converter ports is 6 ms and $t_{\mathrm{TDF}_{B.out}} = 4$ ms, Equation 7 does not hold and a causality problem is generated. The minimum required delay is calculated as before using Equation 6, that is $D_{req_{B.out}} = 1$. Execution is then stopped.

Now we can apply the same algorithm in order to verify that the synchronization issues are solved when this new delay is inserted. Table 2 shows how the simulation times are calculated using this new delay as well as the causality check.

Step	Executed Module/ port (M.p)	j_{M}	Tm_{M} (ms)	Tp_{p} (ms)	D_{p}	t_{DE} (ms) (Eq. 4)	max_t_{DE} (ms)	$t_{\mathrm{TDF}_{M.p}}$ (ms) (Eq. 5)	Causality Check (Eq. 7)
1	A.in	0	6	6	0	0	0	–	–
2	A.in	1	6	6	0	0	0	–	–
3	B.out	0	4	4	1	–	0	4	True
4	B.out	1	4	4	1	–	0	4	–
5	A.in	1	6	6	0	6	6	–	–
6	A.in	2	6	6	0	6	6	–	–
7	B.out	1	4	4	1	–	6	8	True
8	B.out	2	4	4	1	–	6	8	True
9	B.out	2	4	4	1	–	6	12	True
10	B.out	3	4	4	1	–	6	12	True

Table 2: Execution with delay and computation of DE and TDF simulation times

Below, the description of the execution of the model with the new delay added is presented.

1. Starting from the WALKTHROUGHSCHEDULE procedure, module A is executed for the first time ($j_{\mathrm{A}} = 0$) according to the schedule. Procedure DETECTTIMESYNCISSUES is called. Module A has only one converter port, which is an input converter port. Since $R_{\mathrm{in}} = 1$, k is set to 1 as mentioned before. The t_{DE} is be calculated and max_t_{DE} is chosen as shown in Table 2. Since there is no access to an output converter port, the causality check is not performed.
2. There are no more converter ports, so the procedure DETECTTIMESYNCISSUES finishes and j_{A} is incremented.

3. Next, module B is executed for the first time ($j_{\mathrm{B}} = 0$) according to the schedule. Procedure DETECTTIMESYNCISSUES is called. Since module B has only one output converter port, the $t_{\mathrm{TDF}_{B.out}}$ is calculated using the new delay $D_{out} = 1$ as shown in Table 2. Then the causality check using Equation 7 is performed. In this case $max\_t_{\mathrm{DE}} = 0$ ms and $t_{\mathrm{TDF}_{B.out}} = 4$ ms so there is no causality problem.
4. Module B has no more converter ports, so DETECTTIMESYNCISSUES finishes and j_{B} is incremented.
5. Now, module A is executed for the second time ($j_{\mathrm{A}} = 1$). The procedure DETECTTIMESYNCISSUES is called. For its only input converter port the t_{DE} is calculated and max_t_{DE} is chosen.
6. There are no more converter ports, so the procedure DETECTTIMESYNCISSUES finishes and j_{A} is incremented.
7. Then, module B is executed for the second time ($j_{\mathrm{B}} = 1$). The procedure DETECTTIMESYNCISSUES is called. For the output converter port, the $t_{\mathrm{TDF}_{B.out}}$ is calculated. As it can be seen in Table 2, since the max_t_{DE} of all previously accessed input converter ports is 6 ms and $t_{\mathrm{TDF}_{B.out}} = 8$ ms, Equation 7 holds and there is no causality problem.
8. Module B has no more converter ports, so DETECTTIMESYNCISSUES finishes and j_{B} is incremented.
9. According to the schedule, module B needs to be executed one last time ($j_{\mathrm{B}} = 2$). The procedure DETECTTIMESYNCISSUES is called. For the output converter port, the $t_{\mathrm{TDF}_{B.out}}$ is calculated. Once again, since $max\_t_{\mathrm{DE}} = 6$ ms and $t_{\mathrm{TDF}_{B.out}} = 12$ ms, Equation 7 holds and there is no causality problem.
10. Finally, module B has no more converter ports, so DETECTTIMESYNCISSUES finishes and j_{B} is incremented.

Synchronization issues are thus avoided with the introduction of the delays. Finally, our proposal to check for causalities and deduce schedules and delays can be integrated in our model-driven approach. This integration also allows to generate a causality-issue free simulation code (SystemC, SystemC AMS).

6 Comparison

This section compares our schedulability and causality approach to the ones of SystemC MDVP [2] and the SystemC AMS simulator. We selected a toy example presented in [18] and a vibration sensor case study first introduced in [3].

SystemC AMS detects causality issues only at run time. Each causality issue is treated separately, the simulator thus has to be run anew each time.

The SystemC MDVP simulator can detect time synchronization issues between the DE and TDF MoCs by transforming the TDF clusters and their interaction with the DE domain into an equivalent timed-Colored Petri Net model. To do so, the SystemC code must be generated and executed in order to find any possible causality problems during the pre-simulation phase and provide the necessary delay suggestions to avoid these issues.

6.1 Dependency of delays on the static schedule

The model shown in Figure 4 stems from the SystemC MDVP example models [5]. **Tm** is provided for module **A** only.

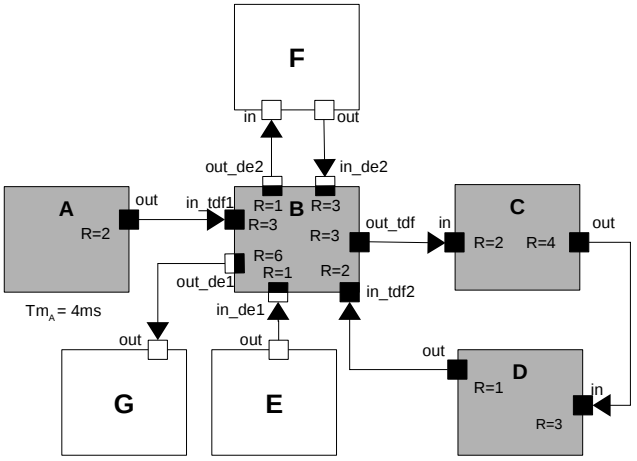

Fig. 4: SysML Model with feedback loops and multiple DE components

Our algorithm, implemented in TTool, finds a schedulability issue due to delays missing in the feedback loop between **B**, **C** and **D**. Also, causality issues are detected within the converter ports of module **B**. These causality issues are due to the interaction between the TDF module **B** with the DE modules **E**, **F** and **G** through the use of converter ports.

The same model is simulated in SystemC MDVP, see Table 3. If suggested delays for B.out_de_1 = 4 and B.out_de_2 = 1 match, TTool suggests to insert a delay of 3 in the B.in_tdf_2 port, while MDVP suggests to insert delays of 3 in the C.in port and 1 in the D.out port.

Port	TTool	MDVP
B.out_de_1	4	4
B.out_de_2	1	1
B.in_tdf_2	3	–
C.in port	–	3
D.out	–	1

Table 3: Suggested delays in TTool and SystemC MDVP

TTool	MDVP
`A-A-B-A-C-D-B-C-D-C-D-D`	`C-D-A-A-B-A-C-D-C-D-B-D`

Table 4: Comparison of schedules

The difference concerning the suggested delays is in fact due to the static schedule each tool produces. Table 4 compares the schedules produced by TTool and SystemC MDVP.

The static schedule of TTool shows that module **A** is executed twice in the beginning. Since module **B** requires 2 samples in port B.in_tdf2 to be executed, a delay of 2 is needed. The next time module **B** needs to be executed, module **D** has only delivered 1 sample to **B**, so module **B** needs another delay of 1; in total it needs a delay of 3 in its input.

In the static schedule determined by MDVP, **C** needs to be executed first, requiring a delay of 3 in its input port C.in. **D** executes and delivers 1 sample to **B**. Again, when module **B** needs to execute, it still requires one extra sample, thus a delay in D.out of 1 is suggested. The same model is simulated in SystemC AMS without inserting any delays. Since there are missing delays in the feedback loop, the simulator indicates a schedule error.

We conclude that both delay suggestions, the one given by TTool and the one by SystemC MDVP are equally valid, since they both solve the schedulability problem that exists when delays are missing in a feedback loop.

This model demonstrates that in the presence of feedback loops, the delay suggestions by SystemC MDVP and TTool can differ. They depend on the static schedule computed by each simulator. Anyway, both suggestions are found to be valid when used in the model created in SystemC AMS.

6.2 Vibration sensor

The small case study is based on a vibration sensor model taken from the TDF model examples provided with the SystemC MDVP simulator [3,5]. It was already shown in [14] to illustrate the implementation of our prototyping tool.

The model of the vibration sensor in SystemC AMS notation is shown in Figure 5. It consists of six TDF modules and one DE module as described below.

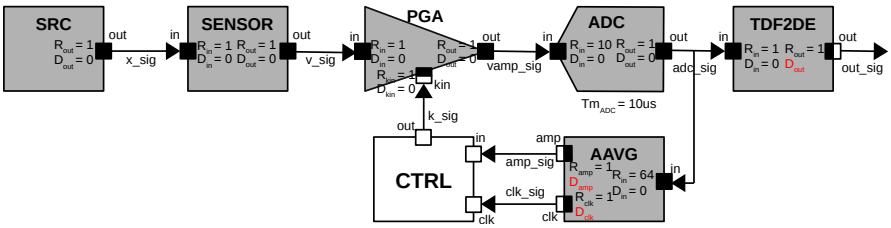

Fig. 5: Vibration sensor model (Adapted from [3]).

The **SRC** module represents the vibration source, which is modeled as a generator of harmonic sinusoidal wavelets representing a displacement signal (**x_sig**).

The **SENSOR** module represents the vibration sensor itself. It takes as an input the displacement signal (**x_sig**) and gives as an output a voltage signal (**v_sig**) which is proportional to the vibration velocity.

The **PGA** module represents a programmable gain amplifier, that amplifies the voltage input signal (**v_sig**) by a factor of 2^k, where k is the input value from signal **k_sig**. This signal is controlled by the gain controller DE module **CTRL**. The output is an amplified voltage signal **vamp_sig**.

The **ADC** module represents an analog to digital converter with a resolution of 5 bits. The ADC has a rate of 10 in its input port `in`. Hence, it takes 10 samples from the amplified voltage signal **vamp_sig** and produces a digitized integer value of N-bits (**adc_sig**) where the most significant bit corresponds to the sign. The module time step is assigned to this module as 10μs.

TDF2DE is a converter module from the TDF signal **adc_sig** to a DE signal **out_sig**. Note that the delay D_{out}, written in red, of its output converter port `out` has not been set yet.

The **AAVG** module represents an absolute amplitude averager. It calculates and outputs to the **amp_sig** the absolute average amplitudes of the received samples from the **adc_sig**. Its input port `in` has a rate of 64, it will thus receive 64 samples to calculate the absolute average amplitude. This module also generates a rate 2 clock signal **clk_sig** at its output port `clk` meaning that a clock edge will be generated twice per activation of the module. Note that initially the delays D_{clk} and D_{amp} of its output converter ports, marked in red, have not yet been set.

The **CTRL** DE module represents the gain controller. It controls the output signal **k_sig** based on the calculated absolute average amplitude given by **amp_sig**, and two given thresholds *low_threshold* and *high_threshold*.

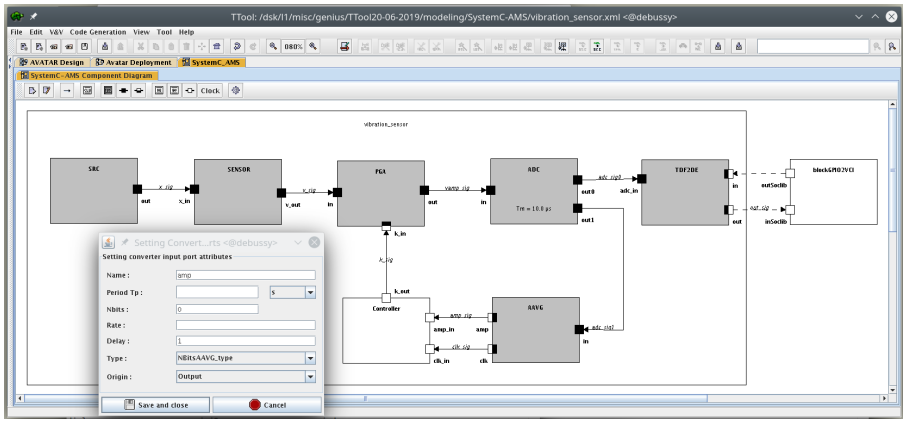

Fig. 6: Screenshot showing the SystemC AMS panel and port parameter configuration

6.3 Co-simulation

In the following, we employ the vibration sensor example to compare the virtual prototypes generated by TTool to SystemC AMS and MDVP. The vibration sensor was modeled in TTool, as shown in Figure 6.

In contrast to the SystemC AMS graphical notation, in our SysML notation, ports and modules are parameterized in pop-up menus, limiting the overload of the graphical representation. On the lower left of the Figure, we show the port parameter configuration window for the output converter port **amp**, where the Delay has been set to 1.

TTool supports full-system co-simulation (i.e. including software) of the analog and digital part, while MDVP co-simulates analog and digital hardware. Both use SoCLib models of digital hardware, whereas SystemC AMS uses TDF models for analog and DE models for digital hardware. All combine event-based and TDF co-simulation.

For a first validation, the three output converter port delays (in red in Figure 5) were set to 0. Figure 7 shows the output of the validation panel of TTool's code generation window: causality issues were found and delays on three ports were suggested to solve them.

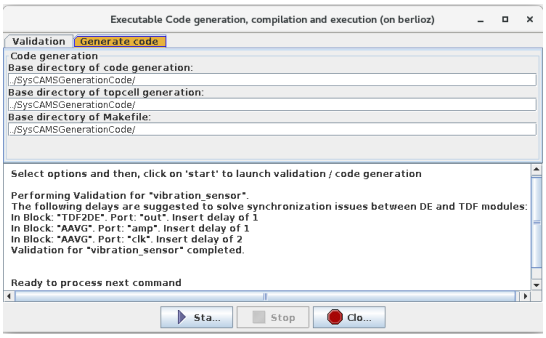

Fig. 7: Suggested delays for the vibration sensor in TTool.

The model was also simulated with the MDVP simulator without providing any delays for its output converter ports. Figure 8 shows the output of the simulator. It suggests the same three delays as TTool.

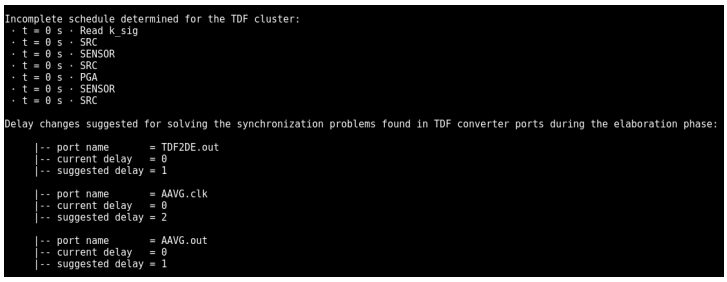

Fig. 8: Suggested delays for the vibration sensor in SystemC MDVP.

Finally, we modify a SystemC AMS model taken from the H-Inception project in order to include the same parameters as the ones used in TTool and

SystemC MDVP —i.e. the same port rates and ADC resolution—. Again, the simulation was executed without assigning any delays to the output converter ports.

Here, the simulator has to be run three times. As shown in Figure 9, synchronization issues are detected by the simulator each time the simulation is run, and delays referring to time units are suggested to solve the causality problems. For the first time the simulation was run, a delay of 9 μs in port `tdf2de.out` is suggested as shown in Figure 9(a). This delay corresponds to a delay of 1 since the propagated time step of this port is 10 μs. After setting this delay, the simulation was run again. This time another synchronization problem was found, and a delay of 639 μs is suggested to the `aavg.clk` port, as Figure 9(b) shows. Since the time step of this port is of 320 μs, a delay of 2 is needed. Finally, after setting this new delay, the simulation was run for the third time. Another causality problem was detected, and a delay of 639 μs suggested for port `aavg.amp`. The time step of this port is of 640 μs, thus a delay of 1 is required (Figure 9(c)).

```
Error: SystemC-AMS: sca-de synchronization failed in: 0 ../../../../../source/src/scam
s/impl/core/sca_solver_base.cpp line: 529 current sca-time: 0 s current sc-time: 9 us
sca-next-time:   10 us insert da delay of at least: 9 us in: tdf2de.out
```

(a)

```
Error: SystemC-AMS: sca-de synchronization failed in: 0 ../../../../../source/src/scams
/impl/core/sca_solver_base.cpp line: 529 current sca-time: 0 s current sc-time: 639 us
sca-next-time:   320 us insert da delay of at least: 639 us in: aavg.clk
```

(b)

```
Error: SystemC-AMS: sca-de synchronization failed in: 0 ../../../../../source/src/scams
/impl/core/sca_solver_base.cpp line: 529 current sca-time: 0 s current sc-time: 639 us
sca-next-time:   640 us insert da delay of at least: 639 us in: aavg.amp
```

(c)

Fig. 9: Suggested delays for the vibration sensor in SystemC AMS: three runs.

All the suggested delays by SystemC AMS simulator are the same as the ones suggested by TTool and the SystemC MDVP simulator, as shown in Table 5. The main difference is that in TTool, the causality problems can be found directly from SysML views thus **before** any code is generated. In SystemC MDVP, the synchronization issues are found in the pre-simulation phase. That means that the SystemC MDVP model needs to be executed only

once to find any synchronization problems. In SystemC AMS, these issues are found during the simulation phase, meaning that the simulation needs to be executed once per causality problem that is found. In this case, it needed to be executed three times.

Port	TTool	SystemC MDVP	SystemC AMS
TDF2DE.out	1	1	1
AAVG.amp	1	1	1
AAVG.clk	2	2	2

Table 5: Comparison of suggested delays for the vibration sensor model.

Once all the delays are set, code generation can be executed and the traces compared.

This case study demonstrates that thanks to the implementation in TTool of our contribution, we can obtain, from SysML view, the same results as the ones suggested by the SystemC AMS and the SystemC MDVP simulators, as long as the computed static schedules of the three simulators present a similar behavior.

7 Conclusion and future work

The main contribution of the paper is the detection of valid TDF schedules and resolution of causality issues directly from high level SysML models, i.e. before any code is generated. Conversely, in other approaches SystemC AMS code needs to be generated / written by hand, compiled and executed.

Moreover, our approach can automatically solve causality problems by adding delays inside feedback loops, and handle situations that cannot be handled by other work. More concrete use cases, not presented in this paper, also demonstrate the relevance of our contribution and the interest of our implementation.

We are currently improving our algorithms so that suggested delays are always optimal. Reusing concepts coming from the simulation engine of SystemC AMS may help reach this goal.

Simulation feedback is currently limited to the digital parts and only semi-automatic. Automating and extending this mechanism to the entire system would enable us to propose a full design space exploration environment for Analog/Mixed Signal systems.

References

1. Accellera Systems Initiative: SystemC AMS extensions Users Guide, Version 1.0 (March 2010)
2. Andrade, L., Maehne, T., Vachoux, A., Aoun, C.B., Pêcheux, F., Louërat, M.M.: Pre-simulation symbolic analysis of synchronization issues between discrete event and timed data flow models of computation. In: Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1671–1676. Grenoble, France (2015)
3. Andrade Porras, L.: Principles and implementation of a generic synchronization interface between SystemC AMS models of computation for the virtual prototyping of multi-disciplinary systems. Ph.D. thesis, UPMC (2016)
4. Apvrille, L.: TTool, an open-source UML and SysML toolkit, `http://ttool.telecom-paris.fr/`
5. Ben Aoun, C.: Principles and Realization of a Virtual Prototyping Environment for Composable Heterogeneous Systems. Ph.D. thesis, UPMC (2017)
6. Bengtsson, J., Yi., W.: Timed automata: Semantics, algorithms and tools. In: Lecture Notes on Concurrency and Petri Nets. pp. 87–124. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag (2004)
7. Benveniste, A., Bourke, T., Caillaud, B., Pagano, B., Pouzet, M.: A type-based analysis of causality loops in hybrid modelers. In: 17th International Conference on Hybrid Systems: Computation and Control (HSCC'14). pp. 71–82. Berlin, Germany (Apr 2014), `http://zelus.di.ens.fr/hscc2014/fullpaper.pdf`
8. Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Le Guernic, P., De Simone, R.: The synchronous languages 12 years later. Proceedings of the IEEE **91**(1), 64–83 (2003)
9. Beyond Dreams: (Design Refinement of Embedded Analogue and Mixed-Signal Systems) (2008-2011), http://projects.eas.iis.fraunhofer.de/beyonddreams
10. Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Elmqvist, H., Junghanns, A., Mauß, J., Monteiro, M., Neidhold, T., Neumerkel, D., et al.: The functional mockup interface for tool independent exchange of simulation models. In: Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical University Dresden, Germany. Linköping University Electronic Press (2011)
11. Boulanger, F., Jacquet, C., Hardebolle, C., Prodan, I.: Tesl: a language for reconciling heterogeneous execution traces. In: Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM/IEEE International Conference on. pp. 114–123. Lausanne, Switzerland (Oct 2014). https://doi.org/10.1109/MEMCOD.2014.6961849, `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6961849`
12. Concepcion, A.I., Zeigler, B.P.: DEVS formalism: A framework for hierarchical model development. IEEE Trans. on Software Engineering **14**(2), 228–241 (1988)
13. Cortés Porto, R.: Integration of SystemC-AMS Simulation Platforms into TTool. Master's thesis, Kaiserslautern (2018), `www-soc.lip6.fr/~genius/research`
14. Cortés Porto, R., Genius, D., Apvrille, L.: Modeling and virtual prototyping for embedded systems on mixed-signal multicores. In: RAPIDO (2019)
15. Damm, M., Grimm, C., Haas, J., Herrholz, A., Nebel, W.: Connecting SystemC-AMS models with OSCI TLM 2.0 models using temporal decoupling. In: FDL. pp. 25–30 (2008)

16. Davare, A., Densmore, D., Meyerowitz, T., Pinto, A., Sangiovanni-Vincentelli, A., Yang, G., Zeng, H., Zhu, Q.: A next-generation design framework for platform-based design. In: Conference on using hardware design and verification languages (DVCon). vol. 152 (2007)
17. Demathieu, S., Thomas, F., André, C., Gérard, S., Terrier, F.: First experiments using the uml profile for marte. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). pp. 50–57. IEEE (2008)
18. Einwich, K.: SystemC AMS PoC2.1 Library (2016)
19. Fritzson, P., Engelson, V.: Modelica—a unified object-oriented language for system modeling and simulation. In: European Conference on Object-Oriented Programming. pp. 67–90. Springer (1998)
20. Genius, D., Cortés Porto, R., Apvrille, L., Pêcheux, F.: A tool for high-level modeling of analog/mixed signal embedded systems. In: MODELSWARD. Scitepress (2019)
21. Genius, D., Li, L.W., Apvrille, L.: Model-Driven Performance Evaluation and Formal Verification for Multi-level Embedded System Design. In: MODELSWARD. SCITEPRESS (2017)
22. H-Inception Consortium: Heterogeneous Inception Project (2012-2015), https://www-soc.lip6.fr/trac/hinception
23. Herrera, F., Villar, E.: A framework for heterogeneous specification and design of electronic embedded systems in systemc. ACM TODAES **12**(3), 22 (2007)
24. IEEE: SystemC. IEEE Standard 1666-2011 (2011)
25. Jantsch, A.: Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation. Elsevier (2003)
26. Jensen, K., Kristensen, L.M.: Coloured Petri Nets. Modelling and Validation of Concurrent Systems. Springer (2009)
27. Lee, E.A.: Disciplined heterogeneous modeling. In: Petriu, D., Rouquette, N., Haugen, O. (eds.) MODELS. pp. 273–287. Springer LNCS 6395 (Oct 2010)
28. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proceedings of the IEEE **75**, 1235–1245 (1987)
29. Lee, E.A., Messerschmitt, D.G.: Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE Trans. on Comp. **C-36**(1), 24–35 (1987). https://doi.org/10.1109/TC.1987.5009446
30. Li, L.W., Genius, D., Apvrille, L.: Formal and Virtual Multi-level Design Space Exploration. Springer Communications in Computer and Information Science, vol 880 (2018)
31. Mallet, F.: Clock constraint specification language: specifying clock constraints with uml/marte. Innovations in Systems and Software Engineering **4**(3), 309–314 (2008)
32. Niaki, S.H.A., Jakobsen, M.K., Sulonen, T., Sander, I.: Formal heterogeneous system modeling with systemc. In: FDL. pp. 160–167. IEEE (2012)
33. Patel, H.D., Shukla, S.K.: Towards a heterogeneous simulation kernel for system-level models: a systemc kernel for SDF models. TCAD **24**(8), 1261–1271 (2005)
34. Selic, B., Gérard, S.: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems. Elsevier (2013)
35. SocLib consortium: The SoCLib project: An integrated system-on-chip modelling and simulation platform. Tech. rep., CNRS (2003), www.soclib.fr
36. Syriani, E., Gray, J., Vangheluwe, H.: Modeling a model transformation language. In: Domain Engineering, pp. 211–237. Springer (2013)

37. Taha, S., Radermacher, A., Gérard, S.: An entirely model-based framework for hardware design and simulation. In: Distributed, Parallel and Biologically Inspired Systems - 7th IFIP TC 10 Working Conference, DIPES 2010 and 3rd IFIP TC 10 International Conference, BICC 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings. IFIP Advances in Information and Communication Technology, vol. 329, pp. 31–42. Springer (2010)
38. Vachoux, A., Grimm, C., Einwich, K.: Analog and mixed signal modelling with SystemC-AMS. In: ISCAS (3). pp. 914–917. IEEE (2003), http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8570
39. Zhao, C., Kazmierski, T.J.: An extension to SystemC-A to support mixed-technology systems with distributed components. In: DATE. pp. 1–6. IEEE (2011)
40. Zhu, J., Sander, I., Jantsch, A.: Hetmoc: Heterogeneous modelling in systemc. In: FDL. pp. 1–6. IET (2010)