Computer Communications 36 (2012) 8–19

Contents lists available at SciVerse ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A comprehensive co-simulation platform for cyber-physical systems

Ahmad T. Al-Hammouri *

Department of Network Engineering and Security, Jordan University of Science and Technology, Irbid 22110, Jordan

ARTICLE INFO

Article history:
Received 27 March 2011
Received in revised form 3 November 2011
Accepted 6 January 2012
Available online 15 January 2012

Keywords:
Sensor-actuator networks (SANETs)
Networked control systems (NCSs)
Distributed control systems (DCSs)
Smart power grid
Industrial automation

ABSTRACT

Cyber-physical systems (CPSs) are the synergy of the physical world with the cyber world. CPSs will bring about unprecedented applications that will enable the monitoring and controlling of the physical environments. CPSs' further progress necessitates the availability of co-simulation platforms that can capture both the physical and the communication dynamics. In this paper, we build on our previous experiences to build a comprehensive co-simulation platform for CPSs. The newly developed platform enjoys several indispensable features. In the process of discussing the steps to engineer the platform, we present several design alternatives that might prove beneficial in other future tools that combine different simulator environments. We discuss thoroughly why we rule in or rule out each of such alternatives. Then, we validate the developed platform to make sure it works correctly. Finally, we present demonstrative examples showing the capabilities of the platform.

1. Introduction

Cyber-physical systems (CPSs) are the synergy of the physical world with the cyber world. The ultimate objective is to attain a better quality of life by monitoring, affecting, and controlling the physical world within which we live using the capabilities of the cyber world. These capabilities encompass sensing, computation, communication, and actuation tasks. CPSs applications are tremendous and include hazardous and remote explorations; aerospace and automotive control; power grid control; and industrial automation [1]. For CPSs to attain its promising objective of enhancing the quality of life, they must be dependable, fault-tolerant, effective, and efficient.

CPSs are generally complex to study, analyze, and design because of the following two main reasons:

- CPSs combine two different domains: physical and digital. Whereas the former is often expressed in terms of continuous mathematics, the latter is usually modeled by discrete mathematics. Although, some physical systems, especially engineered ones, may be modeled in discrete forms, the time scales are different between the models of physical systems and those of cyber counterparts.
- CPSs are often intermittent and usually communicate over heterogeneous and nondeterministic networks. These factors call for probabilistic models that are usually hard to solve.

These two reasons render any mathematical analyses of CPSs intricate. When analytical approaches fall far short of being adequate in studying complex systems, simulation arises as a powerful problem-solving technique for studying and analyzing such complex systems [2]. Therefore, CPSs' realization and further progress will critically depend on the existence of appropriate simulation tools [3]. Moreover, due to the intrinsic tight coupling between the physical and the cyber worlds, CPSs necessitate *co-simulation* [1], which dictates the joint and concurrent simulation of the two domains.

In this paper, we build on our previous experiences to build a *comprehensive* co-simulation platform for CPSs. The newly developed platform integrates `ns-2`, a network simulator, and Modelica, a modeling language for large-scale, complex physical systems. Because each island, `ns-2` or Modelica, has its own notion of time, *synchronization* between their simulated times arises as a major issue. As it will be elaborated upon in the coming sections, the way to achieve the synchronization significantly impacts the capabilities of a given co-simulation tool. Introducing a delicate solution for the synchronization issue, we here address the limitations of and improve over the previously developed co-simulation tools, including our own older version [4,1]. With the proposed solution, the new platform is now capable of supporting asynchronous events inside physical systems, a feature that was missing in most of previous tools. Such added feature is essential and is not merely cosmetic. With the new platform, all physical systems-related functionalities—e.g., scheduling sampling events and sending packets—are taken care of inside Modelica. On the other hand, networks-related functionalities are taken care of inside `ns-2`. This separation has two advantages. First, it is the natural way that aligns with the vital principles of modular design [5]. Second, it

* Tel.: +962 2 720 1000 X22619; fax: +962 2 7201077.
E-mail address: hammouri@just.edu.jo
URL: http://www.just.edu.jo/~hammouri

0140-3664/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2012.01.003

is necessary for the modeling of event-based sampling [6,7] arising in several real-life examples, e.g., sampling when a physical quantity exceeds a threshold, sampling when a physical quantity changes by a specific amount, or sampling in response to asynchronous events whose times are not known beforehand [6,7].

After validating the platform for its correct operation, we use it to simulate two representative examples: one in industrial automation and one in power grid. These two examples are at the heart of CPSs' real-life applications.

In summary, the newly developed platform enjoys the following indispensable features:

1. It relies on the two well-engineered, credible, and well-serviced tools: `ns-2` and Modelica.
2. It can simulate fine-grained and detailed dynamics of communication networks.
3. It utilizes the availability of existent physical systems models for various domains, e.g., mechanical, electrical and electronic, hydraulic, thermal, control, and electric power systems.
4. It is capable of simulating asynchronous events inside the physical systems and inside the cyber systems.

The rest of this paper is structured as follows. Section 2 surveys the related co-simulation platforms for CPSs, including our own older version that we improve upon in this paper. In Section 3, we present the implementation details of the older version and discuss its limitations. Section 4 discusses the full implementation details of the newer comprehensive platform. In Section 5, we present the techniques that we followed to verify the correct operation of the newly developed tool. Section 6 presents several demonstrative and interesting examples illustrating the capabilities of the new tool. Finally, Section 7 concludes the paper.

2. Related work

Since advances in the design and synthesis of CPSs rely eventually on analyzing the simulation-generated data, the co-simulation platform must accurately capture the actual behavior of both physical and communication dynamics. Fine-grained dynamics are required because simulation-generated data can be only as good as the simulation models are. So, if the physical or communication models deviate from the actual real-life systems, output data can be misleading and consequently resulting in erroneous decision making during the design stage.

Although an intuitive way is to engineer a new tool from scratch that would contain building modules for physical components and others for communication components, a good practice and a basic principle in engineering is to avoid reinventing the wheel and to rely on well-developed ideas as much as possible. This latter approach has been pursued almost in all research efforts that targeted the simulation of CPSs; see for example [8–10,1]. However, these co-simulation platforms, which adapt already existent tools, have diverged into three different avenues [1]:

1. Extending physical systems simulators to also simulate the events and dynamics of communication networks. Examples include the original TrueTime [8], which is Simulink® based; the newer TrueTime generations [11–13], which are all Modelica based; and the VisualSense, which is an extension of the Ptolemy framework [14]. All these tools have support for only local-area-networks simulations. Specifically, they allow the simulation of only the physical and the medium-access layer. This limitation inhibits each tool's applicability to more general networks that incorporate higher-layer network protocols—e.g., routing, transport, and application protocols—and geographically distributed networks, i.e., WANs. Providing support for general network settings can be a formidable task because such higher-protocols, in general, utilize complex algorithms that are distributed in nature and encompass multi-hop nodes [1]. Also, these tools usually abstract away the network into stochastic-delay models, and such they lack the fine-grained simulations of network dynamics (a sought-for aim for the reasons mentioned at the beginning of this section).
2. Extending a network simulator to support physical systems simulations. Examples include the `Agent/Plant` [10] and its successors [15]. The problem with such approach is that physical dynamics and control algorithms need to be modeled explicitly by differential–algebraic equations (DAEs) that are to be solved within the simulation script or via a call to an outside utility, e.g., Matlab. Except for relatively simple systems, this process can be a daunting task.
3. Marrying a full-blown network simulator with a full-blown physical systems simulator. Examples include the ADEVS/`ns-2` integrated tool [9], the Simulink®/`ns-2` combined tools [16,17], and the Modelica/`ns-2` integrated tool [1].

In CPSs, there are two radically different realms (i.e., the physical and cyber worlds) with each has its own specialized and mature simulation tools. Therefore, it is conspicuously wise to combine two domain-specific tools into one that utilizes the best features of individual simulators. The tool in [9] requires both explicit modeling of physical systems with DAEs, and dividing a system into continuous processes and discrete ones. The ADEVS/`ns-2` can therefore be thought of as an enhanced version of the `Agent/Plant`. However, it still lacks the ease of modular modeling found in, say, Modelica or Simulink®. The tool presented in [16] exploits exactly the same synchronization mechanism as that of our previous tool [1]. Therefore, it suffers from the same limitations as of our older tool: no support for asynchronous events inside physical systems (elaboration on this point is discussed in Section 3). Moreover, the tool requires that round-trip delays (or the total control loop delays) be shorter than sampling periods, which can be a valid assumption in some scenarios on a scale of local-area-networks but this assumption does not usually hold in wide-area settings, e.g., the Internet. The co-simulation platform presented in [17] provides a GUI to facilitate construction of wireless networks in `ns-2` and tuning of controller's parameters. The platform also improves on the wireless signal propagation model existing in `ns-2`. Although the paper mentioned that it *corrected* the synchronization mechanism over previous versions, insufficient details are provided about the new synchronization solution. All what is mentioned is that Simulink® controls the synchronization by instructing `ns-2` to advance to a given time and that Simulink® then advances *one time-step*. There arise two concerns here: it is unclear what 'one time-step' means, and it is unclear how to deal with the situation if `ns-2` has data to deliver to Simulink® before the given time ordained by Simulink®. In general, in all previously mentioned tools, no attempts were made to validate the correctness of the presented tool.

In this paper, we improve on the Modelica/`ns-2` tool that we previously built [1] by targeting its shortcomings to provide a more comprehensive co-simulation platform for CPSs. The differences between the older and the newer versions are detailed in next two sections. We here summarize the reasons of choosing `ns-2` and Modelica for the simulation of physical systems dynamics and of communication networks dynamics, respectively [1]. For

the network side, we choose ns-2 among other alternatives because

- ns-2 [18] is a free, widespread packet-level simulator.
- ns-2 simulates the exact dynamics and events of individual packets while traversing network elements, e.g., communication links and routers.
- ns-2 supports various routing, transport, application protocols.
- ns-2 is capable of simulating wired, wireless, local- and wide-area networks.
- ns-2 is evolvable by exposing well-defined APIs that greatly facilitate developing new protocols and algorithms.

As for the physical systems side, we choose Modelica among other candidates because

- Modelica [19] is a modeling language for large-scale complex physical systems.
- Modelica is an object-oriented language and supports model construction and reusability.
- Modelica allows acausal modeling.
- Several ready Modelica libraries are available for different domains, e.g., mechanical, electrical and electronic, hydraulic, thermal, control, and electric power systems.
- Several commercial and open source simulation environments are available for Modelica.

3. An older version of the Modelica/ns-2 co-simulation platform

In [4,1], we presented our co-simulation tool that combined ns-2 and Modelica. In this section, we review the implementation details and the limitation of that tool.

3.1. Implementation details of the older version

Modelica and ns-2 are run as two separate processes. The communication between the two processes is achieved via UNIX named pipes. In general, when combining two or more simulators, where each runs as a separate process, to form one hybrid simulator, the major issue becomes how to synchronize their simulated clocks. To illustrate, consider the following example of only two simulators, S_1 and S_2. Both S_1 and S_2 start at simulation time t_0 and at simulation time t_1, S_1 needs to convey data to S_2. Intuitively, we cannot let S_1 and S_2 freely run because it might happen that S_2 passes t_1 before S_1, which will lead to erroneous operation of and will incapacitate the combined tool. Next, we discuss three mechanisms to synchronize S_1 and S_2.

3.1.1. Predetermined communication time instants

The race condition between S_1 and S_2 can be solved straightforward if the time instant t_1 is predetermined and known beforehand. In such case, we allow S_1 and S_2 run freely until t_1, at which point they both pause, they exchange data, and then they resume execution until the next predetermined communication time instant t_2, and so on. However, in realistic simulations, not all time instants at which communication between the two simulators must occur are often known a priori. For example, in most cases, the communication between the two simulators is triggered by internal events inside one or both simulators depending on meeting some conditions, which might be stochastic or even deterministic, but are not known a priori.

3.1.2. Real-time synchronization

Several simulators, including ns-2 and Modelica, possess the capability of synchronizing their simulated clocks with real-life time, i.e., the wall-clock time. If the two simulators have this real-time synchronization feature, the race condition and synchronization between the two simulators is therefore completely resolved because both simulators will advance at the same rate (the wall-clock rate) and they will never outpace one another. However, this functionality is still unimplemented in some simulators and is still experimental and correct operation is not guaranteed in others, e.g., ns-2 [20,17]. Another disadvantage of real-time synchronization is that simulations will progress at the pace of real-life clocks and thus will take long time to finish for long-time simulations, i.e., an hour-long simulation will take exactly one real hour to finish no matter how powerful the hardware on which it is running is. Consequently, this collapses a major advantage of simulation—the ability to compress long time into a very shorter period [21].

3.1.3. Synchronization in our previous tool

In general, achieving synchronization between distributed simulators is a challenging issue. Therefore, in [4,1], we relaxed some constraints on the requirements the combined tool must meet. In particular, while the tool fully supports communication events between ns-2 and Modelica that depend on ns-2's internal events, those communication events depending on internal events inside Modelica are not supported.

To elaborate, consider the following example. Suppose that this tool is used to simulate the simplest form of a CPS consisting of a single physical system and a remote controller such as the one in Fig. 1. The sensor samples the values of physical quantities, writes them in a packet, and sends the packet over the network to the controller. The controller examines the received sample to generate a control signal that is then sent over the network to the actuator. When we first developed the tool, we based it on the legitimate assumptions that are often assumed in CPSs research realm: the plant is time driven (and most often, the sampling times are uniform) and that the controller is event driven and its computation time is negligible or constant; see for example [23] and the references therein. Due to such assumptions, the design methodology was to enslave Modelica by ns-2 in that ns-2 controls and determines *all* time instants at which the communication between Modelica and ns-2 should occur. So, in reference to the example of Fig. 1,

- The sampling events of the plant ought to be dealt with inside ns-2 not inside Modelica. The sampling intervals can be regular or irregular. However, the next sampling time cannot depend on some quantity or variable inside a Modelica model (e.g., sampling a physical quantity and sending a packet once a Modelica variable crosses some threshold).
- The computation delay of a controller is assumed to be zero. Then, when ns-2 delivers the packet carrying the sampled data to the controller modeled inside Modelica, ns-2 collects the output of the controller instantly. Notice that the tool can still support the case if the controller has a delay independent of any Modelica variable, for example, a known constant delay or a delay that changes based on a predefined trend. This delay should be "coded" inside ns-2 though.

That is, ns-2 determines on behalf of Modelica the time instances of packet transmissions originating from Modelica including sensor samples and controller commands. Notice that the

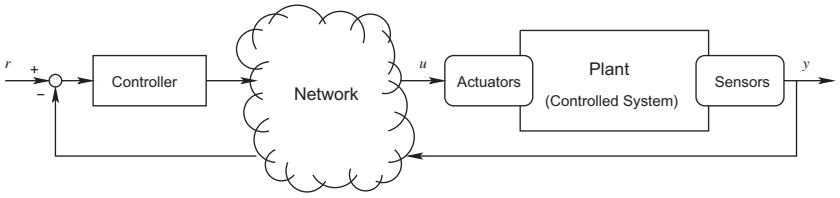

Fig. 1. A CPS with one controlled system (a.k.a. *plant*) and one controller. Both the sensor and the actuator are co-located at the plant site. Figure is adopted from [1].

opposite alternative, i.e., enslaving ns-2 inside Modelica, was ruled out because if Modelica were to control the communication events between the two simulators, the communication between Modelica and ns-2 could not depend on internal events inside ns-2. In turn, unpredictability and nondeterminisim of communication networks would be eliminated and unsupported by such design choice. In the adopted choice, on the other hand, the unpredictability of physical systems is unsupported. We opted to support nondeterminisim in networks over nondeterminisim in physical systems for the following two reasons.

1. The time instants of when to deliver data from Modelica to ns-2 (e.g., sampling a plant and collecting a control signal) are often assumed predictable and can be coded inside ns-2.
2. In general, the time instants of when data needs to be delivered from ns-2 to Modelica (e.g., arrival of packets) are unpredictable due to random delays, presence or absence of cross traffic, uncontrolled losses that the simulated packets incur.

To make the inter-simulator communication feasible, the following detailed steps are executed. It is realized that the flow of data between Modelica and ns-2 occurs in both directions: data flows from Modelica to ns-2 and vice versa. When data is to be sent from ns-2 to Modelica, we refer to this process by a write event (using the ns-2 side convention). Conversely, a read event is when data is sent from Modelica to ns-2; see Fig. 2.

1. Without loss of generality, we assume that both Modelica and ns-2 start from a common time, $t_i = t_0$.
2. While Modelica is pausing at t_i, ns-2 runs until the time of the first event that mandates communication with Modelica, $t_{i+1} \geqslant t_i$.
3. ns-2 executes the respective event, it pauses at t_{i+1}, and it instructs Modelica to run until t_{i+1}. In which case
   - If the event is a read operation, ns-2 instructs Modelica to write to the named pipe. Then, ns-2 reads the data.
   - If the event is a write operation, ns-2 instructs Modelica to read from a named pipe.
4. Steps 2 and 3 are then repeated with $t_i = t_{i+1}$ until the end of simulation. These steps are illustrated in Fig. 3.

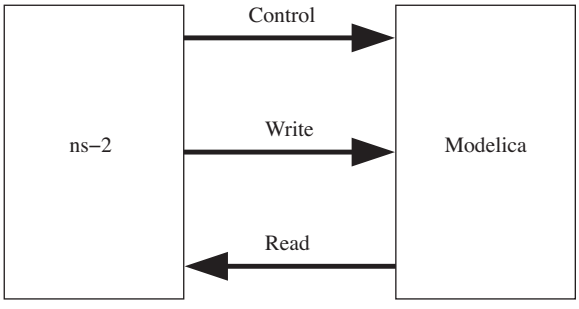

Fig. 2. The read and the write operations between Modelica and ns-2.

The simulation time of ns-2 is always leading that of Modelica. Note that when ns-2 is progressing in time (i.e., running), Modelica is pausing; and when Modelica is progressing, ns-2 is pausing. So, at a given time either ns-2 or Modelica is running and the other is pausing. With this mechanism, it can be thought that while Modelica is pausing, ns-2 is exploring its way searching for events that require communication with Modelica. Modelica and ns-2 pausing mechanism is achieved by blocking reading from an empty named pipe; see Fig. 2.

3.2. Limitation of the older version

Because ns-2 determines the communication times between the two simulators, Modelica cannot determine when to deliver data to ns-2 and it is that ns-2 that determines on behalf of Modelica when it should communicate with ns-2. Therefore, sending data between Modelica and ns-2 in response to events generated inside Modelica is not supported. That is, aperiodic control and alarm signals that are generated in response to events triggered exclusively inside the physical system (i.e., inside Modelica) are not accounted for. The same synchronization methodology we implemented in [1] was also adopted in [16]. On the contrary, in [17], the synchronization was achieved via enslaving ns-2 by Simulink®.

Although the tool in [1] provided a very general tool that is sufficient to simulate myriad of real-life setups, there is still a need for a more comprehensive tool that addresses all such limitations.

4. A new comprehensive tool

In this paper, we improve the previous tool [1] by giving it the ability to respond to internal asynchronous events inside Modelica, for example, sending of packets that are not necessarily deterministic from the ns-2 point of view. In this section, we present the implementation details and the enhancements of this tool. First, ns-2 is an event-driven simulator where it keeps future events in an appropriate data structure, which is called the Scheduler. Additionally, ns-2 exposes well-defined APIs to access and possibly to manipulate the Scheduler, e.g., obtaining the next future event and deleting and inserting new events manually.

4.1. A preliminary observation

Suppose that, similar to ns-2, we have also access to Modelica's Scheduler. The obvious solution would have been as follows.

1. We start with both simulators pausing at a common time t_i.
2. We consult both simulators' Schedulers, to retrieve the time, t_f^n, of the earliest future event among all events already scheduled inside ns-2, and the time, t_f^M, of the earliest future event inside Modelica.
3. During the time interval $\left[t_i, \min\left(t_f^n, t_f^M\right)\right)$, we are certain that the two simulators need not to communicate. So, we let both simulators run freely until $t_{i+1} = \min\left(t_f^n, t_f^M\right)$ and pause there.

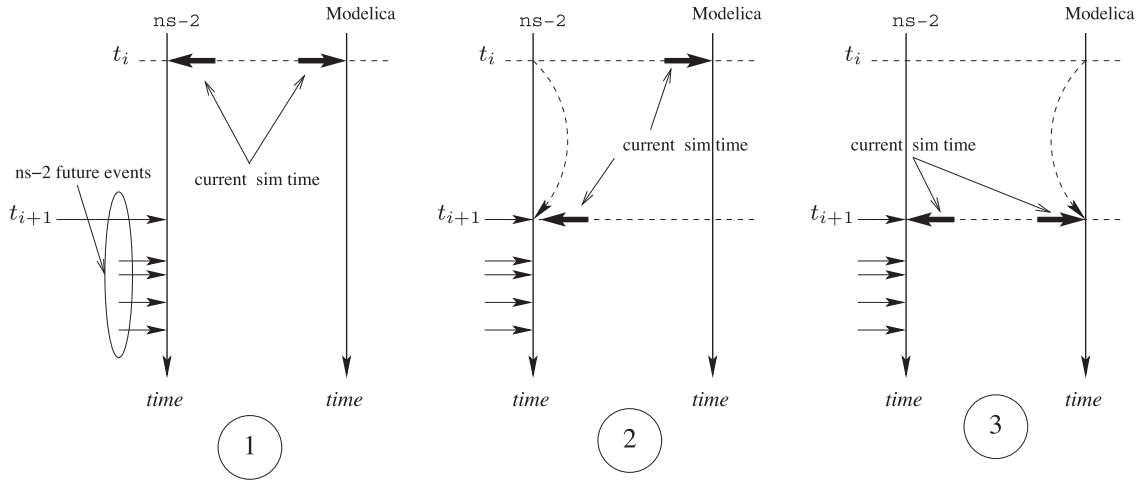

Fig. 3. Illustration of Modelica and `ns-2` pausing and progressing steps. Circled numbers correspond to steps in text.

4. We let the simulator having the earliest future event execute its event only and pause there afterward.
5. If such event requires one simulator to deliver or obtain data from the other, we facilitate their communication through the named pipe. Both simulators are now pausing at t_{i+1} again.
6. We repeat steps 2–5 until the end of simulation with $t_i = t_{i+1}$. Note that after processing the event at $t_{i+1} = \min\left(t_f^n, t_f^M\right)$, we next execute step 2 and do not progress to time $t' = \max\left(t_f^n, t_f^M\right)$ because it might happen that the event that we just processed (i.e., at t_{i+1}) has triggered new events inside one or both simulators during the interval $\left[\min\left(t_f^n, t_f^M\right), \max\left(t_f^n, t_f^M\right)\right]$. These steps are illustrated in Fig. 4.

Note that updating the Schedulers in both simulators occurs implicitly without our intervention. We only intervene in the stopping and restarting process.

The only obstruction to realize this neat solution is that we have no internal access to Dymola, the Modelica simulation environment we are using, and its Scheduler. In the next subsection, we explain how we work around this problem. The synchronization mechanism that is just mentioned was implemented in [9] where access to both `ns-2` and `adevs` Schedulers is feasible.

4.2. Synchronization in the new tool

Because we have access only to `ns-2`'s Scheduler, we enslave Modelica with `ns-2` as was the case in our older tool. However, instead of allowing `ns-2` to advance to the imminent event where the two simulators have to communicate and then to instruct Modelica to progress to that event time, we attain the synchronization as follows.

1. We start with both simulators pausing at a common time t_i.
2. We consult the `ns-2`' Scheduler, to retrieve the time, t_f^n, of the earliest future event.
3. While `ns-2` is pausing at t_i, Modelica is instructed to progress to t_f^n. Then, there arise two cases:
   - During the time interval $\left[t_i, t_f^n\right)$, Modelica encounters no events that require communication with `ns-2`. Thus, Modelica reaches t_f^n. Thereupon, `ns-2` proceeds to time $t_i = t_f^n$, executes the respective event, and pauses there.

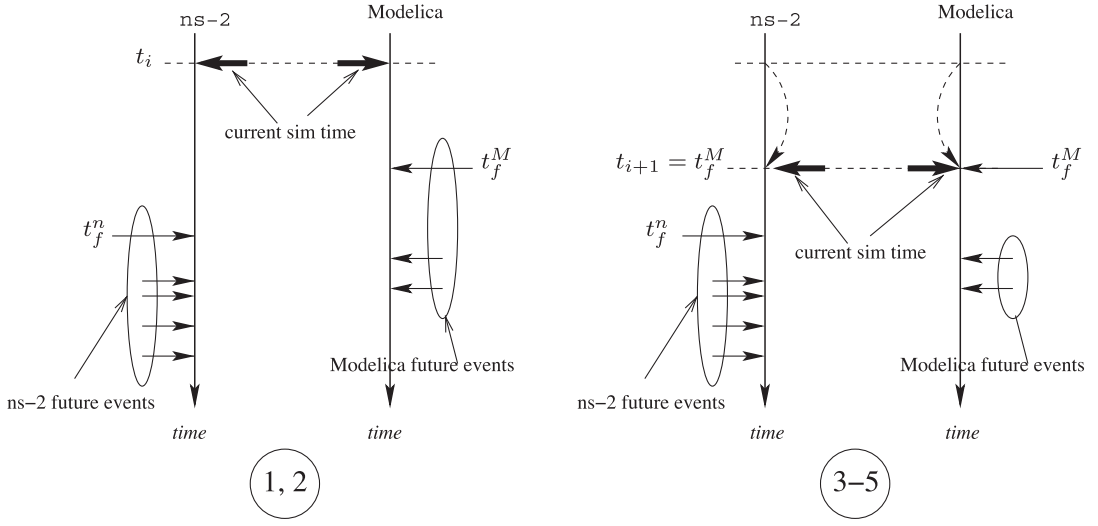

Fig. 4. Illustration of Modelica and `ns-2` pausing and progressing steps. Circled numbers correspond to steps in text.

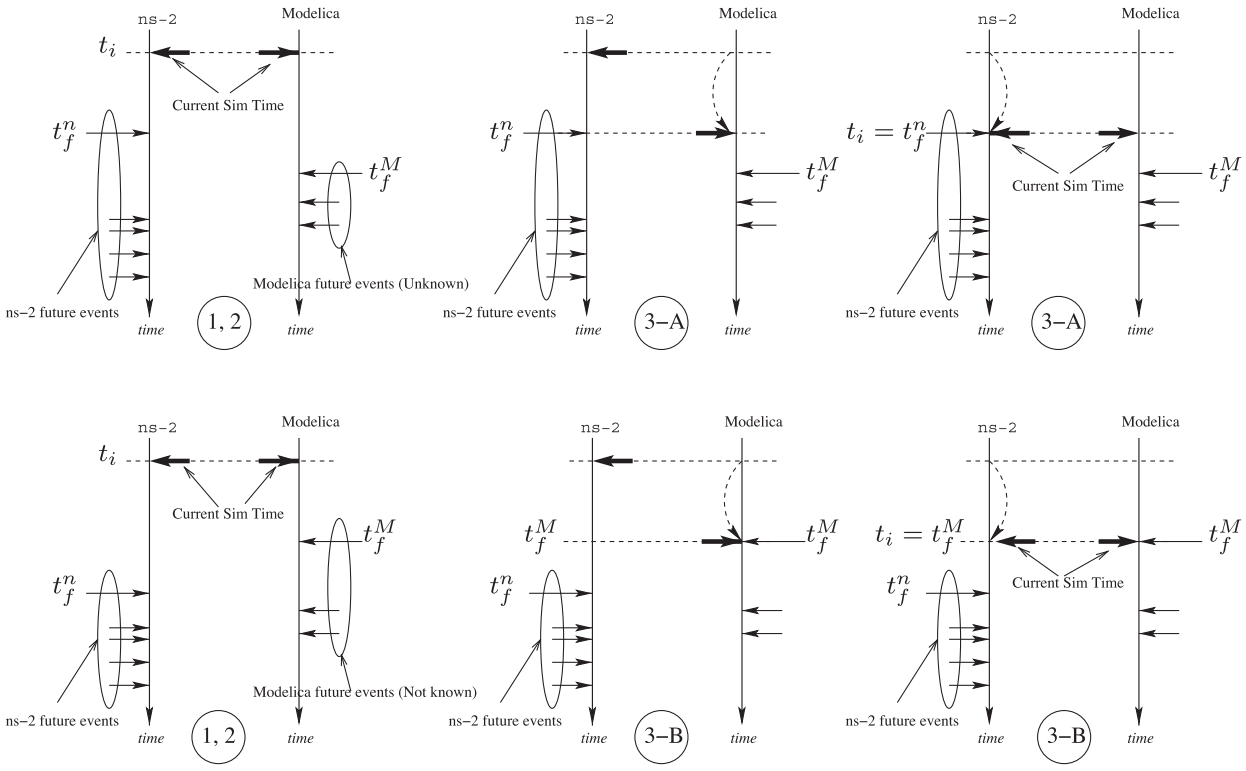

Fig. 5. Illustration of Modelica and `ns-2` pausing and progressing steps. Circled numbers correspond to steps in text.

- During the time interval $\left[t_i, t_f^n\right)$, Modelica has to communicate with `ns-2` at least once. Modelica runs until the time t^M where it first needs to deliver data to `ns-2`, it executes this event, and it pauses there $\left(t_i \leqslant t^M \leqslant t_f^n\right)$. Then, `ns-2` proceeds to $t_i = t^M$, obtains data from Modelica, and pauses at t_i.

In both cases, Modelica informs `ns-2` whether it progressed to t_f^n without interruptions or it paused at $t^M \in \left[t_i, t_f^n\right)$ for an event requiring communication with `ns-2`.

4. We repeat steps 2 and 3 until the end of simulation. These steps are illustrated in Fig. 5.

This scheme will not have any problem of missing any events inside both Modelica and `ns-2` because of two reasons:

- We always pause at *every* `ns-2` event assuming it will require communication with Modelica.
- Modelica runs ahead of `ns-2` and it also pauses when executing events that require communication with `ns-2`; see Fig. 5. So, Modelica explores its way searching for events that require communication with `ns-2`. When those events occur, Modelica pauses and then `ns-2` follows.

Having access to `ns-2`'s Scheduler has eliminated the need to use other highly *expensive* mechanisms such as the one in [24]. A synchronization solution based on [24] can be employed to support asynchronous events within Modelica. Such solution follows along the same lines as the mechanism that was employed in the older version with the following modification (refer to steps 1–4 in Section 3.1.3 and Fig. 3). In step 3, if Modelica encounters an internal event that requires communication with `ns-2` at time $t^M \in [t_i, t_{i+1})$, Modelica pauses at t^M. The state of `ns-2` is *rolled back* to t_i and is then *re-executed* from t_i to t^M. The process then repeats. This solution suffers from the following disadvantages. First, the implementation of *state saving*, *state restoration*, and *rollback and re-execution* in `ns-2` can be complicated. Second, the overhead of rollback and re-execution can be substantial. On the other hand, our approach represents an elegant solution by not having `ns-2` to run ahead of Modelica.

4.3. Implementation of the synchronization

We have implemented the steps above inside `ns-2` and Modelica. Inside Modelica, We created a new model, called `Network`, that embodies the necessary functionalities of pausing Modelica's simulation time at a specific time instant, progressing Modelica's simulation time to a given prescribed time instant, and communicating with `ns-2` to receive commands and to deliver and receive data. The `Network` model is constructed as a MIMO (multiple inputs and multiple outputs) block diagram. The data

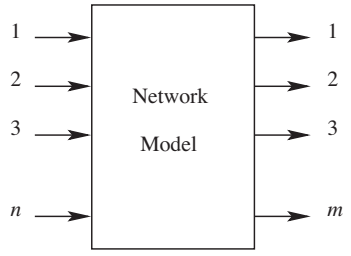

Fig. 6. The abstract view (i.e., the interface) of the `Network` model inside Modelica.

that is to be transported via network is fed into an input port; whereas, the data that is to be received from network is connected to an output port; see Fig. 6. As for the `ns-2` side, we created a new derived class that inherits from the calendar's scheduler class [20]. The new class, called `CPSsim`, adapts `ns-2`'s default scheduler to control the simulation steps of the combined tool (i.e., the above pause-progress mechanism). Also, we created an application-level `ns-2` agent, called `PhySys`, that can take a role of any physical world's entity, e.g., a plant, a sensor, and actuator, or a controller. The mapping between Modelica and `ns-2`, i.e., between `Network` inputs and outputs and `PhySys` instances, is accomplished via common ID numbers. Again, the communication between Modelica and `ns-2` is facilitated via named pipes.

5. Validation of the new tool

In this section, we present the techniques that we followed to make sure that the newly developed tool is working correctly. This is a critical step because the tool combines two different and relatively large simulation environments that must work not only in harmony but in a *correct* manner. Although several books differentiate between the meanings of the two terms *validation* and *verification* [21,25], we here use the two words interchangeably to refer to the process of making sure the developed tool is working correctly.

5.1. Extensive testing and debugging

Throughout the development process of the tool, we made sure to verify the correctness of every submodule's functionality. While developing individual Modelica and `ns-2` modules and functions, we developed dummy programs that would interact with Modelica or with `ns-2` as it were the other party. In addition, we generated synthetic inputs to feed into Modelica or `ns-2` and we verified the correct operation of the developed modules by comparing the traces they generate to the traces generated by carrying out manual hand simulations.

5.2. Formal validation

Although the above steps are necessary to finally produce a credible tool, they are not sufficient. Here, we validate our tool by comparing the operation of the complete tool to an analytical solution and to an output produced by a 'pure' Modelica's model for the same simulation example. We emphasize that the intent here is not to show the capabilities of the tool; rather, it is to verify the tool's correctness.

5.2.1. Setup

We consider the following network topology and parameters for the simulation example; see Fig. 7. The plant and the controller are interconnected via an intermediate router. The two links connecting the plant and the router and the controller and the router are identical with each having capacity of 1 Mbps and propagation delay of 1.2 ms. The plant and the controller exchange the same packet sizes of 100 bytes each.

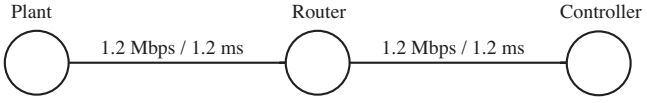

Fig. 7. Network topology for the evaluation simulation example.

The plant is a first-order system [26] whose state $x(t)$, input $u(t)$ and output $y(t)$ are governed by

$$\begin{cases} \dot{x}(t) = ax(t) + bu(t) \\ y(t) = cx(t). \end{cases} \tag{1}$$

Moreover, the control law is a pure proportional controller with gain k, i.e., $k(r(t) - y(t_k))$, where $y(t_k)$ is the sampled plant output at time t_k and $r(t)$ is the reference input that the plant is required to follow. We choose the following specific values for the parameters above: $a = -1$, $b = 1$, $c = 1$, and $k = 2$. The plant is sampled every $h = 0.01$ s. and is excited with a sinusoidal reference input, $r(t)$, having amplitude 1, frequency 2 Hz, and phase 0.0 rad. The simulation lasts for 2.5 s. Although this example might seem simple, it contains all types of events that would unveil any possible erroneous operation of the tool, including multiple simultaneous events occurring inside Modelica, multiple simultaneous events occurring inside `ns-2`, and multiple simultaneous events occurring inside Modelica and `ns-2` at once. On the other hand, such example enjoys two features: it has a closed-form analytical solution, and it fits within the capabilities and constraints of a physical systems simulation environment, e.g., Modelica.

5.2.2. Analytical solution

Since the incorporation of the network in feedback loop introduces time delays (1) becomes (note that $c = 1$)

$$\begin{cases} \dot{x}(t) = ax(t) + bu(t - \tau_{cp}) \\ u(t) = k(r(t) - x(t - \tau_{pc})), \end{cases} \tag{2}$$

where τ_{cp} and τ_{pc} are the plant-to-controller and the controller-to-plant one-way delays, respectively. Combining the two equations in (2) yields

$$\dot{x}(t) = ax(t) + bk(r(t - \tau) - x(t - \tau)), \tag{3}$$

where $\tau = \tau_{pc} + \tau_{cp}$ is the round-trip delay between the plant and controller. Note that reference input $r(t)$ has been deliberately shifted in time by amount of τ_{pc} to facilitate the analysis. Also, note that τ in our experiment setup is constant and equals 8 ms. The sampled system of (3) at the sampling times, t_k, is given by [27]

$$\dot{x}(t_{k+1}) = e^{ah}x(t_k) + \frac{b}{a}(e^{a(h-\tau)} - 1)u(t_k) + \frac{b}{a}(e^{ah} - e^{a(h-\tau)})u(t_{k-1}), \tag{4}$$

where $u(t_k) = k(r(t_k) - x(t_k))$.

5.2.3. Models

We constructed the models of the networked system given in (1) using our tool and Modelica; see Fig. 8. In Modelica, the network is modeled using two fixed-delay modules `P2CDelay` and `C2PDelay`. Again, note that the one-way delay form the plant to the controller is 4 ms and it is the same for the delay from the controller to the plant. The deliberate delay, τ_{pc}, that is introduced in $r(t)$ in (3) is accomplished by introducing a lagging phase shift in the sinusoidal input of amount $2\pi f\tau_{pc}$, where f is the frequency of $r(t)$, i.e., 2 Hz.

5.2.4. Comparison

Fig. 9 shows the plant response, $y(t)$, as a function of time for the model built using our tool, for the analytical solution according to (4), and for the model built in Modelica. Although it seems that the plant response in the three cases is identical, a closer look reveals that there is an error between the one based on our tool and the analytical one; see Fig. 10. Because the error is extremely small (less than 1×10^{-8}) and it occurs only for few points, we assert

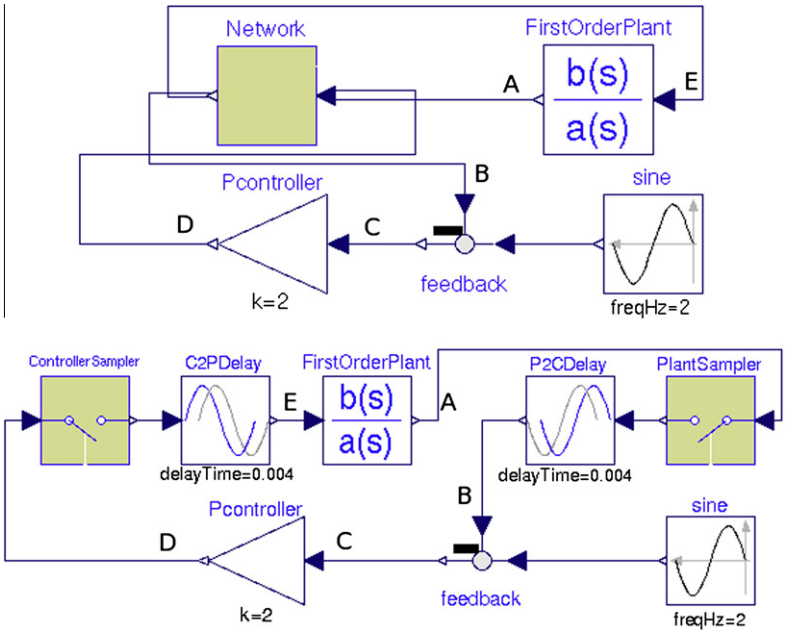

Fig. 8. The two models corresponding to the networked system given in (1) using our tool (above) and Modelica (below).

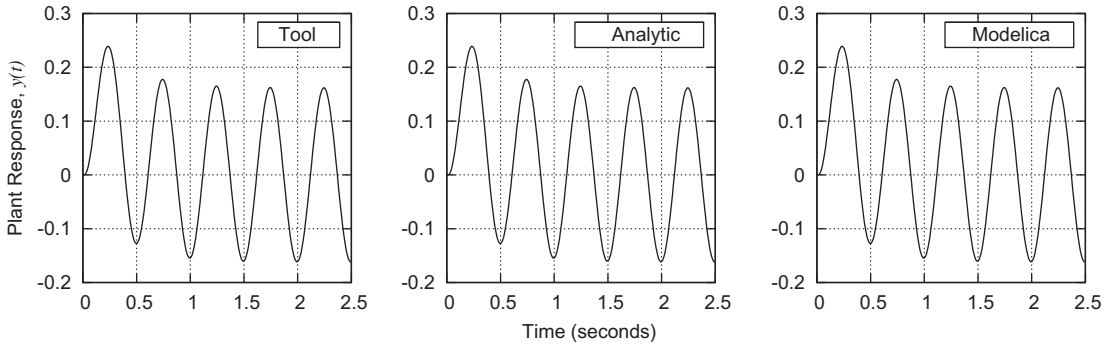

Fig. 9. The plant response, $y(t)$, as a function of time for the model built using our tool (left), for the analytical solution according to (4) (middle), and for the model built in Modelica (right).

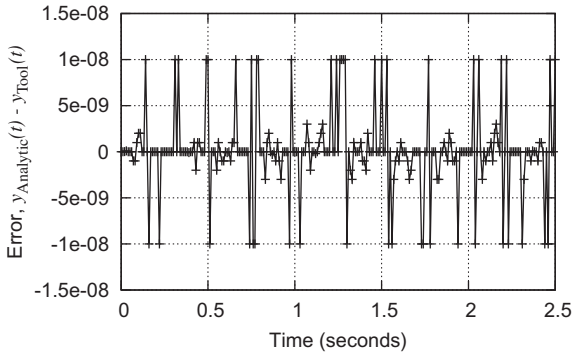

Fig. 10. The error between the analytical solution and the output from the model built using our tool.

it is a mere rounding error and is not introduced by the tool. To justify this, we also ascertain the error between the output produced with our tool and that produced using the Modelica-based model not only for the response $y(t)$ but also for the five points, A to E, shown in Fig. 8. We indeed found that the error in the response between every two corresponding points is identical to zero, which strongly confirms our assertion. Therefore, we hold an extreme confidence about the credibility of the newly developed tool.

6. The new tool at work

In this section, we present examples illustrating the capabilities and the features of the tool. We will show some excerpts of the required `ns-2` code, figures of constructed Modelica models, and the corresponding simulation results.

6.1. *ns-2 code*

The following code fragment demonstrates the steps to craft a network simulation script in `ns-2`. The code elaborates exclusively on the steps relevant to our tool, such as instantiating CPS objects, configuring them, and linking them to models inside Modelica. Code for other steps that are typical in any `ns-2` code is suppressed for brevity, e.g., building a network topology and configuring its parameters. The code is made self-explanatory by inserting enough comments, i.e., the lines starting with #.

```
# 1. Create an instance of the Simulator Class.
set ns [new Simulator]

# 2. Use the newly developed Scheduler that is specific
# for CPS simulations.
$ns use-scheduler Calendar/CPSsim

# 3. Create network nodes; connect nodes via wireline or
# wireless links with desired parameters to form some
# topology; and create transport-layer protocols (e.g.,
# UDP, RTP, or TCP) with required parameters.

# 4. Create a CPS instance that takes a role of a plant.
# Note that the actual functionality will be implemented
# inside Modelica. So, the 'plant' here serves only as
# an object that relies data between ns-2 and Modelica
# but has no physical-systems functionality.
set plant [new Application/PhySys]

# 5. Tie/map the plant application agent to
# corresponding input and output ports of the 'Network'
# module inside Modelica.
$ns ports-ids $plant 1 2

# 6. Associate a transport-layer protocol with the
# plant. Here, $transport is UDP agent that was created
# and tied to a given node; see step 3 above.
$plant attach-agent $transport

# 7. Configure the plant's packets sizes.
plant set pktsize_ 100

# 8. Schedule some events.
$ns at 2.50 "finish"
$ns run
```

6.2. Example 1: an industrial automation system

The first example we present is a controlled flexible drive system; see Fig. 11. This example arises as an adequate model for several industrial automation systems, such as, assembly line robots, conveyor belts, and other systems embedding rotating parts. Since contemporary industrial automation systems practices depend on distributed control, we assume that the plant is controlled by two remote controllers as in Fig. 11. Communication between the two controllers and the plant occurs over a switched Ethernet LAN, which represents a point-to-point communication medium. Such setup reflects the fact that the two controllers and the plant are located in different rooms, floors, or even buildings. The network topology and parameters are shown in Fig. 12. The bandwidth of the link connecting the two switches is intentionally chosen as a scaled down value of that of real-life Ethernet links, e.g., 100 Mbps or 1 Gbps, to highlight the effect of sampling on CPSs operation. Other than this, other network parameters are chosen to reflect real-life settings.

The angle measurements are transported over the network to the proportional (P) controller. The output from the P controller and the speed measurements are transferred over the network to the proportional-integral (PI) controller. Finally, the output of the PI controller is fed back via the network into the plant and is used to generate torque to drive a tandem of two inertia units. As shown in Fig. 12, the angle measurements and the speed measurements are multiplexed onto the same subset of links, while the P-to-PI packet flow and the speed measurements are multiplexed onto the same link. Although there is one control loop, these multiple measurements and control flows can be regarded as multiple control loops that are closed over the network; see the Results paragraph below. The objective in this whole system is to control the load inertia to rotate in a constant angular speed of 2.0 rad/s during the period [2,22] s.

Results. Fig. 13 shows the effect of sampling of the angle and the speed measurements on the plant's performance and stability. When both quantities are sampled at the same uniform sampling rate of 1000 sample/s, the system goes out of control. This occurs because the packets carrying both measurements traverse the same bottleneck link connecting both switches. At a sampling rate of 1000 sample/s, packets form the two flows congest the link and thus lead to increased delays and packet losses (results are not shown). In turn, these two factors caused the instability. On the other hand, by reducing only the sampling rate of the angle measurements to 250 sample/s and keeping the speed's sampling rate at 1000 sample/s, the system is brought into stability and it is performing well; see Fig. 13. Although the problem of congestion has been solved in this example statically, in [15], we proposed a dynamic congestion control scheme that allocates the bandwidth among several

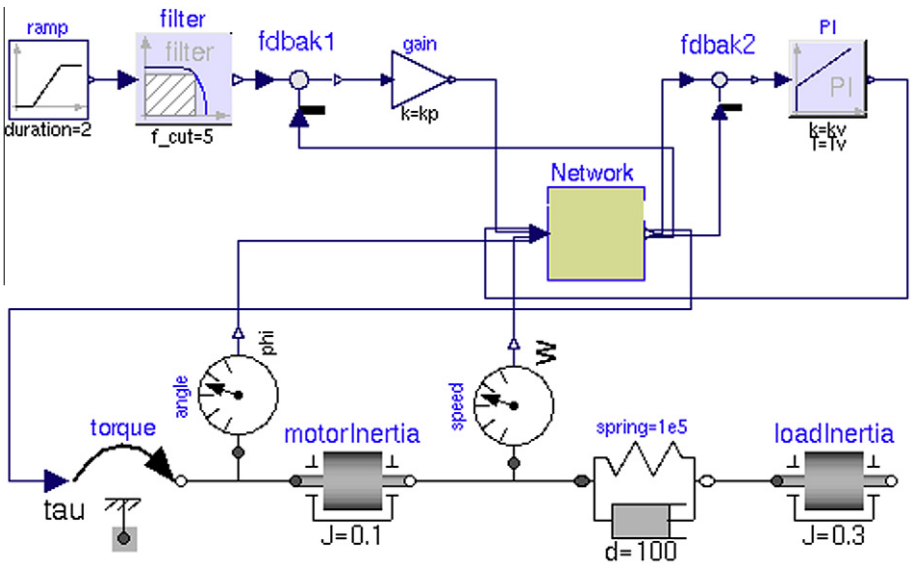

Fig. 11. A CPS example: flexible drive system.

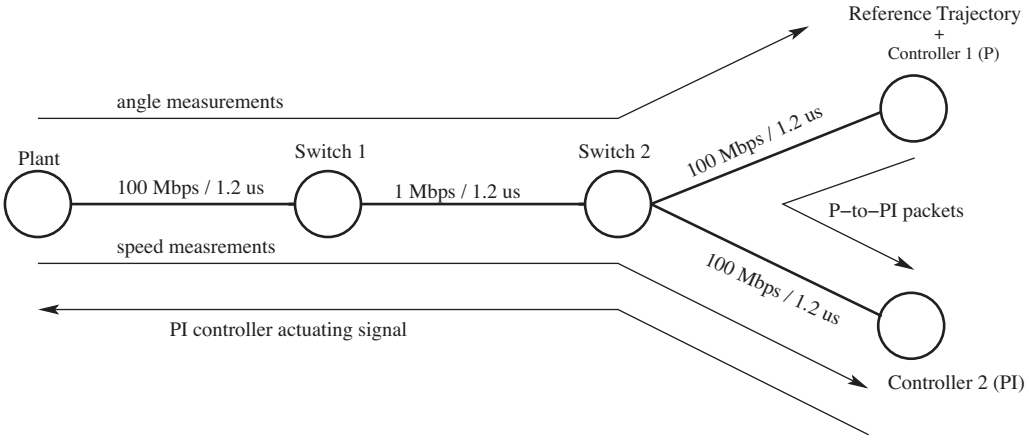

Fig. 12. The network topology and links parameters corresponding to Fig. 11.

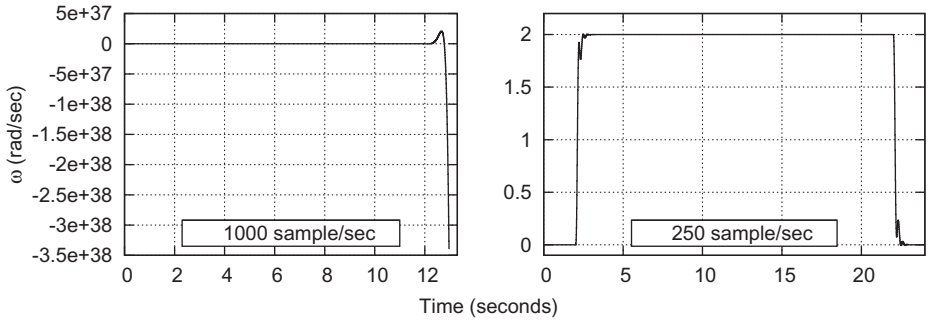

Fig. 13. Effect of sampling of both the angle and the speed measurements on the plant's performance and stability: when both quantities are sampled at 1000 sample/s (left); and when the angle's sampling rate is 250 sample/s (right). Note that in the first case, the plant response is shown only during the interval [0,12.5] sec.

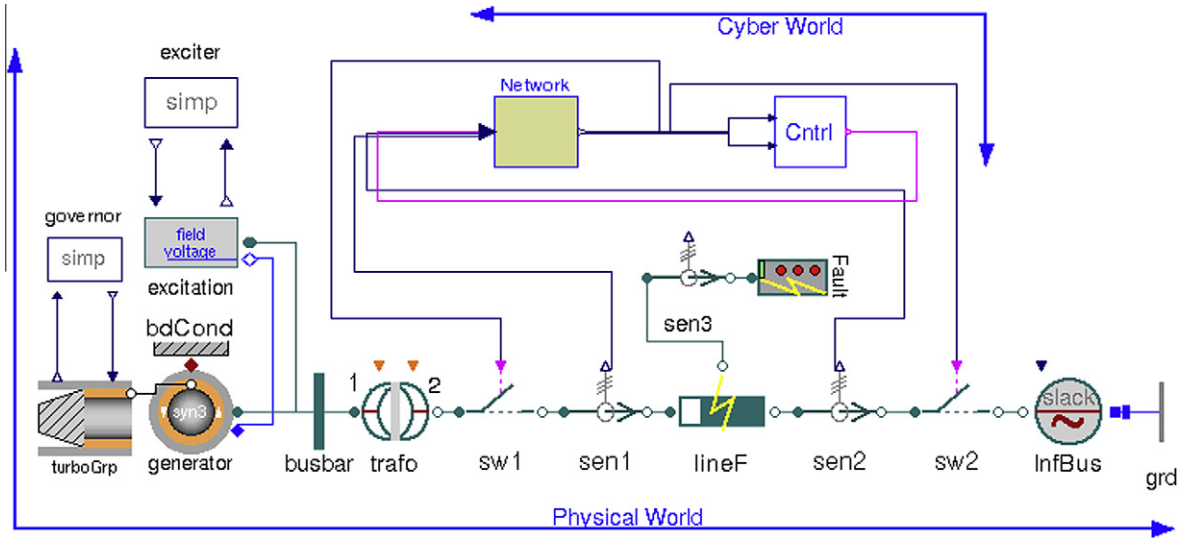

Fig. 14. A CPS example: electric power transmission system.

competing CPSs control loops efficiently. Also, the effect of sharing the same network infrastructure between control loops and data flows was studied in [1].

6.3. Example 2: a smart power grid

The second example we present is a simple power system; see Fig. 14. This example stresses the necessity for augmenting the power grid with smartness by arming it with the ability to sense, infer about itself and to self react and self heal in face of faults.

The power grid is the largest physical system the man have ever engineered [28]. Due to its indispensable role in modern society, it must be dependable and reliable, efficient, and fault tolerant. There have recently been special focus and enormous efforts to make the grid smart.

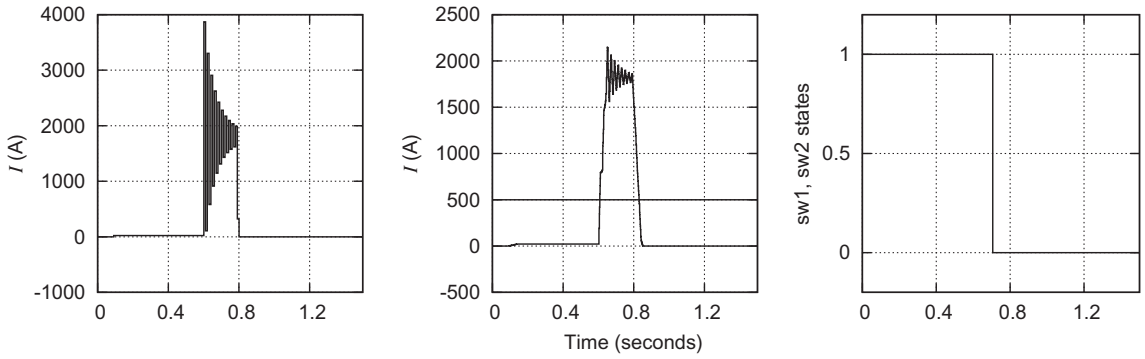

Fig. 15. The absolute difference between current measurements at both ends of the line (left). The time-averaged of the absolute difference between current measurements at both ends of the line, and the fault-detection threshold (middle). The fired fault signal: 0 means the switch is open (right).

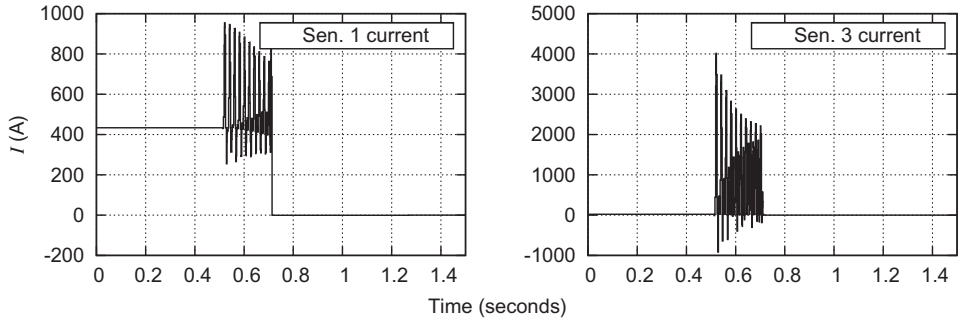

Fig. 16. The single-phase current of the faulty line measured at the near terminal to the generator (left). The short circuit single-phase current (right).

In our specific example, we show how a simple sensor-actuator network can detect and isolate a fault in a power transmission line. The power system consists of a steam or water turbine generator and a transmission line connecting the generator via a transformer to an infinite slack bus. The line is 430 km long. The currents at both ends of the line are measured and are transported over a WAN to a control logic. The control logic compares the measurements of the three-phase currents and produces appropriate signals to open the circuit breaker switches at both ends of the line if a fault is detected. The detection algorithm is based on the time-averaged difference between the corresponding measurements at both ends of the line: if the low-pass filtered difference exceeds a given predefined threshold, which is 500 A in our case, then a fault signal is fired [29]. Such asynchronous control signals are also transported over the WAN. This scenario represents a system where the control logic is not located near any end of the transmission line. We assume that all communications take place over a power line broadband network (BPL) [30] with all links have bandwidth of 1 Mbps and propagation delays of 80 ms.

Results. At simulation time 0.5 s, a single-phase short circuit occurs. Before the fault, the absolute difference between the current measurements at both ends of the line is near zero; and is hence the time-averaged value too; see Fig. 15. In this case, the circuit breaker switch is closed. Once the fault strikes, most of the current flowing from the right end does not reach the other end, and thus, the absolute difference widens and its low-pass filtered value tracks the increase in the absolute difference between the current measurements at both ends. Once the time-averaged value exceeds the threshold, the controller opens the two switches at both ends and the fault is contained effectively. This whole scenario is illustrated in Figs. 15 and 16.

With aid of the sensor-actuator network, the line fault has been quickly and accurately detected and isolated. This accuracy is highly desirable in modern power systems. Other schemes completely eliminate the need of communication, rely only on one-end measurements, and employ artificial neural networks to infer line faults. However, such schemes attain less accuracy and suffer from nonnegligible false positive and/or false negative rates [31].

6.4. Capabilities and versatility of the tool

All the sampling events in the previous two examples are "coded" inside the corresponding Modelica models. Additionally, the asynchronous 'open' switch signal in Example 2 depended solely on a Modelica variable that crossed a threshold; see Fig. 15. These functionalities were not possible in the older platform.

Based on our architecture, the two simulation environments, `ns-2` and Modelica are loosely coupled where they communicate via the named pipes. Therefore, changing some models in one side does not affect the other as long as the common ID numbers do not change; see end of Section 4. For example, the network topology and parameters can be modified without affecting the Modelica models. Also, changing the wireline network to a wireless network will not affect the Modelica side. Likewise, changing or even replacing Modelica models does not affect the associated `ns-2` scripts. All of this is valid as long as the mappings between Modelica and `ns-2` are preserved. Therefore, one can build the needed physical models inside Modelica and then overlay the network atop the physical system, or vice versa.

7. Conclusion

In this paper, we have presented a comprehensive co-simulation platform for cyber-physical systems. After elaborating thoroughly on several design alternatives, we validated the newly

developed tool. Moreover, we presented two examples to demonstrate the features of the tool: one is in the domain of industrial automation and another relates to the smart grid. Unlike previous studies, we covered the issue of synchronization in much depth suspecting it may prove beneficial in future tools that combine different simulator environments. Finally, we emphasize that we focused here on the more general and the harder case of CPSs where the platform has to address a network of sensors and actuators. However, the tool can be readily applicable to simulate a network of sensors only where no synchronization between Modelica and `ns-2` is required.

Acknowledgments

I would like to thank the three anonymous reviewers for their insightful and constructive comments that greatly enhanced the clarity of this paper.

References

[1] A.T. Al-Hammouri, M.S. Branicky, V. Liberatore, Co-simulation tools for networked control systems, in: International Conference on Hybrid Systems: Computation and Control (HSCC 08), St. Louis, MO, USA, 2008.
[2] H. Szczerbicka, K.S. Trivedi, P.K. Choudhary, Discrete event simulation with application to computer communication systems performance, in: Information Technology, vol. 157, 2004, pp. 271–304.
[3] V. Liberatore, A.T. Al-Hammouri, Networked control of the smart grid, in: NITRD Workshop on New Research Directions for Future Cyber-Physical Energy Systems, Baltimore, Maryland, USA, 2009, available at <http://www.ece.cmu.edu/nsf-cps/file.php?id=54>.
[4] A.T. Al-Hammouri, V. Liberatore, H. Al-Omari, Z. Al-Qudah, M.S. Branicky, D. Agrawal, A co-simulation platform for actuator networks, in: Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys 07), Sydney, Australia, November 2007. Demonstration.
[5] C.Y. Baldwin, K.B. Clark, Design Rules: The Power of Modularity, The MIT Press, 2000.
[6] K. Astrom, B. Bernhardsson, Comparison of Riemann and Lebesgue sampling for first order stochastic systems, in: Proceedings of the IEEE Conference on Decision and Control, 2002.
[7] T. Henningsson, E. Johannesson, A. Cervin, Sporadic event-based control of first-order linear stochastic systems, Automatica 44 (11) (2008) 2890–2895.
[8] A. Cervin, M. Ohlin, D. Henriksson, Simulation of networked control systems using TrueTime, in: Proceedings International Workshop on Networked Control Systems: Tolerant to Faults, Nancy, France, 2007.
[9] J. Nutaro, P.T. Kuruganti, L. Miller, S. Mullen, M. Shankar, Integrated hybrid-simulation of electric power and communications systems, in: IEEE Power Engineering Society General Meeting, 2007, pp. 1–8.
[10] M.S. Branicky, V. Liberatore, S.M. Phillips, Networked control system co-simulation for co-design, in: Proceedings American Control Conference, Denver, 2003.
[11] P. Reuterswärd, J. Åkesson, A. Cervin, K.-E. Årzén, TrueTime Network—A network simulation library for Modelica, in: International Modelica Conference, Modelica Association, 2009.
[12] D. Henriksson, H. Elmqvist, Cyber-physical systems modeling and simulation with modelica, in: International Modelica Conference, Modelica Association, 2011.
[13] F. Wagner, L. Liu, G. Frey, Simulation of distributed automation systems in modelica, in: International Modelica Conference, Modelica Association, 2008.
[14] P. Baldwin, S. Kohli, E.A. Lee, X. Liu, Y. Zhao, Modeling of sensor nets in Ptolemy II, in: Proceedings of Information Processing in Sensor Networks (IPSN), Berkeley, CA, 2004.
[15] A.T. Al-Hammouri, M.S. Branicky, V. Liberatore, S.M. Phillips, Decentralized and dynamic bandwidth allocation in networked control systems, in: Proceedings of International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 06), Island of Rhodes, Greece, 2006.
[16] M.S. Hasan, H. Yu, A. Griffiths, T.C. Yang, Co-simulation framework for networked control systems over multi-hop mobile ad-hoc networks, in: IFAC World Congress, the International Federation of Automatic Control, Seoul, Korea, 2008.
[17] T. Kohtamäki, M. Pohjola, J. Brand, L.M. Eriksson, Piccsim toolchain—design, simulation and automatic implementation of wireless networked control systems, in: Proceedings of the 2009 IEEE International Conference on Networking, Sensing and Control, Okayama, Japan, 2009.
[18] The it ns Manual, Online. Available: <http://www.isi.edu/nsnam/ns/>.
[19] Modelica and the Modelica Association, Online. Available: <http://www.modelica.org/>.
[20] The Network Simulator—`ns-2`, Online. Available: <http://www.isi.edu/nsnam/ns/doc/index.html>.
[21] J. Banks, J. Carson, B.L. Nelson, D. Nicol, Discrete-Event System Simulation, Pearson Prentice Hall, 2005.
[22] V. Liberatore, Integrated play-back, sensing, and networked control, in: Proceedings of IEEE INFOCOM, 2006.
[23] W. Zhang, M. Branicky, S. Phillips, Stability of networked control systems, IEEE Control Systems Magazine 21 (1) (2001) 84–99.
[24] S. Yoo, K. Choi, Synchronization overhead reduction in timed cosimulation, in: IEEE International High Level Design Validation and Test Workshop, 1997.
[25] A.M. Law, W.D. Kelton, Simulation Modeling and Analysis, McGraw-Hill, 2000.
[26] K. Ogata, Modern Control Engineering, Pearson Prentice Hall, 2010.
[27] K.J. Astrom, B. Wittenmark, Computer-Controlled Systems: Theory and Design, Prentice Hall, 1996.
[28] R. Rajkumar, I. Lee, L. Sha, J. Stankovic, Cyber physical systems: the next computing revolution, in: Design Automation Conference, 2010.
[29] M. Kezunovic, B. Perunicic, Automated transmission line fault analysis using synchronized sampling at two ends, IEEE Transactions on Power Systems 11 (1) (1996) 441–447.
[30] M. Reardon, CNET News, Earthlink, GE fund broadband over power line provider, Online. Available Via: <http://news.cnet.com/>.
[31] N. Zhang, M. Kezunovic, Transmission line boundary protection using wavelet transform and neural network, IEEE Transactions on Power Delivery 22 (2) (2007) 859–869.