Simulation Modelling Practice and Theory 23 (2012) 71–86

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

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

A SystemC/Matlab co-simulation tool for networked control systems

Davide Quaglia, Riccardo Muradore *, Roberto Bragantini, Paolo Fiorini

Department of Computer Science, University of Verona, Strada le Grazie 15, 37134 Verona, Italy

ARTICLE INFO

Article history:
Received 25 February 2011
Received in revised form 29 December 2011
Accepted 12 January 2012
Available online 8 February 2012

Keywords:
Co-simulation
Network simulation
Networked control systems
SystemC

ABSTRACT

Real-time systems connected through packet networks belong to the family of networked control systems, and they can be easily destabilized by communication delay and packet losses, when they are not properly compensated. The largest part of the solutions available in the literature are mainly based on control and system theory where the parameters of the network are assumed to be given. This classical approach could be improved by designing at the same time the network, e.g., by introducing quality-of-service guarantees as currently done in teleconference applications. Such control/network co-design needs a simulation framework where both aspects are properly and jointly addressed. The paper addresses this topic starting from the discussion of its critical issues, and then proposing an accurate co-simulation tool based on SystemC and Matlab/Simulink. SystemC will be used for the network simulation and protocol design whereas Matlab/Simulink for plant modeling and control design.

1. Introduction

In recent years, embedded, distributed and pervasive systems have gained a significant importance in a number of applications which include exploration, industrial automation, service robotics, remotely operated vehicles, and telemedicine. These applications go under the general name of Networked Control Systems (NCSs), i.e., feedback control systems in which the control loop is closed through a shared digital communication network rather than by an ideal point-to-point connection [1]. In NCS, the communication channel can significantly affect the quality of the control, as in case of the widespread TCP/IP architecture which is however of great interest for a broad range of applications.

NCS performance is mainly affected by two problems: communication delay and packet loss. The first problem is well known and it has been addresses by several authors. With respect to teleoperation systems, for example, algorithms have been proposed based on impedance control [2,3], passivity theory [4], wave variables [5], and linear observer. Robust control theory has been applied in [6], and the use of variables that are not explicitly time-dependent, so that they are not affected by delay, has been proposed in [7]. Several authors tried to predict the delay of a sequence of packets through a mathematical model of the delay of an Internet connection [8–11]. However, this approach can be used, in practice, only for short Internet segments, and it is not well suited for developing general delay models [12].

The simulation of NCS plays a crucial role in the verification, validation and fine-tuning of many of these solutions. A simulator should capture and represent both the control and communication aspects. For instance, the control aspects include signal generation and analysis, and plant/controller specification, whereas the communication aspects include channel and protocol specification, and packet flow generation.

* Corresponding author.
E-mail addresses: davide.quaglia@univr.it (D. Quaglia), riccardo.muradore@univr.it (R. Muradore), roberto.bragantini@edalab.it (R. Bragantini), paolo.fiorini@univr.it (P. Fiorini).

1569-190X/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2012.01.003

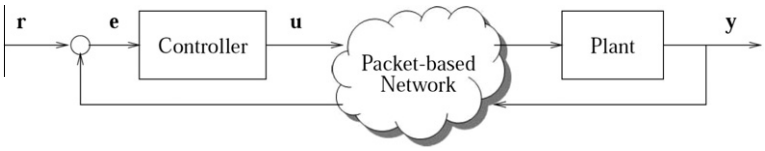

Fig. 1. NCS block diagram.

This paper addresses the problem of building an accurate simulator for NCS's. Up to now, Matlab/Simulink is the standard tool to design and simulate dynamical systems. Unfortunately, it does not simulate the network behavior in a simple way. Therefore, it is necessary to *integrate* Matlab/Simulink with some other tool allowing the network simulation in a easy and trustfully way. Amongst the available possibilities, SystemC endowed with an *ad hoc* library, SCNSL, [13], is the tool selected to simulate a packet-based network.

The use of SystemC compares favorably with respect to other network simulation approaches (e.g., those involving NS-2 [14]) since this language provides a direct path to HW/SW design and synthesis of NCS components. The advantages of SystemC/Matlab in electronic system design have been also highlighted in [15–20].

The crucial point behind this approach is the concept of *co-simulation*. Co-simulation is a strategy to make working together different tools on which different parts of the overall system are simulated. Since the tools have to run simultaneously (to reduce the simulation time) and in a synchronized way (to provide correct results), the exchanging data mechanism is the key point. The contribution of this paper is a co-simulation strategy allowing Matlab/Simulink and SystemC working together to simulate at the same time the control/modeling aspects and the networking aspects.[1] Matlab/Simulink and SystemC are executed in two different processes and they are synchronized through inter-process communications.

This co-simulation strategy can also be seen as a framework to support future co-design of both the control part and the communication part. In fact, while the problem of controlling networked control systems has been studied during the last two decades mainly from the control perspective (see [1,21] for recent surveys), little effort has been devoted to the development of tools to support joint control/network simulation and design [22–29]. This work aims at filling the gap by proposing a co-simulation framework where both the control and network aspects are accurately modeled.

The paper is organized as follows. Section 2 reports some preliminaries and the related work about networked control systems and network communication. In Section 3 the problem is stated together with the design requirements. The proposed co-simulation framework is fully described in Section 4 whereas its validation is reported in Section 5. Conclusions and future work are given in Section 6.

2. Preliminaries and related work

In this section the class of networked control systems is described by highlighting all the critical aspects. The final goal of this analysis is to clearly identify the issues that should be taken into account for an efficient and effective simulation tool.

2.1. Networked control systems

In NCS, the communication channel may significantly affect the quality of the control, because of communication delay and data loss due to the lack of Quality-of-Service (QoS) guarantees.

Fig. 1 shows the classical block diagram scheme for a networked control system. NCS are described in several papers and the available different control solutions are mainly due to the different assumptions on the communication channel [30–33].

From the control viewpoint, the simulation requirements are mainly focused on the possibility to design and to easily implement advanced controllers, novel strategies to send and receive data on the network, packing/unpacking strategies, and concealment techniques. According to the traditional block-based representation of control models, each block generates sequences of scalar or vector sampled data. To send data over a packet-based network, one or more of these samples are grouped together. Grouping more samples into each packet introduces the so-called *algorithmic delay* but increases the transmission efficiency since the overhead cost of the packet header is distributed among several bytes of the payload. Furthermore, compression and error management techniques can be applied efficiently to groups of samples [34,35]. The designer of a NCS has the freedom to decide how (and when) to merge different types of samples in each packet, even sampled at different frequency. Therefore the creation of packets is a critical aspect in the design of NCS's.

Among different simulation tools (e.g. Modelica [36], 20-sim [37]), Matlab/Simulink [38], has become the *de facto* tool (at least in the academic world) to perform such a kind of analysis and synthesis. Within the Matlab framework it is easy to introduce model uncertainties, measurement noise, saturation (just to name a few) in the modeling phase and to design controllers based on standard techniques (e.g., LQG controllers, MPC controllers, passivity theory, robust control, etc.). Unfortunately, the time simulation performance are not so good when highly complex nonlinear dynamical models are involved. For this reason in some cases external host processes have to be developed in C++ to solve the differential equations of complex systems.

[1] Is important to highlight that the proposed co-simulation strategy is quite general and can be adapted to any pair of tools.

The stability of a NCS is not a trivial issue since packet loss and delay may compromise the control loop. In case of teleoperation (taken as test bench in Section 5), the problem is even more critical since a human operator is within the control loop, and force feedback is introduced to improve the perception of a virtual presence, i.e., *transparency* [39]. Therefore user movement and operation depend on the feedback returned by the remote slave. With an ideal infinite-capacity network, transparency increases with the amount of information per time unit transmitted on the channel. For instance, the increase of the command sampling frequency improves the system promptness. Viceversa, with an actual network, when the information rate exceeds its capacity, delay and data loss occur thus reducing control performance and transparency. Therefore, a possible teleoperation design objective could be to find out the optimal transmission rate maximizing transparency.

2.2. Communication perspective

According to traditional results from information theory, a communication channel may loose information when transmission data exceeds its capacity. In the context of NCS this case may happen when the channel is shared among different flows. If no channel arbitration is present (as in case of local area networks and Internet) capacity overflow leads to bit errors, packet drops, and queuing delays. Other causes of bit errors may be noise or hardware failure. In case of retransmission of corrupted or lost packets (e.g., in wireless LAN and TCP) the final effect is an increase of the end-to-end delay. Furthermore, due to the non-deterministic nature of data sources, packet loss rate and delay have time-varying statistical descriptions.

Communication issues can be addressed in the NCS design or in the network design, or both. From the NCS perspective, corrupted packets, lost packets and delayed packets (which do not arrive in time to the actuator input) can be replaced in different ways, e.g., with null data or with estimations based on the previous data [40]. In this direction a further approach to improve robustness consists in adding redundancy to each packet according to its importance [41]. The introduction of redundancy and the estimation of missing data requires to design a smarter receiver. Another approach consists in designing specific transmission protocols to fully control packet creation and retransmission, e.g., those based on the User Datagram Protocol (UDP) [1].

The described communication issues have been already faced in the attempt of introducing multimedia traffic in traditional networks [42–44]. In fact, in term of quality of service, interactive multimedia applications such as telephony-over-IP have the same conceptual constraints as NCS's. Quality-of-service guarantees can be introduced either by adding network resources (*over-provisioning*) or by a smarter allocation of network capacity among applications (refer to [45] for a detailed description). This last approach consists of assigning a different priority to each packet. The model named Integrated Services assigns priorities to each level-4 flow; this approach is very precise but too complex. Other approaches are based on the aggregation of packets into a small number of priority classes; this process can be done at IP-level with the Differentiated Services model, at data-link level with virtual LANs, in wireless LANs with the 802.11e recommendation, or with the Multi-Protocol Label Switching (MPLS).

All these algorithms need to be easily implemented in a simulation tool that aims at supporting a co-design framework. Furthermore, it should:

- provide parametric models of well-known traffic sources, e.g., web traffic, file transfers, voice-over-IP sessions;
- reproduce the main sources of uncertainty within the communication channel since they strongly affect the performance level of the overall system and may also, if not properly managed, destabilize the NCS;
- model the different layers involved in the packet transmission;
- allow to model the priority-based behavior. Since traffic priorities could be a good way to provide quality-of-service guarantees to NCS applications, the designer should be able to test the system performance by using a fine-tuned priority scheme.
- produce at run-time network statistics about the communication channels used by the controlled system; the collection of statistics about packet loss rate, bit error rate, and delay distribution are decisive to assess the quality of service provided by the solution under design.

The *Network* consists of a set of nodes connected through communication links. Each link has specific features, e.g., the type of channel (i.e., wired/wireless), the capacity, and the propagation delay. The way in which nodes exchange data can be modeled at different level of detail, from packet transmission to signal propagation [14,46]. Packet-based network simulators are event-driven simulators whose events consist of packet transmission, queuing in intermediate nodes, drop in case of full queue, dequeuing and reception.

Among the different network simulators present in the literature, this work deals with SystemC [47] and the SystemC Network Simulation Library (SCNSL) [13]. SystemC is a flexible language and simulation environment that reproduces the behavior of concurrent HW and SW systems. SCNSL is an open source library aimed at extending the SystemC simulation capabilities to network components.

3. Motivation and problem statement

Most of the algorithms proposed in literature assume that the network has a statistical behavior. For example the packet loss rate are commonly simulated by using Bernoulli processes [21]. Even though the network has a random behavior, the

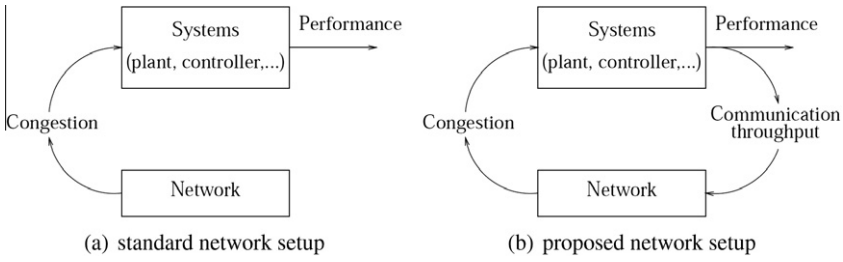

Fig. 2. Processes–network relationships.

statistics are assumed fixed whatever kind of controller is implemented. This approach follows the scheme in Fig. 2a: transmission delays and packet losses modify the performance of the plant but the controller itself does not have any impact on the network congestion. In other word the controller has a network-invariant behavior.

Such approach has some advantages from the analytical perspective: it allows to formally state and prove stability results. On the other hand it is sometimes unrealistic. The simulation approach also takes into account the effect of the traffic due to the measurements (from plant to controller) and to the commands (from controller to plant) on the network. Fig. 2b shows the *circular* mutual effects between network and systems through congestion and communication throughput.

As described in the previous section, not all components of a NCS can be appropriately simulated through the same tool. In digital system design, the concept of co-simulation refers to the integrated simulation of different components of the system under test. Components can belong to different simulation domains (i.e., SW, HW, network and dynamic systems) which may require the coordinated use of different tools and techniques. We propose the same approach for the accurate simulation of NCS.

The combined use of different simulation tools requires their synchronization to share the same simulation time. Fig. 3 shows the tools involved in the simulation of a NCS. For the reason explained in the previous section, the chosen tools are Matlab/Simulink and SystemC. The standard block diagram is reported in Fig. 4 where the two shadowed regions refer to Matlab/Simulink, to model the control architecture, and SystemC, to model the network, respectively. A similar list of processes can be found in the teleoperation scenario shown in Fig. 8 of Section 5 where there are also two C++ executable modules integrating robot master's and slave's differential equations. All these processes must communicate in a proper way. In this work, we focus on the interaction between Matlab and SystemC because it is the most critical point.[2]

The first issue to be solved is the *interconnection of the simulation tools*. Assuming that each tool is executed by a specific process in the host operating system, simulation data should be exchanged by using inter-process communications, e.g., shared memory or network sockets. The former is more efficient but the latter allows distributed simulations over different machines. The transfer of simulation data between tools should be efficient and independent of the complexity of the simulated model.

Another issue consists in the introduction of *new modeling entities* in each tool to represent the connection of the standard entities provided by the tool with the other components modeled by the other tool. For example, with reference to Fig. 4, in Matlab/Simulink workspace new blocks are needed to represent the fact that the controller and the plant are connected together through a component modeled by SystemC. Clearly, the same issue concerns the network model in the SystemC tool in which there should be a clear representation that some nodes exchange data with Matlab.

The third issue is the creation of the *same time domain* for the global simulation which implies that tools should perform simulation in a synchronized way and that cause-effect relationship between events belonging to different tools should be preserved. For example, when Matlab generates data for SystemC, message transfer is not enough since this event has to be seen with the same simulation time by both tools. Events caused by the external tool should co-exist with internal events and generate other subsequent events giving the effect that a unique simulation is performed.

4. The co-simulation framework

A co-simulation framework consists of a communication mechanism between the simulation tools, specific modeling entities, and a synchronization algorithm to create a global simulation time. These issues are described here in details.

4.1. Communication mechanism between simulators

In this section, we discuss how Matlab and SystemC have to be modified to build an integrated simulation environment. Each simulator consists of a kernel and a set of simulated modules. The kernel controls the simulation time and calls simulated modules according to a given policy. Matlab kernel is a time-driven simulator and calls modules periodically. SystemC kernel is an event-driven simulator and it manages a list of time-sorted simulation events.

[2] It is worth highlighting that the proposed approach can be easily adapted to make SystemC jointly working with any other tool handling the control part.

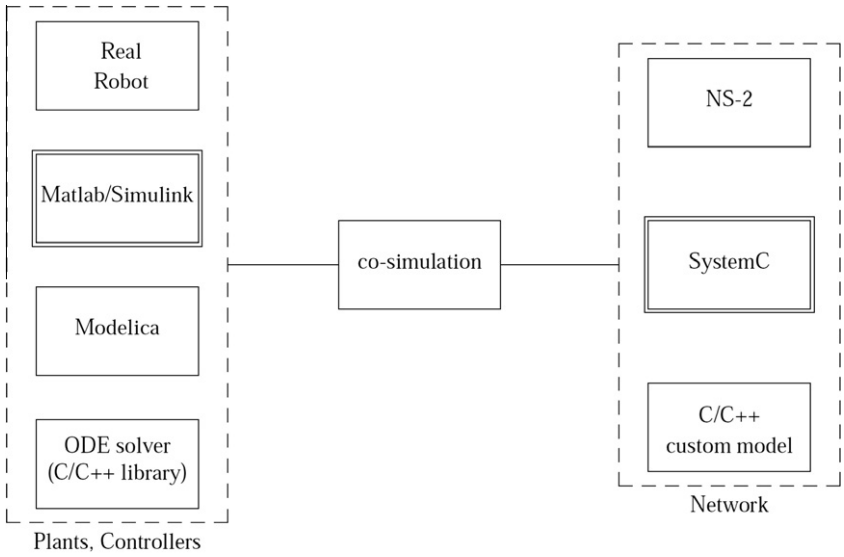

Fig. 3. Co-simulation and available tools.

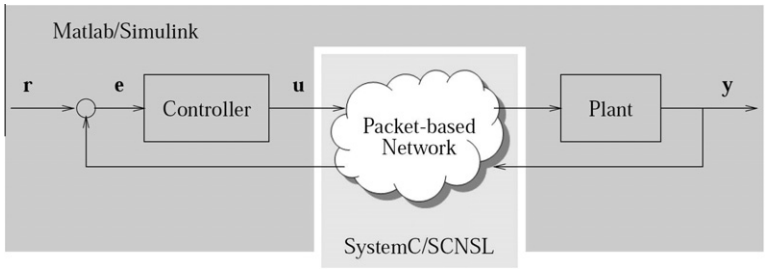

Fig. 4. Chosen tools for NCS co-simulation.

When a simulated module needs to communicate with another module belonging to the other simulator, two solutions are given, i.e., *module-based communication* and *kernel-based communication*. In the first case, each pair of modules belonging to different simulators should manage an inter-process channel (e.g., a socket). In this case, an explicit user-level API for message exchange has to be placed in both modules limiting their re-use. Furthermore, each pair of interacting modules needs a different channel leading to scalability issues. Finally, consistent time synchronization is not easy to achieve since simulation time is managed within each simulation kernel. In kernel-based communication, the simulation kernels directly exchange messages to transfer data between models and to keep synchronization. With respect to the former approach, synchronization is easier to maintain and communication overhead is lower when multiple module pairs are involved in the interaction.

As depicted in Fig. 5, this work follows a mixed approach. Kernel-based communication is used at SystemC side since an open source implementation of the kernel has been extended. Module-based communication is used at Matlab side since it is a proprietary tool. To this purpose, a special Simulink block, named *Matlab Wrapper*, has been developed as Level-2 S-function.

All modules interacting with SystemC are connected to this Wrapper which concentrates all inter-tool communications thus preserving scalability. The notion of simulation time can be recovered in the Wrapper since Matlab simulation is time-driven and events happen at constant rate.

The Matlab Wrapper uses a socket library to connect to the SystemC kernel. The use of sockets, instead of shared memory mechanisms, allows to distribute simulation not only among different CPU's but also among different hosts to enable load-balancing or remote on-demand business models.

4.2. New entities

In this section new entities at the Matlab/Simulink side and at the SystemC side will be illustrated. They are based on the concept of *port* which is shared between the interacting modules belonging to different tools. The port has a unique identifier and a storage size which in this Matlab/SystemC co-simulation tool allows to record a double-precision floating-point number.

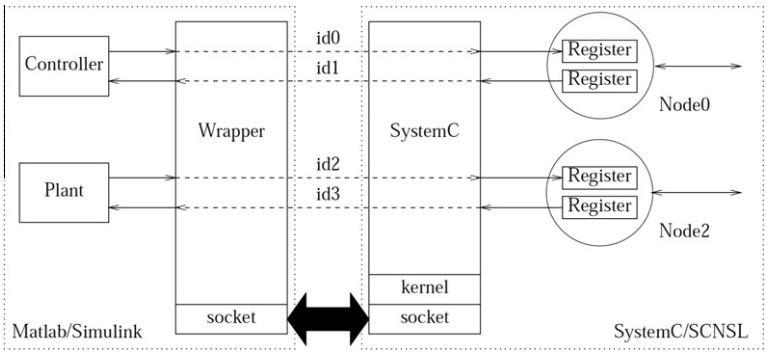

Fig. 5. Relationship between the new entities in Matlab and SystemC.

4.2.1. New entities in Matlab/Simulink

In Matlab's models the role of bridge towards the other simulation tool is played by the Matlab Wrapper. This block can be connected to a user-defined number of scalar and vector input and output ports. Each port has a unique identification number and a given update frequency. Matlab executes this block at the highest of these frequency values and for each input/output port to be updated it calls the corresponding *Packer/Unpacker*, respectively.

The Packer collects data from the port, puts them in the body of a co-simulation message to be sent to the SystemC kernel. The message also contains the identification number of the port and the simulation timestamp. The Unpacker writes data on a given port when the Wrapper receives and elaborate a message coming from SystemC with the corresponding destination identifier. As discussed in Section 3, the creation of the message body from raw data is a design task and, therefore, the designer of the NCS is in charge of the implementation of the different Packers/Unpackers.

All messages to/from Wrapper's input/output ports are multiplexed in the socket connected to SystemC kernel.

4.2.2. New entities in SystemC

The SystemC library has been extended to provide the new ports `cosim_out` and `cosim_in` to send/receive data to/from another simulation tool (in our case Matlab). They are derived by template classes `sc_out` and `sc_in`, and are managed by overriding methods `write()` and `read()`. Each instance of `cosim_out` and `cosim_in` has a unique identification number which is used to associate it to the corresponding port in the Matlab Wrapper.

Whenever the SystemC kernel receives a message from Matlab, it generates an event for the destination `cosim_in` port with the timestamp specified in the message. The kernel then wakes up the function sensitive to that event and this function can retrieve data by calling the `read()` method. Whenever a SystemC process calls the `write()` method to send data to a `cosim_out` port, the simulation kernel builds a message for Matlab.

4.2.3. Use of entities to model NCS

Fig. 5 shows a possible use of these entities in a modeling scenario for NCS's as in Fig. 1. The Simulink block named `Controller` exchanges data on the network through Node0; to this purpose, the output flow of `Controller` is linked to a SystemC input port (`cosim_in`) in Node0 and the input flow is linked to SystemC output port (`cosim_out`) in Node0. Similarly, Node2 receives data from the network and sends them to the Simulink block named `Plant`. Node0 and Node2 are connected through a network simulated by SCNSL that is not reported in Fig. 5.

4.3. Synchronization

The synchronization between simulation tools should take into account different simulation approaches, i.e., time-driven and event-driven approaches. Matlab follows a time-driven approach in which:

- the output and the internal state are evaluated at constant-time steps,
- the length of the time step may change during simulation to better describe the evolution of the system.

SystemC follows an event-driven approach in which:

- events do not happen at constant rate,
- events have a fire time and they are kept in a ordered list according to the fire time.

Each tool, i.e., Matlab and SystemC, contains a part of the whole simulation model and performs simulation on it; tools are executed concurrently by the operating system. Therefore a synchronization mechanism is needed to assure that the same simulation time is kept by both tools. For instance, if a simulator reads/writes data from/to another simulator whose

simulation time is different, then the effect of such data in the first/other simulator may not be correct. In the rest of the section different synchronization approaches are presented. All of them are based on the exchange of messages through the socket; each message contains the port identifier, the command (e.g., read value and write value), the simulation time of the tool which generated it and the data (in case of write operation). Synchronization is performed by blocking read/write operations on the socket. In fact, a blocking socket operation can interrupt simulation both in the Matlab Wrapper and in the SystemC kernel. The exact approach is based on the traditional algorithm to access shared memory; it assures perfect alignment of simulation times but it requires that just one tool is running at each time. The other approaches allow concurrent execution at the cost of more complex algorithms.

4.3.1. Exact approach

In the exact approach each port between the SystemC model and the Matlab Wrapper is considered as a variable shared between two concurrent processes. Each simulator executes until a cosimulation event is to be scheduled; at this point it blocks itself and notifies its simulation time to the other simulator which is blocked on another cosimulation event; an handshake is performed in which the simulation times are compared and the simulator with the lower one is re-started.

4.3.2. Approach 1

Fig. 6 shows the pseudocode of the first approach of the synchronization algorithm implemented in the SystemC kernel (left side) and in the Matlab Wrapper (right side). Both algorithms start by reading messages coming from the other peer tool. The corresponding events are put in a time-sorted queue. In SystemC kernel this queue also contains internally-generated events either involving internal modules or requiring to send a message to the peer. Then, SystemC kernel executes all the queued events belonging to the same time while Matlab executes all the queued events belonging to past and current time slots. An event to be executed could be old if it comes from the other peer whose simulation time is lower. In this case, its timestamp is updated to the current time before execution and a warning notification is issued to the designer since simulation output may not be correct. Then, Matlab sends a message to SystemC for each input port of the Wrapper. Finally, both tools wait for an acknowledge for each request of a read or write operation and increase the simulation time.

An acknowledge is waited after the request of a read or write operation; in case of read command, the acknowledge also bears the requested value. If a tool receives a request with a lower simulation time, sends immediately the acknowledge and issues a warning on the user console since simulation results may be wrong. If the request has a greater simulation time, the receiver serves it when the same simulation time is reached; since the acknowledge is delayed the tool which made the request is forced to wait this realigning.

4.3.3. Approach 2

The second approach is based on the first one but introduces a further mechanism to re-align simulation times. If a tool has a higher simulation time than the one of the received command, it sends immediately the acknowledge and issues a warning as in approach 1 in Section 4.3.2; then it blocks itself after having asked the requester to be notified when it reaches the same simulation time.

4.3.4. Approach 3

To use this synchronization approach each tool has to know the timing of future events involving data exchange with the other tool. This approach is based on the definition of some synchronization checkpoints. The first checkpoint is fixed in the setup phase, tools exchange the time of the next event involving cosimulation, i.e. read/write operations. A special message (called NEXT in the following) is used. The tool receiving a NEXT message records the time of the next event in its event queue (Fig. 7a and b).

```
while( T < Tmax )
{
  read messages coming from the other peer
  and enqueue them (together with local events)

  pick all the next events from the queue
  with the same timestamp and execute them (for
  each executed event which is old, issue
  a warning notification to the user)

  for each executed event for which a message has
  been sent to the peer, wait for its response

  update T to the timestamp of the executed event
}
```

```
while( T < Tmax )
{
  read messages coming from the other peer
  and enqueue them

  pick all the next events from the queue falling
  in the current time slot or in the previous ones
  and execute them (for each executed event which
  is old, issue a warning notification to the user)

  for each packet to be sent to the network, send
  a message to the peer

  for each message sent to the peer, wait for its
response

  increase T of 1 time slot
}
```

Fig. 6. Pseudocode of the time-synchronization procedure implemented in (left) the SystemC kernel and (right) the Matlab network wrapper.

After this setup phase the tools start execution till the first checkpoint. At the checkpoint, the enqueued NEXT message blocks the receiver waiting for the read/write message of the other tool, Fig. 7c. The other tool sends the corresponding message, computes the next cosimulation event, sends a new NEXT message and waits for a NEXT message coming from the cosimulation peer to update its queue with the new checkpoint. On the other side, after handling the event, Fig. 7d, the tool waits for the NEXT event message of the peer, computes its next cosimulation event and sends a NEXT event message, Fig. 7e and f.

Once tools have updated their event queue with the new NEXT messages, the next synchronization point is defined and both events executed till the next checkpoint, Fig. 7g.

In case of Matlab, the evaluation of the next co-simulation event is simple since the update frequency of the Wrapper ports is known. In case of SystemC, only the next event in the queue is known and it could not be a cosimulation event. For this reason, a virtual clock has been introduced to generate periodic synchronization events inside SystemC simulator; if the next event is not a cosimulation event, the virtual synchronization event is used. The virtual clock frequency is set by the user in SystemC at the beginning of the cosimulation session; the higher is the frequency, the slower is the simulation but a too low frequency could lead to mis-alignment of the simulation times. The right trade-off depends on the simulation model.

The detailed pseudo-code of this approach is reported in Appendix A.

5. Tool validation

A particular example of NCS is a bilateral teleoperation system that is introduced here as a test-bench scenario. A bilateral teleoperation system consists of a *master device* through which the operator controls the *remote slave robot* and a *packet-based communication channel* which delivers all the signals (e.g., commands and measurements). Fig. 8 shows the block diagram of a teleoperation architecture. A model of a Unimation PUMA 560 is used as slave robot having the same degrees of freedom of the master device.

The master device interacts with a human operator O and is controlled by the master controller C_m, whereas the slave robot interacts with the environment E and is controlled by the slave controller C_s. A packet-based communication network connects the two sides and the interface blocks, I_m and I_s, encode commands and measurements into network packets. Usually teleoperation is used on very delicate and dangerous tasks as robotic surgery and nuclear plant maintenance. This means that a very accurate tool to simulate the overall plant (devices and network) is crucial. Some recent results about the teleoperation control problem can be found in [48–56].

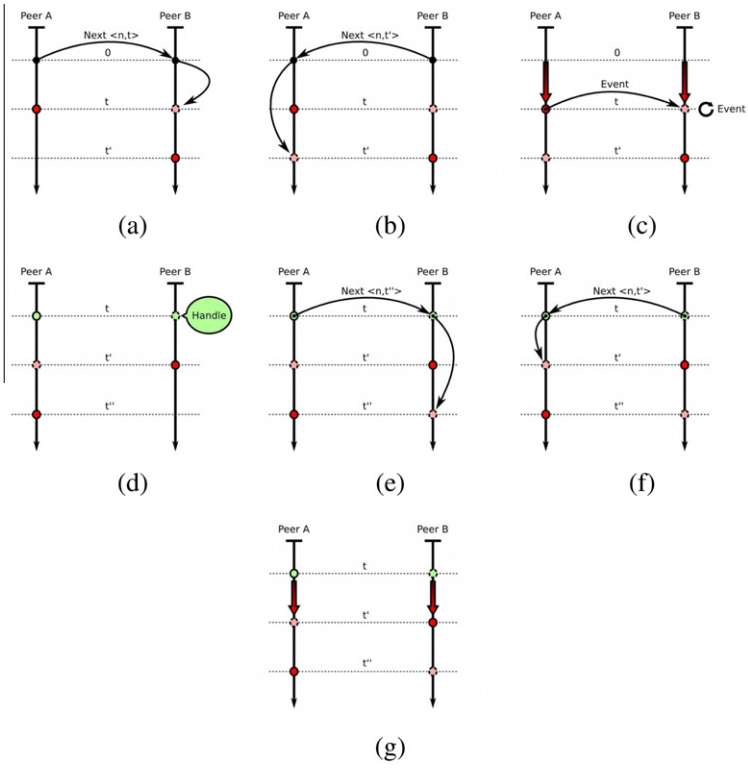

Fig. 7. Synchronization phases of the co-simulation Section 4.3.4.

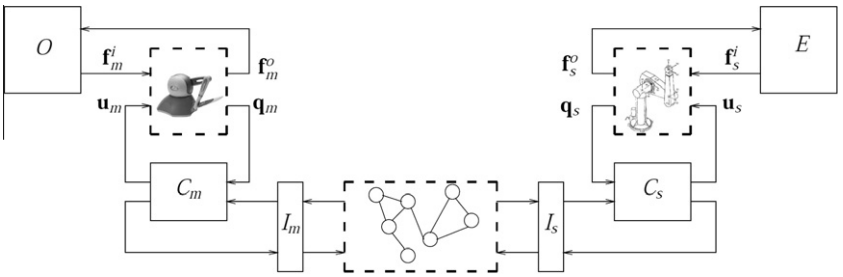

Fig. 8. Teleoperation scheme.

The dynamic model of any serial link manipulator with n degrees of freedom (as the Puma robot) can be written as a set of nonlinear differential equations:

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + N(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{u} \tag{1}$$

where $\mathbf{u} = [u_1 \quad \dots \quad u_n]^T$ is the command torque vector, $\mathbf{q} = [q_1 \quad \dots \quad q_n]^T$ is the vector of generalized coordinates (e.g., angles) with corresponding velocity $\dot{\mathbf{q}}$ and acceleration $\ddot{\mathbf{q}}$. In this standard Lagrangian representation, M is the symmetric non singular inertia matrix, C is the Coriolis and centrifugal force matrix and N is the gravitational and frictional torques at the joints [57].

In this scenario, the slave robot moves according to the commands coming from the master through the network and it has to *replicate* as accurately as possible the movement of the master robot due to the operator O (eventually, by using a scaling factor to improve accuracy). At the slave side, controller C_s aims at:

- increasing performance (model-based controller, gravity compensation),
- preventing undesirable effects due to disturbance and/or out of range commands,
- interacting safely with the environment E.

At the master side, controller C_m aims at:

- providing a weightless and frictionless master device for the operator (gravity and friction compensation),
- providing force feedback perception (i.e., tele-presence).

The goal of any teleoperation systems is to improve *transparency* by enhancing both the operator promptness and the robustness against communication delays and packet losses.

Let $\mathbf{q}_m(t)$ and $\mathbf{q}_s(t)$ be the joint angular positions at the master and at the slave side, respectively. The vector $\mathbf{q}_m(t)$ is sent by the master robot to the slave side through the network. The slave controller receives this data after the communication delay from master to slave $\tau_{m2s}(t)$ or never if the packet gets lost. The controller C_s uses this data as reference signal and computes the command in order to have $\mathbf{q}_s(t)$ close to $\mathbf{q}_m(t)$. The sensors at the slave side measure the joint angles $\mathbf{q}_s(t)$ and send them back to the master side. The operator experiences these values after the communication delay from slave to master $\tau_{s2m}(t)$ or never if the packet gets lost. The communication delays $\tau_{m2s}(t)$ and $\tau_{s2m}(t)$ are related to the state of the network at time t.

The following tracking error gives a measure of the teleoperation performance:

$$\mathbf{e}_{ms}(t) := \mathbf{q}_m(t) - \mathbf{q}_s(t). \tag{2}$$

It summarizes the overall displacements between the slave angles and the master angles due to the control aspects and the network aspects. The simulations refer to a sampling frequency of 100 Hz for the controllers and to a teleoperated scenario without force feedback (i.e., the slave is moving in an unconstrained environment).

The proposed simulator aims at testing the effect of both the control strategy and the communication problems (i.e., delays and packet losses) on the overall system performance. As said before, the objective of the paper is to verify that co-simulation can be a proper tool to accurately reproduce the behavior of a networked control system and to study novel control algorithms and network protocols.

Fig. 9 shows the network scenario described with SCNSL and implemented in the present test bench. Nodes are represented by circles and links by continuous arrows. For each link, capacity C and delay D are defined. The scenario represents a typical bottleneck topology in which peripheral nodes are connected through dedicated links to a shared backbone link. Nodes have queues to store packets exiting towards a busy link; the Figure reports the queues of Nodes 1 and 2 which are the crossing points of different network paths. Since the backbone capacity is shared among the different traffic flows, queue level may vary during simulation and congestion may happen. Over this topology two end-to-end application flows have been defined between applications end-points; in the figure, they are represented by dashed lines between squared

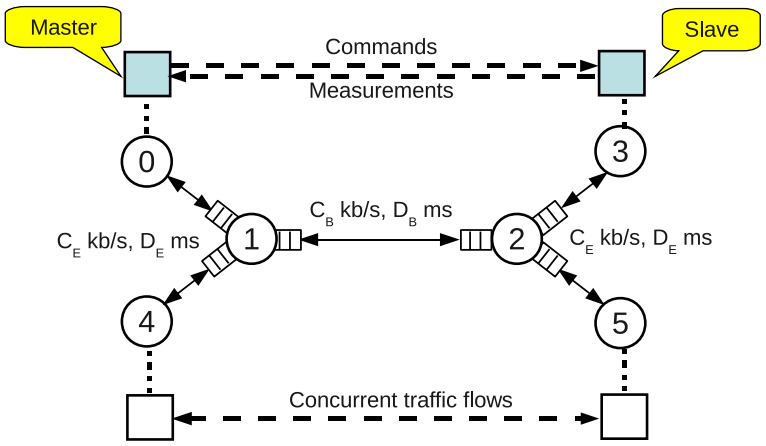

Fig. 9. Example of network scenario described with the SystemC network simulation library.

boxes. The packet flow between master and slave sides in the teleoperation application (in general, between controller and plant in a NCS) has a constant bit-rate since samples of commands and measurements are taken at constant rate and put in packets. A concurrent flow has been modeled between Nodes 4 and 5. It features an ON/OFF behavior with constant bit-rate during ON periods.

Table 1 reports all the parameters related to the simulated network scenario. The capacity of edge links C_E has been chosen to allow each corresponding traffic. Viceversa, to study the effect of network congestion, the capacity C_B of the bottleneck link has been chosen to be smaller than the sum of the bit rates of the teleoperation application B_{APP} and the concurrent traffic B_{CONC}. This means that when the concurrent traffic is ON, packets are accumulated in the queues at Nodes 1 and 2 and they experience increasing transmission delay. Having the queues a finite length L_Q, the teleoperation system will also experience packet dropouts. In the simulated scenario, the parameters have been chosen in such a way that in the congested intervals the percentage of lost packets is around 20% for the teleoperation application. Propagation delays D_B and D_E refer to the time needed to move a bit on the link. The delay experienced by a packet to traverse a link is the sum of the propagation delay of the link and the time needed to enter the link which depends on packet size and link capacity. Therefore, when the simulated network is not congested, the delay experienced by a teleoperation packet is:

$$D_E + \frac{S_{APP} \cdot 8}{C_E} + D_B + \frac{S_{APP} \cdot 8}{C_B} + D_E + \frac{S_{APP} \cdot 8}{C_E}. \tag{3}$$

Starting from this network scenario, two test cases have been considered. In the first test case only the teleoperation application uses the network and thus there is no congestion. In the second test case, the concurrent traffic is introduced to reduce channel availability. The master and slave controllers are the same in both test cases. The same trajectory is applied by the operator to the master device.

Fig. 10 shows the tracking error (joint by joint) in the first test case when there is no congestion, i.e., when the delays $\tau_{m2s}(t)$ and $\tau_{s2m}(t)$ are constant and computed according to Eq. 3.

With reference to Fig. 9, in the second test case the concurrent traffic flow has been introduced to study the control performance when the network experiences congestion. Figs. 11 and 12 show the communication delays τ_{m2s}, τ_{s2m} and the packet loss rates p_{m2s}, p_{s2m}, respectively. The vertical lines separate the ON and OFF intervals in both directions. During OFF periods, the delay is constant and equal to Eq. 3 as in the previous case. This is the lower bound value of the propagation

Table 1
Parameters of the simulated network scenario.

Name	Value	Description
D_E	5×10^{-4} s	Propagation delay in both direction (edge links)
D_B	5×10^{-4} s	Propagation delay in both direction (bottleneck link)
C_E	1312 kbit/s	Capacity of edge links (both directions)
C_B	6560 kbit/s	Capacity of the bottleneck (both directions)
S_{APP}	1000 byte	Packet size (application flow)
S_{CONC}	1000 byte	Packet size (concurrent flow)
T_s	0.01 s	Sample time of the control loops (master and slave side)
L_Q	40,000 byte	Queue length
B_{APP}	800 kbit/s	Bit rate of the teleoperation application (both directions)
B_{CONC}	6320 kbit/s	Bit rate of concurrent traffic when active (both directions)

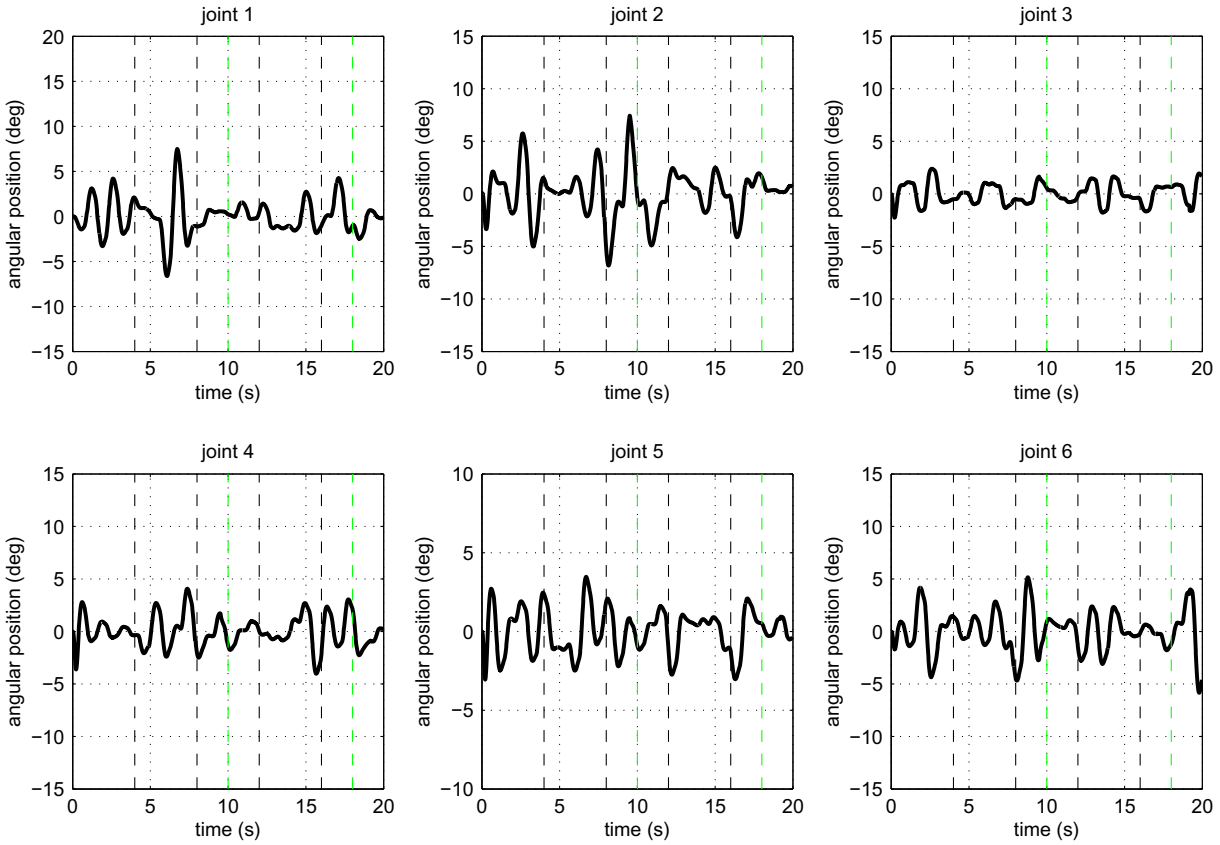

Fig. 10. Joint angular errors $\mathbf{e}_{ms}(t)$ in the un-congested network scenario.

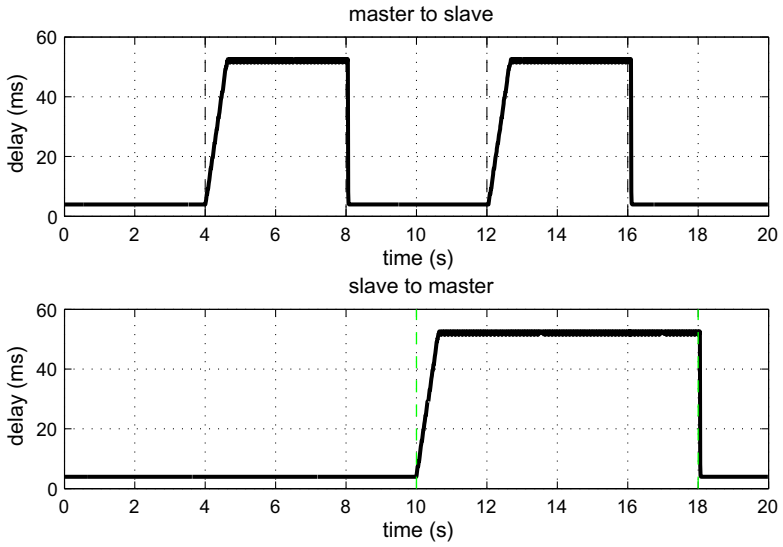

Fig. 11. Communication delays experienced by the teleoperation application in the congested scenario. Top: master to slave route. Bottom: slave to master route.

delay. When the concurrent source is switched on, queues at the edges of the shared link start to grow and the delay increases. When the queues are full, arriving packets are dropped as Fig. 12 shows. The time needed to fill a queue completely depends on the queue length and on the bit rate difference between incoming flows and exiting flow; in the simulated case the filling time is:

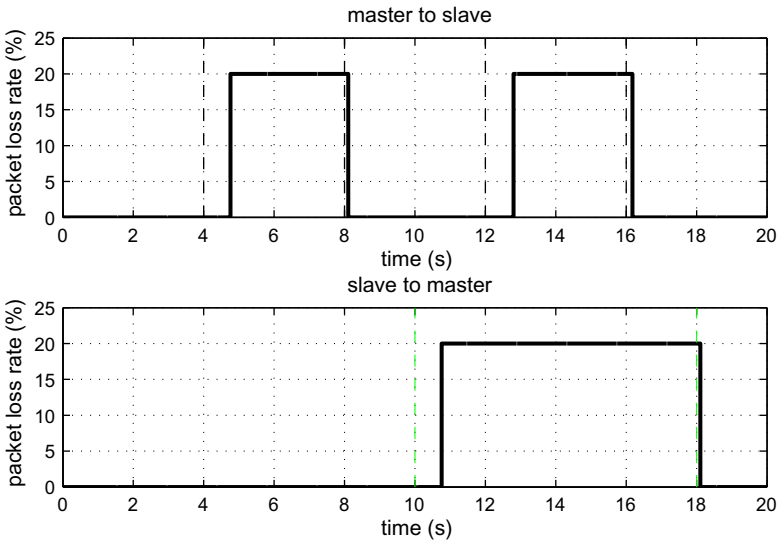

Fig. 12. Packet loss rate experienced by the teleoperation application in the congested scenario. Top: master to slave route. Bottom: slave to master route.

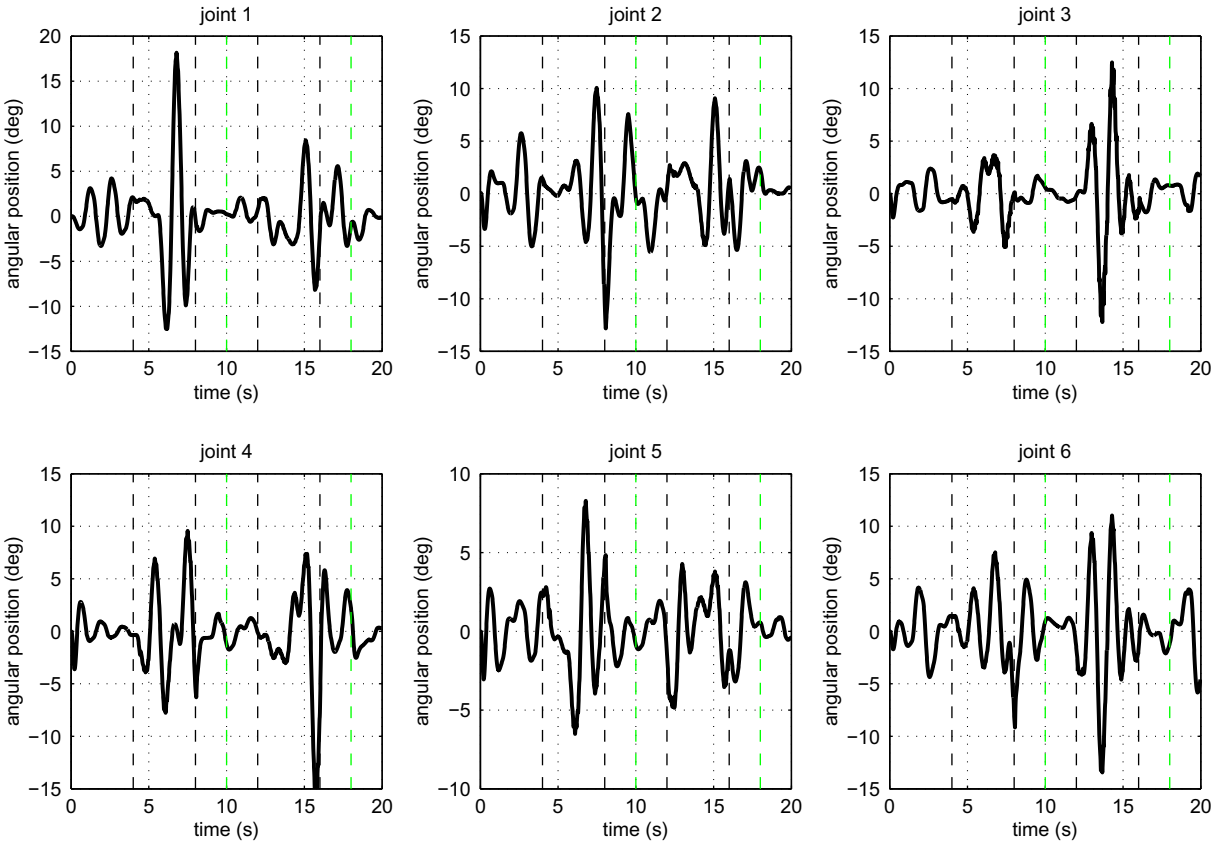

Fig. 13. Joint angular errors $\mathbf{e}_{ms}(t)$ in the congested network scenario.

$$\frac{L_Q \cdot 8}{B_{APP} + B_{CONC} - C_B}. \tag{4}$$

During congestion the queue remains full since a new packet is enqueued only when another packet leaves the queue. These packets experience an additional delay of $L_Q \cdot 8/C_B$ seconds which leads to the almost horizontal plateau of Fig. 11. When the concurrent traffic is switched off, the number of enqueued packets decreases and the queue becomes empty after

$L_Q \cdot 8/(C_B - B_{APP})$ seconds. In the reported case this slope seems almost vertical. This phenomenon leads to the "trapezoidal shapes" plotted in Fig. 11. Comparing Figs. 10 and 13, it is easy to see that the error increases whenever the network is congested. Figs. 10 and 13 have been generated by Matlab while Figs. 11 and 12 have been generated by SCNSL.

These results show that the co-simulation tool works as expected according to the reported equations. More complicated network scenarios and concurrent traffic models (e.g., probabilistic, actual recorded traffic, etc.) can be easily implemented to model all possible kinds of working conditions.

6. Conclusions and future work

A comprehensive and formally-correct co-simulation framework for networked control systems has been developed and presented. It combines a network simulator (SystemC) and Matlab/Simulink for control design and modeling. To reach this goal, several critical aspects have been discussed and analyzed. The most important is the time synchronization of the different tools. An algorithm has been developed and implemented in Matlab and in SystemC by ad hoc components. The proposed co-simulation strategy has been validated on a detailed model of a teleoperated system in which master commands and slave measurements are transmitted over a packet-based network. Teleoperation packets share a network link with interfering ON/OFF traffic thus leading to congestion. Simulation results capture well the effect of congestion in term of increasing delay, packet loss and tracking error thus showing the effectiveness of the tool to validate different design solutions both from the control and the network perspective. Future work will exploit the tool to explore joint control/network design solutions; for instance, different mechanisms can be introduced in the network to assure a given quality of service and the controller can be designed to take into account the network behavior.

Acknowledgement

This work has been partially supported by the European Project CONtrol FOR COORDination of distributed systems (CON4COORD – FP7-2007-IST-2-223844).

Appendix A. Pseudocode approach 3

The following pseudo-code shows the scheduler of an event-driven simulator, i.e., SystemC, extended to synchronize with Matlab by using the approach described in Section 4.3.4. The following definitions holds:

`read-nb ()`	socket non-blocking read
`read-b ()`	socket blocking read
`write ()`	write a message on the socket
`msg`	message exchanged between tools
`enqueue ()`	put an event on the time-sorted event queue
`dequeue ()`	extract the next event from the time-sorted event queue
`queue.next-event ()`	get (without extracting) the next event from the time-sorted event queue
`execute ()`	execute an event

The pseudo-code is as follows:

```
1 get_hello_message ();// Get the hello message from ISS/Matlab instance;
2 msg = read_nb (socket);// Get the next event from peer;
3 if (msg.type!=NEXT) print_error_message;
4 else enqueue (msg);// Compute the next event

5// Search in the queue events to the peer
6 for (j = 1; j < event_queue->size (); j++){
7   if(event.type==TO_PEER){
8     found = true;
9     next_ev.type = NEXT_EVENT;
10     next_ev.time = event.time;
11     //Communicate to peer the time of my next event
12     write (next_ev, socket);
13   }
14}
```

(*continued on next page*)

```
15 if (found==false){// Unable to compute my next event
16  next_ev.type = NEXT_EVENT;
17  next_ev.time = next_v_tick;
18  //Communicate to peer a virtual time in which we'll synchronize
19  write (next_ev, socket);
19}
20 while (T < Tmax){// Start to execute events
21  do {
22    event = dequeue ();
23    if (event.type==READ || event.type==WRITE || event.type==INTERRUPT
23        ||event.type==STOP)
24      ret = execute (event);
25    else if (event.type==NEXT_EVENT){
26      // Found a next event message computed by me
27      if(event.isInternal){// Compute the next event
28        // Search in the queue events to the peer
29        for (j = 1; j < event_queue->size (); j++){
30          if(event.type==TO_PEER){
31            found = true;
32            next_ev.type = NEXT_EVENT;
33            next_ev.time = event.time;
34            // Communicate to peer the time of my next event
35            write (next_ev, socket);
36          }
37        }
38        // I was unable to compute my next event
39        if (found==false){
40          next_ev.type = NEXT_EVENT;
41          next_ev.time = next_v_tick;
42          // Communicate to peer a virtual time in which
43          // we'll synchronize
45          write (next_ev, socket); synchronize
46        }
47        // Handle events until a next message from peer
48        // gets received
49        while (1){
50          msg = read_b (socket);
51          if (msg.type==READ || msg.type==WRITE || msg.type==INTERRUPT
52                  || msg.type==STOP)
53                ret = execute (msg);
54          else if(msg.type==NEXT_EVENT){
55                // Replace eventually old next event from peer
56                // with this new next event message
57                for (j = 1; j < event_queue->size (); j++){
58                  if (event.type==TO_PEER && event.isExternal==true)
59                    extract (event);
60                }
61                push (msg);
62                break (while);
63          }
64        }
65      }// end if (event.isInternal)
66      else {// Found a next event coming from peer
67        // Prevent to handle an old next_event of the peer
68        if (msg.time < last_v_tick) break;
69          // Search if there's a fake event with the same
70          // notify_time of this next_event.
71          // If I find it I won't serve it.
72          for (j = 1; j < event_queue->size (); j++){
```

```
73              if (event.type==TO_PEER && event.time==msg.time){
74                break;}// Do not serve it.
75          }
76          // Handle events until a next message from peer
77          // gets received
78          while (1)
79              msg = read_b (socket);
80              if (msg.type==READ || msg.type==WRITE || msg.type==INTERRUPT
81                        || msg.type==STOP)
82                ret = execute (msg);
83              else if (msg.type==NEXT_EVENT){
84                // Replace eventually old next event from
85                // peer with this new next event message
86                for (j = 1; j < event_queue->size (); j++){
87                  if (event.type==NEXT_EVENT && event.isExternal==true)
88                    extract (event);
89                }
90              push (msg);
91              break (while);
92              }
93            }
94          }// end if (event.isExternal)
95        }
96        else {
97          print_error_message ();
98          exit;
99        }
100     T = event.time;
101   }
102 }
```

References

[1] J. Hespanha, P. Naghshtabrizi, Y. Xu, A survey of recent results in networked control systems, Proceedings of the IEEE 95 (1) (2007) 138–162.
[2] B. Hannaford, Stability and performance tradeoffs in bi-lateral telemanipulation, in: Proceedings of the IEEE International Conference on Robotics and Automation, 1989.
[3] J. Colgate, Robust impedance shaping telemanipulation, IEEE Transactions on Robotics and Automation 9 (4) (1993) 374–384.
[4] A. Eusebi, Sistemi di teleoperazione bilaterale di posizione e forza in presenza di ritardi, Ph.D. thesis, Universita degli studi di Bologna, 1995.
[5] C. Benedetti, M. Franchini, P. Fiorini, Stable tracking in variable time-delay teleoperation, in: Proceedings of 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001.
[6] G. Leung, B. Francis, J. Apkarian, Bilateral controller for teleoperators with time delay via mu;-synthesis, IEEE Transactions on Robotics and Automation 11 (1) (1995) 105–116.
[7] R. Oboe, P. Fiorini, A design and control environment for internet-based telerobotics, The International Journal of Robotics Research 17 (4) (1998) 433.
[8] L. Huang, K. Sezaki, End-to-end internet delay dynamics, in: Proceedings of the 6th Asia–Pacific Conference on Communications (APCC), 2000.
[9] Q. Li, D. Mills, Jitter-based delay-boundary prediction of wide-area networks, IEEE/ACM Transactions on Networking 9 (5) (2001) 578–590.
[10] P.H. Bauer, M.L. Sichitiu, K. Premaratne, On the nature of the time-variant communication delays, in: Proceedings of the IASTED Conference on Modeling, Identification and Control (MIC'01), 2001.
[11] Y. Tsang, M. Coates, R. Nowak, Network delay tomography, IEEE Transactions on Signal Processing 51 (8) (2003) 2125–2136.
[12] S. Floyd, V. Paxson, Difficulties in simulating the internet, IEEE/ACM Transactions on Networking 9 (4) (2001) 392–403.
[13] F. Stefanni, D. Quaglia, F. Fummi, SystemC Simulation of Networked Embedded Systems, in: Languages for Embedded Systems and their Applications, Springer Lecture Notes in Electrical Engineering, vol. 36, 2009, pp. 201–211.
[14] S. McCanne, S. Floyd, NS Network Simulator – version 2 <http://www.isi.edu/nsnam/ns>.
[15] C. Xi, X. Ningyi, Z. Zucheng, A methodology for SystemC algorithmic model verification applying MATLAB, in: Proceedings of 5th International Conference on ASIC, 2003, vol. 1, IEEE, 2004, pp. 294–297.
[16] Y. Vanderperren, G. Sonck, A. Paul van Oostende, A design methodology for the development of a complex SoC using UML and executable system models, in: The Proceedings of Forum on sSpecification & Design Language, 2002, pp. 64–69.
[17] K. Hylla, J. Oetjens, W. Nebel, Using SystemC for an extended MATLAB/Simulink verification flow, in: Specification, Verification and Design Languages, 2008, FDL 2008, Forum on, IEEE, 2008, pp. 221–226.
[18] J. Boland, C. Thibeault, Z. Zilic, Using MATLAB and Simulink in a SystemC verification environment, in: North American SystemC Users Group Meeting, 2004.
[19] K. Tomasena, J. Sevillano, N. Arrue, A. Cortés, I. Vélez, Embedding Matlab in SystemC transaction level modeling for verification, in: Design of Circuits and Integrated Systems Conference, DCIS, 2009.
[20] W. Hassairi, M. Bousselmi, M. Abid, C. Sakuyama, Using matlab and Simulink in SystemC verification environment By JPEG algorithm, in: 16th IEEE International Conference on Electronics, Circuits, and Systems, 2009, ICECS 2009, IEEE, 2010, pp. 912–915.
[21] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, S. Sastry, Foundations of control and estimation over lossy networks, Proceedings of the IEEE 95 (1) (2007) 163–187.

[22] A. Chamaken, L. Litz, M. Kramer, R. Gotzhein, Cross-layer design of wireless networked control systems with energy limitations, in: European Control Conference 2009 (ECC'09), Budapest, Hungary, 2009, pp. 2325–2330.
[23] L. Xiao, M. Johansson, H. Hindi, S. Boyd, A. Goldsmith, Joint optimization of communication rates and linear systems, IEEE Transactions on Automatic Control 48 (1) (2003) 148–153.
[24] L. Shi, K. Johansson, R. Murray, Estimation over wireless sensor networks: Tradeoff between communication, computation and estimation qualities, in: Proceedings of IFAC World Congress, vol. 17, 2008, pp. 605–611.
[25] A. Al-Hammouri, M. Branicky, V. Liberatore, Co-simulation tools for networked control systems, hybrid systems: computation and control, in: HSCC 2008, LNCS, vol. 4981, 2008, pp. 16–29.
[26] S. Shakkottai, T. Rappaport, P. Karlsson, Cross-layer design for wireless networks, IEEE Communications Magazine 41 (10) (2003) 74–80.
[27] M. Branicky, S. Phillips, W. Zhang, Scheduling and feedback co-design for networked control systems (I), in: IEEE Conference on Decision and Control, vol. 2, 2002, pp. 1211–1217.
[28] D. Dacic, D. Nesic, Simultaneous controller and protocol design for networked control systems with packet based communication, in: 2006 45th IEEE Conference on Decision and Control, 2006, pp. 508–513.
[29] Z. Lei, H. Dimitrios, Communication and control co-design for networked control systems, Automatica 42 (6) (2006) 953–958.
[30] M.-Y. Chow, Z. Sun, H. Li, Optimal stabilizing gain selection for networked control systems with time delays and packet losses, IEEE Transactions on Control Systems Technology 17 (5) (2009) 1154–1162.
[31] T. Matiakis, S. Hirche, M. Buss, Control of networked systems using the scattering transformation, IEEE Transactions on Control Systems Technology 17 (1) (2009) 60–67.
[32] G. Walsh, H. Ye, L. Bushnell, Stability analysis of networked control systems, IEEE Transactions on Control Systems Technology 10 (3) (2002) 438–446.
[33] F.-L. Lian, J. Moyne, D. Tilbury, Network design consideration for distributed control systems, IEEE Transactions on Control Systems Technology 10 (2) (2002) 297–307.
[34] Jerry D. Gibson et al, Digital Compression for Multimedia: Principles and Standards, Morgan Kaufmann, 1998.
[35] Y. Wang, Q. Zhu, Error control and concealment for video communication: a review, Proceedings of the IEEE 86 (5) (1998) 974–997.
[36] Modelica Association, Modelica <http://www.modelica.org>.
[37] Controllab Products B.V., 20-sim <http://www.20sim.com>.
[38] Mathworks Co., Matlab <http://www.mathworks.com>.
[39] P. Hokayem, M. Spong, Bilateral teleoperation: an historical survey, Automatica 42 (12) (2006) 2035–2057.
[40] L. Schenato, To zero or to hold control inputs with lossy links?, IEEE Transactions on Automatic Control 54 (5) (2009) 1093–1099
[41] S. Lin, D.J. Costello, Error Control Coding: Fundamentals and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1983.
[42] D. Quaglia, J. De Martin, Delivery of MPEG video streams with constant perceptual quality of service, in: Proceedings of the International Conference Multimedia and Exhibition, vol. 2, 2002, pp. 85–88.
[43] J. De Martin, D. Quaglia, Distortion-based packet marking for MPEG video transmission over DiffServ networks, in: IEEE International Conference on Multimedia and Expo, 2001.
[44] E. Masala, D. Quaglia, J. De Martin, Variable time scale multimedia streaming over ip networks, IEEE Transactions on Multimedia 10 (8) (2008) 1657–1670.
[45] R. Hunt, A review of quality of service mechanisms in IP-based networks – integrated and differentiated services, multi-layer switching, MPLS and traffic engineering, Computer Communications 25 (1) (2002) 100–108.
[46] AWE Communications, WinProp: Software-Tool for the Planning of Mobile Communication Networks <http://www.awe-communications.com>.
[47] IEEE Std 1666 – 2005, IEEE Standard SystemC Language Reference Manual, IEEE Std 1666–2005, 2006, pp. 1–423.
[48] A. Aziminejad, M. Tavakoli, R. Patel, M. Moallem, Transparent time-delayed bilateral teleoperation using wave variables, IEEE Transactions on Control Systems Technology 16 (3) (2008) 548–555.
[49] N. Chopra, P. Berestesky, M. Spong, Bilateral teleoperation over unreliable communication networks, IEEE Transactions on Control Systems Technology 16 (2) (2008) 304–313.
[50] J. Yan, S. Salcudean, Teleoperation controller design using infty-optimization with application to motion-scaling, IEEE Transactions on Control Systems Technology 4 (3) (1996) 244–258.
[51] Y. Xu, H.B. Brown, Jr., M. Friedman, T. Kanade, Control system of the self-mobile space manipulator, IEEE Transactions on Control Systems Technology 2(3) (1994) 207–219.
[52] M. Sirouspour, S. Salcudean, Suppressing operator-induced oscillations in manual control systems with movable bases, IEEE Transactions on Control Systems Technology 11 (4) (2003) 448–459.
[53] I. Aliaga, A. Rubio, E. Sanchez, Experimental quantitative comparison of different control architectures for master–slave teleoperation, IEEE Transactions on Control Systems Technology 12 (1) (2004) 2–11.
[54] J.-H. Ryu, C. Preusche, B. Hannaford, G. Hirzinger, Time domain passivity control with reference energy following, IEEE Transactions on Control Systems Technology 13 (5) (2005) 737–742.
[55] J. Speich, M. Goldfarb, An implementation of loop-shaping compensation for multidegree-of-freedom macro–microscaled telemanipulation, IEEE Transactions on Control Systems Technology 13 (3) (2005) 459–464.
[56] A. Casavola, E. Mosca, M. Papini, Predictive teleoperation of constrained dynamic systems via internet-like channels, IEEE Transactions on Control Systems Technology 14 (4) (2006) 681–694.
[57] R. Murray, Z. Li, S. Sastry, A Mathematical Introduction to Robotic Manipulation, CRC Press, 1994.