Simulation Modelling Practice and Theory 117 (2022) 102501

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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

Cyber–physical system modeling with Modelica using message passing communication

Victorino Sanz [*], Alfonso Urquia

Dpto. de Informática y Automática, E.T.S.I. Informática, Universidad Nacional de Educación a Distancia (UNED), Juan del Rosal, 16, 28040, Madrid, Spain

ARTICLE INFO

Keywords:
Cyber–physical systems
Hybrid systems
Modelica
Object-oriented modeling
Simulation

ABSTRACT

Modelica is an object-oriented modeling language whose design and features facilitate the description of cyber–physical systems (CPS). Message passing communication (MPC), seen as the transmission of impulses of information between model components, eases the description of the discrete-event parts of CPS models. However, Modelica does not currently supports MPC. Modelica supports an equation-based component connection rationale, where Modelica tools automatically transform component connections into model equations, following a physical modeling approach. The differences between MPC and Modelica connections are analyzed. A proposal for supporting MPC in Modelica is presented, inspired by the coupled PDEVS model communication approach. The presented MPC proposal is based on the definition of structures to manage messages, named buffers, interface ports and communication channels. Also, an implementation of the proposed MPC mechanism in the form of a new free Modelica library, named MSGLib, is presented. MSGLib includes functionality to manage and dynamically store messages, and describe component communications. Two examples, a pick and place system and a robotic arm, are presented to demonstrate the use of the library, and its combination with other Modelica models.

1. Introduction

Cyber–physical systems (CPS) are composed of physical and computer-based (i.e., cyber) parts, which are highly interconnected. Modeling and simulation of CPS is an extensive field of research, due to their ubiquity and increasing importance [1,2]. Hybrid dynamic models, composed of a combination of continuous-time and discrete-event dynamics, are commonly used in Control Engineering applications for describing CPS.

The research presented in this paper is aimed to extend Modelica functionality, so that message passing communication (MPC) among model components is facilitated. MPC can be used to describe multiple types of systems where asynchronous communication is used between components, such as computer networks (e.g., Ethernet, TSN) or field buses (e.g., CAN, PROFIBUS, FlexRay) commonly found in CPS. Message passing constitutes an alternative communication approach that can be used to describe transmission of impulses of information between model components. The MPC approach is different to the equation-based connections already supported by Modelica. Both communication approaches can be combined in the same model to enrich the description of component connections. This new functionality, combined with the rest of Modelica features, provides a versatile approach for describing CPS in Modelica using a single modeling language and simulation tool, facilitating the task of users and

[*] Corresponding author.
E-mail addresses: vsanz@dia.uned.es (V. Sanz), aurquia@dia.uned.es (A. Urquia).

https://doi.org/10.1016/j.simpat.2022.102501
Received 30 July 2021; Received in revised form 13 January 2022; Accepted 23 January 2022
Available online 5 February 2022

model developers. Additionally, MPC in Modelica serves to facilitate the combination of multiple discrete-event modeling formalisms with the rest of Modelica functionality to describe other kind of hybrid systems.

The development of the proposed MPC mechanism is based on previous developments performed by the authors to support Parallel DEVS (PDEVS) [3] and process-oriented modeling [4] in Modelica. These developments have been used by other authors to describe state machines [5]. The previous work [3,4] was focused on the description of PDEVS models, while the work presented in this paper is focused on the description of a general-purpose MPC mechanism. This new mechanism can be used to describe PDEVS models but it is not limited to it, as shown in the case study presented in Section 7. Also, these previous developments [3,4] imposed several limitations on the use of the MPC mechanism:

- The description of different types of messages required the modification of the Modelica code and the external C code used to manage them.
- The description of collective communication between multiple components required the use of intermediate components (e.g., the `DUP` model) to duplicate the messages transmitted from one source to multiple destinations.
- Similarly, an additional model (e.g., the `BreakLoop` model) was needed to describe communication loops, in order to break the algebraic loop defined between the variables of the connections in the loop.

The current proposal, implemented in the MSGLib library, provides a versatile and powerful MPC mechanism without these limitations.

The structure of the paper is as follows. The problem of supporting message passing communication in Modelica is described in Section 2. The related work is discussed in Section 3. Next, a proposal for supporting MPC in Modelica is presented in Section 4 from a conceptual point of view. The elements required for MPC in Modelica are presented. The MSGLib library, that constitutes the implementation of the proposal, is presented in Section 5. Two case studies are discussed: the support of PDEVS using the proposed MPC and its application to describe a model of a pick and place system [6] is presented in Section 6; and, in Section 7, a model of a six degrees-of-freedom robotic arm from the Modelica Standard Library (MSL) version 3.2.3 [7] is modified to integrate an Ethernet network from MSGLib in order to show the combination of the physical modeling paradigm [8] and MPC. Finally, some conclusions and future work ideas are presented in Section 8.

2. Problem description

Modelica is an object-oriented modeling language that supports modular and hierarchical modeling, and multiple inheritance [9]. It is designed to facilitate abstraction, and information encapsulation and hiding. Model classes can be described as composed of internal behavior, and interface.

The internal behavior can be described as a combination of equations, discrete-events and algorithms (i.e., sections of imperative assignments) [10,11]. Event conditions, based on the model state and the simulated time, are evaluated during numerical integration in order to detect, precisely find the instant of the occurrence, and trigger events [12]. Also, some Modelica functions may trigger events during their evaluation [13]. Discrete-time variables can only be modified during the management of events. The management of events can be used to efficiently handle discontinuous behavior in the models [14]. The *if* and *when* clauses can be used to select different sets of equations, depending on the evaluation of event conditions. The *sample* operator can be used to trigger periodic time events (e.g., for describing sampled systems). The *pre* operator can be used to obtain the value of a discrete variable or a discrete-time expression previous to an event. Synchronous language elements, such as *Clocks*, have been included in Modelica to facilitate the description of periodic control systems [15].

The interface of Modelica models can be composed of `connector` classes, that are used to declare interface variables classified either as *potential* (by default), *flow* (using the `flow` modifier) and *stream* (using the `stream` modifier). Connections between component connectors can be defined using `connect` sentences, which are automatically translated into equations by the Modelica modeling environment [16].

The component connection rationale is different in discrete-event modeling formalisms, such as DEVS [17] and discrete-event simulation tools such as Arena [18] (to cite only two examples of many). Model interface in DEVS and Arena is composed of ports. DEVS models use output ports to send messages to other models, while entities in Arena use them to leave the modules. On the other hand, DEVS models receive messages in their input ports, and entities in Arena use them to enter the modules. Thus, connecting component ports is equivalent to establishing channels for MPC.

The description of component connections using an MPC approach is not facilitated in Modelica. Modelica's `connector` class and `connect` sentence do not fully satisfy the requirements to describe the transmission of information impulses, or messages, between models. An approach to describe MPC using current Modelica connections is discussed next, followed by its limitations.

Messages could be described using any data structure supported by Modelica (e.g., the `record` class), and declared as components of a `connector`. The message communication channel could be defined using `connect` sentences. The transmission of a message could be described by assigning the desired values to components of the `connector`, at discrete instants. Also, if the value of the new message equals its previous value, a boolean variable could be included in the `connector` and used to signal the transmission event.

Modelica follows the synchronous data-flow principle and the single-assignment rule [13]. The former indicates that *"at every time instant the active equations express relations between variables which have to be fulfilled concurrently"*. The latter imposes that *"the total number of equations is identical to the total number of unknown variables"*, thus a variable can only be computed using a unique set of properly arranged (sorted) equations. These rules facilitate the symbolic manipulation of the model equations performed by Modelica tools in order to generate the executable code, and to guarantee deterministic simulations. However, the application of these rules when trying to describe MPC results in the following limitations:

- Modelica component connections do not allow to separate the information transmitted from the communication channel, since the connection introduces equations between the variables declared in the connectors. The variables declared in a connector cannot be assigned with different values at the same time instant, to represent different messages that need to be transmitted simultaneously through the same communication channel. When the number of simultaneous messages is known in advance, simultaneous transmissions could be described using multiple connectors, or multiple variables declared in the same connector, but this makes the development of the model more complex and error prone. For example, the solution of packet collisions in Ethernet buses may involve immediate re-trying of the communication, thus sending multiple messages through the same port during the same time instant.
- Similarly, collective communications with multiple sources of information at the same time (i.e., $N-1$ connections) are not allowed due to the application of the single-assignment rule. Modelica does not allow to connect multiples sources of messages, where the variables of the connectors are assigned with different values, to a single destination. Following the previous example, nodes connected to the same Ethernet bus may perform simultaneous communications that lead to packet collisions.
- The equations that describe component connections in certain system topologies (e.g., ring networks, feedback loops, and loops of processes in logistic systems) introduce algebraic loops among discrete-time variables. For example, modeling full-duplex communication in computer networks. Also, these topologies may require to send multiple messages through the same port due to a chain of events propagated across the loop during the same instant, requiring to define different triggering conditions for each event in the chain.

Additional extensions to Modelica have included language elements that can be used to describe synchronous behavior usually found in sampled data systems [15]. However, MPC communication is usually asynchronous and these new semantics do not get over the mentioned limitations for describing MPC. Asynchronous message transmissions could be described using conditionally defined clock variables, whose condition is triggered when a new message is either sent or received. However, this communication approach does not provide a solution for the simultaneous transmission of a previously unknown number of messages, since dynamically changing variables are not supported, nor the description of collective communications, since these will depend on the standard Modelica connection approach.

3. Related work

Multiple effort has been performed to design and develop specific tools that support both continuous-time and discrete-event dynamics found in CPS. This effort can be divided into:

- *Discrete-event Simulators* that are extended by including continuous-time and numerical integration functionality to allow the description and simulation of continuous-time dynamics. This approach includes the use of traditional numerical integration methods [19], DEVS-based implementations of these methods [20], or Quantized State System (QSS) integration methods [21].
- *Continuous-time Simulators* that are extended to facilitate the description of discrete-event models. These extensions make use of the discrete-event detection and management functionality already supported by some tools. Some examples are the TrueTime toolbox for Matlab/Simulink [22], or multiple Modelica libraries such as the Network Library [23], TrueTime [24] or the Modelica_EmbeddedSystems library [25].

The application of formal methods [26,27] and multi-paradigm modeling [28] to the description of CPS has been also considered. Another approach constitutes the development of Domain Specific Languages (DSL) that facilitate the description of systems in particular application domains [29–31].

This effort can be further extended with the use of a co-simulation approach [32–35]. Co-simulation considers the combination of multiple simulators, each one specialized in the description of a particular part of the CPS. This approach seeks to leverage the specific functionality of each tool to provide a better or easier description of the CPS. Also, it allows to combine and reuse already available models of each part, facilitating the development of new models.

Many co-simulation approaches take advantage of the standardization effort that has been performed to facilitate the inter-operation of tools, such as the High Level Architecture (HLA) [36] and the Functional Mock-up Interface (FMI) [37]. Some authors have also discussed about the integration [38] and combination [39] of both standards.

Co-simulation examples include the combination of NS-2, Modelica, VHDL, SystemC, Matlab, POOSL and QEMU among many others [32–34,40]. A detailed analysis of co-simulation applied to the description of power and ICT (Information and Communication Technology) systems is performed in [35].

The necessity to agree on a unifying formalism to allow for conceptual composability to support collaborative ensembles for describing CPS is discussed in [41], but still there is not agreement on how to achieve this objective.

The Modelica language constitutes an international effort to facilitate system modeling applying the physical modeling paradigm [8]. A detailed description of Modelica and its functionality can be found in the language specification [13]. Modelica is widely used in academia and industry. The proposal described in this paper is oriented to improve the modeling functionality provided by Modelica in order to better fulfill the necessities found when modeling CPS.

4. Proposal for MPC in modelica

The proposed MPC mechanism is designed to describe component connections analogously to the current Modelica `connector` class and `connect` sentence, solving the limitations discussed. This proposal is based on the communication mechanism defined in the PDEVS formalism [17]. PDEVS and its operational semantics are briefly introduced next to facilitate the comprehension of the proposal.

PDEVS models can be described behaviorally, as atomic models, or structurally, as coupled models. An atomic PDEVS model is defined by the tuple [42]:

$$AM = (X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$$

where, $X = \{(p, v) | p \in InPort, v \in X_p\}$ and $Y = \{(p, v) | p \in OutPort, v \in Y_p\}$, $InPort$ is the set of input ports, X_p is the set of possible input values in the port p, $OutPort$ is the set of output ports, and Y_p is the set of possible output values in the port p. S is the set of sequential states. $\delta_{int} : S \rightarrow S$, $\delta_{ext} : Q \times X^b \rightarrow S$, and $\delta_{con} : Q \times X^b \rightarrow S$ are the internal, external and confluent transition functions. $\lambda : S \rightarrow Y^b$ is the output function and $ta : S \rightarrow \mathbb{R}^+_{0,\infty}$ is the time advance function. X^b and Y^b are bags over the elements in X and Y, respectively, $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$, and e is the time elapsed since the last transition.

Coupled PDEVS models are defined by the tuple [42]:

$$CM = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC)$$

where, X and Y are defined as for atomic PDEVS models. D is the set of component names. M_d is the set of PDEVS models, for each $d \in D$. EIC are External Input Couplings, that represent connections between inputs of CM with its internal components. EOC are External Output Couplings, that represent connections between internal components with outputs of CM. IC are Internal Couplings, that represent connections between internal components.

As defined, the interface of PDEVS models is composed of input and output ports. Message transmissions are triggered by the occurrence of internal events scheduled in time. The simulator manages an event calendar in order to properly order and execute internal events. The simulation is performed as follows [42]:

- The time for the most imminent internal event (t_{next}) is obtained from the calendar, together with the set of components that have an internal transition scheduled at that time (INT).
- Events triggered at time t_{next} are removed from the calendar.
- Simulation time is advanced to the imminent event instant ($t = t_{next}$).
- For all elements in INT, the output function is evaluated with the current state of each component ($\lambda(s)$). This may generate output messages that are synchronously transmitted to their destinations, following the defined couplings between ports (EIC, EOC and IC), and triggering external transitions in the receiving components (EXT).
- A transition function is evaluated for each component in INT and EXT:
  - For the elements in $\{INT - EXT\}$, that have an imminent internal event and do not have received an external input, the internal transition function is used to calculate the new state $s_{new} = \delta_{int}(s)$.
  - For the elements in $\{INT \cap EXT\}$, that have scheduled an imminent internal event and have received an external input, the confluent transition function is used to update the state $s_{new} = \delta_{con}(s, e, bag)$.
  - For the elements in $\{EXT - INT\}$, that have external inputs and do not have scheduled an imminent internal event, the external transition function is used to update the state $s_{new} = \delta_{ext}(s, e, bag)$. Scheduled internal events for the elements in EXT need to be removed from the calendar.
- For each element with updated state in $\{INT \cup EXT\}$ a new internal event is scheduled at time $t_{new} = ta(s_{new}) + t$, and included in the calendar.

Note that message transmissions are synchronous and depend on the time scheduled for executing internal transitions. Asynchronous behavior can be described by the proper scheduling of internal events.

The proposed MPC mechanism is composed of the following elements:

- *Buffers* represent the structures used to store and manage messages within models. This concept is equivalent to the *bags* of events used in PDEVS [42]. Buffers can be used as an internal storage for messages, allowing to define a dynamically changing data structure. This functionality is new to Modelica, since variables of dynamically changing size are not supported.
- *Ports* are used to describe the MPC interface of models, analogously to the Modelica `connector` class. Since MPC communication is directional, ports need to be classified into input, output and router types. Input ports are used to describe the destination of messages. Output ports are used to describe the origin of messages. Router ports are used to describe the interface of coupled models, as intermediate points in the communication. This definition of the interface is analogous to that of PDEVS, and is illustrated in Fig. 1.
- The communication *channels* define the flow of messages between ports, analogously to the `connect` sentence. The direction of the communication needs to be maintained by channels, from output to input ports. Channels describe how a message is transmitted from the buffer of an output port to the buffer of the destination input port.

In order to comply with the PDEVS communication mechanism and maintain a deterministic behavior of the simulations that include message transmissions, the following rules have to be applied:

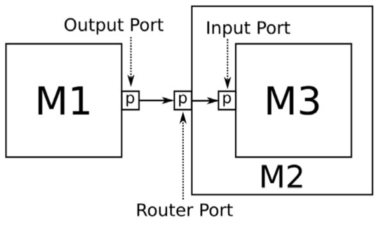


Fig. 1. Interface ports.

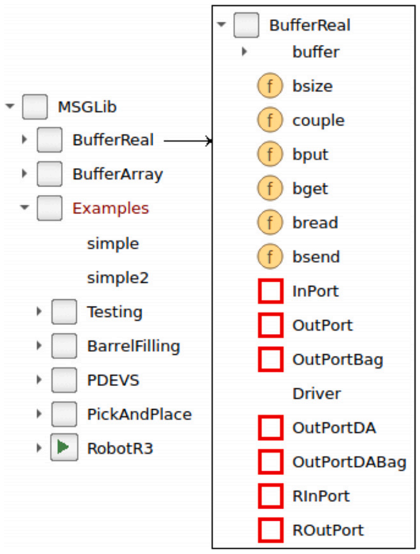


Fig. 2. MSGLib library structure.

- Message transmissions can only be scheduled using time events.
- The values of the messages to be transmitted can only depend on parameters, constant variables or the `pre()` values of discrete-time variables.

When the transmission event is triggered, the content of the message is known as it only depends on constants, parameters or the state of the model previous to the event, which complies with the generation of outputs in PDEVS. Also, the transmission of messages only depends on time events, as required by PDEVS due to the use of the time advance function to schedule new internal events. The equations corresponding to the transmission of output messages will be correctly ordered, usually at the beginning of the model and always before message receptions.

The application of these rules does not impose any limitation in the use of the MPC mechanism. Message communication as a result of the management of an state event can be performed scheduling an immediate time event for the transmission. The transmitted message will be based on the state of the model after the state event, that corresponds to the state of the model before managing the time event. Modelica provides an event iteration procedure to manage the occurrence of these chains of nested events.

5. Implementation of the proposed MPC mechanism

The preferable way to implement the proposed MPC mechanism would be to introduce new language elements in Modelica (e.g., buffer, port, channel), and implement their semantics into Modelica tools. In this section, an implementation of the MPC mechanism described in Section 4 is presented in the form of the MSGLib library. MSGLib has been developed using Modelica 3.4 functionality in order to be used in any Modelica tool. The structure of the library is shown in Fig. 2. The elements, models and functions, included in the library to describe the components of the MPC mechanism are illustrated in the figure, together with the example packages also included in the library. These elements correspond to the names in typewriter font included in the description below.

The MPC elements introduced in the previous section have been implemented as follows:

- *Buffers* have to be described using a Modelica external object (`buffer`), in order to support the variability in their size. An external object is a data structure whose behavior is externally defined using C code. Additional functions are implemented to observe the size of the buffer (bsize), insert (bput), extract (bget), read (bread) and transmit (bsend) messages from buffers. Individual messages are represented as single Real numbers (BufferReal package) or as arrays of Real numbers (BufferArray package), that can be managed using the appropriate data types in Modelica. This representation of messages is more versatile than the previous approach, that was based on the use of C structs and Modelica records, since different types of messages can be described as arrays of Real numbers of different lengths. For example, the age, height and weight of a person can be represented using an array of three numbers, while the brand, engine power, kilometers and speed of a car can be represented using an array of four numbers. Note that string-to-number mappings can be pre-defined during modeling (e.g., to describe different car brand names as numbers). Additionally, other types of messages could still be easily represented adapting the external C code.

- *Ports* are implemented as input (InPort), output (OutPort, OutPortBag, OutPortDA, and OutPortDABag), and router (RInPort and ROutPort). Each port includes a buffer, used to store the transmitted messages, and a Real variable (M), used to synchronize the transmission of messages between ports. Additionally, the OutPort port includes: the message itself (msg), the transmission time (tSend), and a Driver model. OutPortBag and OutPortDABag do not include the msg, but allow to insert the messages directly into the buffer before scheduling the transmission (i.e., they represent a bag of output messages). Also, OutPortDA and OutPortDABag include a driver array (note these models include "DA" in their names) that can be used to schedule multiple transmissions in subsequent event iterations during the same instant.
  The transmission of messages is managed by the Driver model. Driver implements the state automaton shown in Fig. 3. It has four states: initial, passive, active and fired. From the initial state it switches to active or fired, depending on if a message transmission is scheduled at the beginning of the simulation or not. The driver remains in passive state until the time scheduled for a new transmission is reached, switching to active state. When active, the msg is copied to the buffer and transmitted, and the value of M is increased. In OutPortBag and OutPortDABag, messages are already in the buffer so they are just transmitted. The driver remains in the fired state until a new transmission is scheduled ahead in time, thus each driver model can be used to transmit a single message per time instant. As mentioned before, multiple messages can be transmitted during the same time instant adding additional drivers to an output port and using them sequentially (e.g., driver arrays in the OutPortDA and OutPortDABag ports).
- *Channels* are described as relationships between the buffers and the M variables of the ports. These relationships can be described using MSGLib as follows:

  – The couple function has been implemented in MSGLib to define coupling relationships between buffers. These couplings describe how the transmitted messages are routed from output ports to input ports, either directly or across router ports (e.g., couple(A,B) defines a directed coupling from buffer A to buffer B). Messages transmitted from output buffers are immediately transported to their destinations using these coupling relationships. The internal implementation of the couple function automatically analyzes the routes from source buffers to destinations, eliminating intermediate router buffers. This way, transmitted messages are directly routed to all their destinations, and the required copies are performed if multiple destinations are simultaneously present. For example, defining couple(A,B) and couple(A,C), all messages transmitted through buffer A will be copied to buffers B and C. This behavior simplifies the definition of coupling relationships, eliminating the requirement of using the DUP model to define such collective communications.
  – Relationships between the M variables of the ports need to be added as equations, considering that the M variable of each input port has to be equaled to the sum of the M variables of the ports coupled to that input. The reception of messages is detected observing the variation of values of the M variable of an input port. Synchronization between transmission and reception is then guaranteed due to the synchronous data-flow principle, since the variations of the M variables are synchronized in output and input ports. Also, since the transmission of messages is synchronized, it is performed using time events, and the value of the messages is known at the beginning of each simulation step, the description of loops in the structure of the model does not generate algebraic loops between the variables used to define the channels. Thus, the use of an intermediate model to break the algebraic loop (e.g., BreakLoop) is not required and facilitates the description of the structure of the model.

  Some examples of communication channels are shown in Fig. 4, including couplings between buffers and equations between M variables:

  – $M1 \rightarrow M2$ (top-left) shows how a simple direct coupling between models $M1$ and $M2$ is described. Messages from $M1$ will be transported to M2.
  – $M1, M2 \rightarrow M3$ (top-right) shows an example of many-to -1 coupling. Models $M1$ and $M2$ can simultaneously send messages to $M3$.
  – $M1 \rightarrow M2, M3$ (bottom-left) shows an example of 1-to-many coupling. Messages from $M1$ will be simultaneously transported to $M2$ and $M3$.
  – $M1, M2 \rightarrow M3 \rightarrow M4$, shows an example of router port. Models $M1$ and $M2$ can simultaneously send messages to $M3$ but these are automatically routed to $M4$ as their final destination, since port in $M3$ is an intermediate router port.

6. Case study: Pick and place system

The functionality of MSGLib and its support to PDEVS is demonstrated by means of modeling a pick and place robotic system, composed of a robot, a controller and a computer network. The objective of this example is to illustrate the integration of an

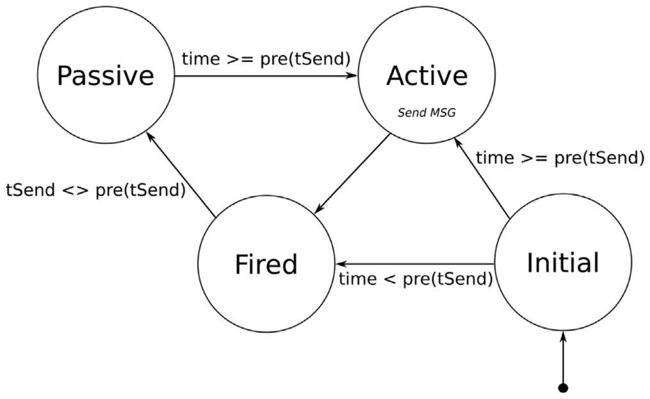
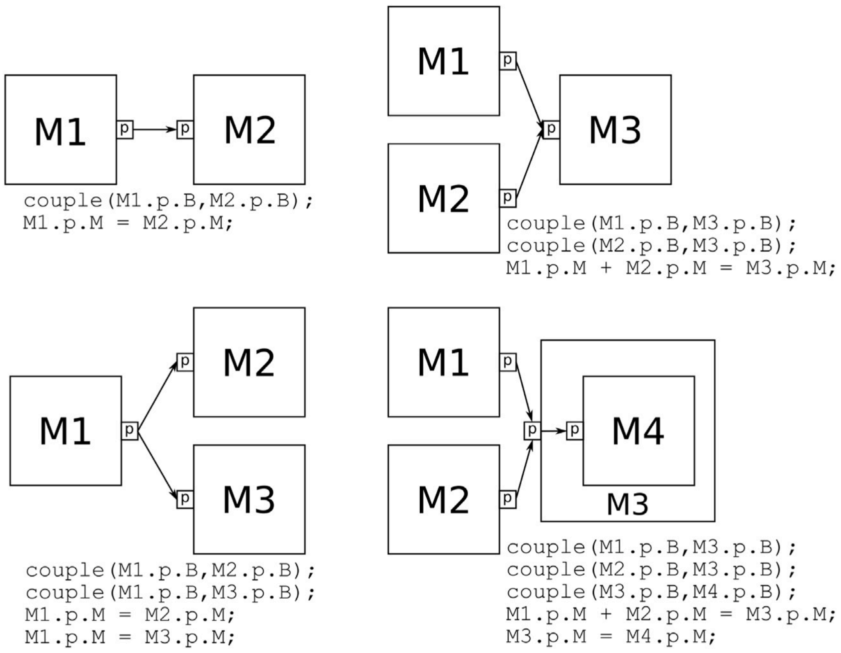


Fig. 3. State transition diagram of the `Driver` model.



```
couple(M1.p.B,M2.p.B);
M1.p.M = M2.p.M;
```

```
couple(M1.p.B,M3.p.B);
couple(M2.p.B,M3.p.B);
M1.p.M + M2.p.M = M3.p.M;
```

```
couple(M1.p.B,M2.p.B);
couple(M1.p.B,M3.p.B);
M1.p.M = M2.p.M;
M1.p.M = M3.p.M;
```

```
couple(M1.p.B,M3.p.B);
couple(M2.p.B,M3.p.B);
couple(M3.p.B,M4.p.B);
M1.p.M + M2.p.M = M3.p.M;
M3.p.M = M4.p.M;
```

Fig. 4. Examples of MPC communication channels.

Ethernet model with a control system, composed of a continuous-time plant and a discrete PID controller. The Ethernet model illustrates the new features introduced in MSGLib: messages represent network packets, and are described using an array of nine Real numbers without any modification in the library; multiple nodes are directly connected to the network, without requiring any additional intermediate model (e.g., DUP) nor generating algebraic loops; and, the transmission of messages is synchronized and allows the implementation of the Ethernet media access control protocol, which sometimes requires the instantaneous re-transmission of packets during the management of collisions. The parameters of the model have been replicated from its previous implementation in [6], in order to facilitate the comparison. The structure of the model and its components are shown in Fig. 5.

6.1. Robot

The robot is composed of two inter-connected arms that are attached to the ceiling and move in the X-Z plane. A scheme of the robot is shown in Fig. 6. The robotic arms are moved using motors, one for each arm, whose torques ($\boldsymbol{\Gamma} = (\Gamma_1, \Gamma_2)$) are the inputs for the robot model. These motors are placed in the revolute joints, P_1 and P_2, which are hanged from the ceiling at points x_p and $-x_p$. The angles of these revolute joints with the X axis constitute the outputs of the robot model ($\mathbf{q} = (q_1, q_2)$). Each arm is composed of two beams, also connected using revolute joints, of lengths L and l, and masses m_1 and m_2, respectively. The end of both arms is connected to a single tool, the traveling plate of mass m_3, used to load a mass m_l.

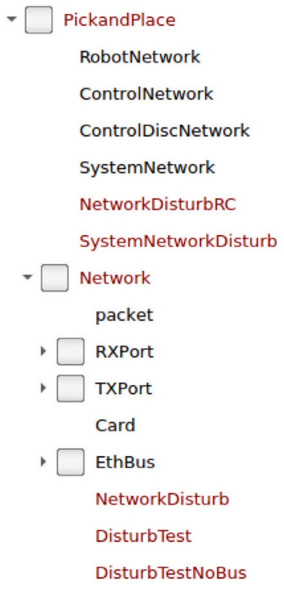


Fig. 5. Structure of the `PickAndPlace` package.

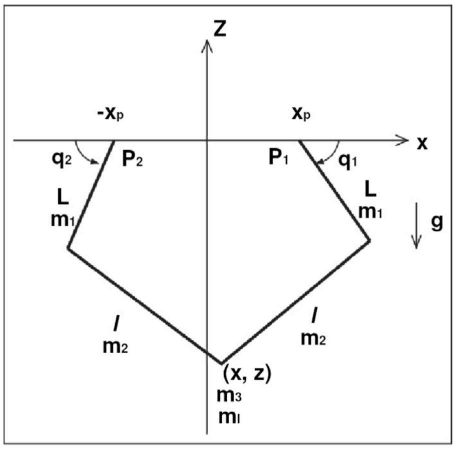


Fig. 6. Scheme of the robot [6].

The position (x, z) of the traveling plate depends on the angles q_1 and q_2, following Eqs. (1) and (2) (also written as $\phi(\mathbf{x}, \mathbf{q}) = 0$).

$$0 = (x - x_p - L\cos q_1)^2 + (z + L\sin q_1)^2 - l^2 \tag{1}$$

$$0 = (x + x_p + L\cos q_2)^2 + (z + L\sin q_2)^2 - l^2 \tag{2}$$

The dynamics of the robot are defined using Eqs. (3)–(11), that describe the torques and frictions of the system [43].

$$\boldsymbol{\Gamma} = \boldsymbol{\Gamma}_{red} + \boldsymbol{\Gamma}_{fric} + \boldsymbol{\Gamma}_{arm} + \boldsymbol{\Gamma}_{farm} + \boldsymbol{\Gamma}_{tplate} \tag{3}$$

$$\boldsymbol{\Gamma}_{red} = v^2(J_{mot} + J_{red})\ddot{\boldsymbol{q}} \tag{4}$$

$$\boldsymbol{\Gamma}_{fric} = sign(\dot{\boldsymbol{q}})F_s + F_v\dot{\boldsymbol{q}} \tag{5}$$

$$\boldsymbol{\Gamma}_{arm} = I\ddot{\boldsymbol{q}} - m_1\ g\cos\mathbf{q} \tag{6}$$

$$\boldsymbol{\Gamma}_{farm} = 0.5m_2L(L\ddot{\boldsymbol{q}} - g\cos\mathbf{q}) \tag{7}$$

$$0 = (\mathbf{J}_x^T\mathbf{J}_q^{-1})\boldsymbol{\Gamma}_{tplate} + (m_2 + m_3 + m_l)(\ddot{\mathbf{x}} + \mathbf{g}) \tag{8}$$

$$\mathbf{J}_q = \partial\phi/\partial\mathbf{q} \tag{9}$$

Table 1
Parameters of the robot.

Description	Notation	Value
P_1 x-coordinate	x_p	0.1 m
P_2 x-coordinate	$-x_p$	0.1 m
Arm length	L	0.3 m
Forearm length	l	0.7 m
Arm mass	m_1	0.82 kg
Forearm mass	m_2	0.14 kg
Tool mass	m_3	0.5 kg
Load mass	m_l	5 kg
Motor moment of inertia	J_{mot}	0.37×10^{-4} kg m^2
Gears moment of inertia	J_{red}	9.09×10^{-4} kg m^2
Arm moment of inertia	I	0.0188 kg m^2
Friction coef.	F_s	3 N m
Friction coef.	F_v	0.5 N m s
Gear reduction ratio	v	5
Gravity	g	9.81 m s^{-2}

$$\mathbf{J}_x = \partial\phi/\partial\mathbf{x} \tag{10}$$

$$\mathbf{g} = (0, -g) \tag{11}$$

where $sign(a) = 1$ if $a \geqslant 0$ and -1 otherwise. Γ_{red} are the torques due to inertia of the motor and gears, Γ_{fric} are the torques due to dry and viscous frictions between links, Γ_{arm} and Γ_{farm} are, respectively, the torques due to the inertia and weight of the arms and forearms, and Γ_{tplate} are the torques generated by the dynamics of the traveling plate.

Eqs. (1)–(11) can be directly coded in Modelica using the parameter values shown in Table 1 (cf. model `RobotNetwork` in Fig. 5).

The angles (q_1, q_2) are sampled at 1 kHz and sent to the controller using the Ethernet network. For this purpose, the robot model includes a `Card` model that is used to communicate with the network (detailed below).

6.2. Controller

The controller is used to control the position of the traveling plate of the robot. The desired position (x_d, z_d) is defined using Eqs. (12) and (13).

$$x_d(t) = -0.35\sin(\pi t/2) \tag{12}$$

$$z_d(t) = -0.7 + 0.1\cos(\pi t/2) \tag{13}$$

which, together with Eqs. (1) and (2), are used to calculate the reference trajectory for the angles $\boldsymbol{q}_d$.

The control is performed using a discrete PID controller with the robot angles $\boldsymbol{q} = (q_1, q_2)$ as inputs and the robot torques $\boldsymbol{\Gamma} = (\Gamma_1, \Gamma_2)$ as outputs.

Control signals are periodically computed every T_s interval. For each sample $n = 1, 2, \dots,$ control signals $\boldsymbol{\Gamma}_n$ are computed using Eqs. (14)–(17). Note that subscript n is added to indicate the corresponding sample number. The parameters of the controller are shown in Table 2. Similarly to the robot, the controller includes a `Card` model to communicate with the network. The model is implemented in the `ControlDiscNetwork` model (cf. Fig. 5).

$$\boldsymbol{e}_n = \boldsymbol{q}_{d,n} - \boldsymbol{q}_n \tag{14}$$

$$\boldsymbol{I}_n = \boldsymbol{I}_{n-1} + T_s\boldsymbol{e}_{n-1} \tag{15}$$

$$\boldsymbol{D}_n = \frac{\boldsymbol{e}_n - \boldsymbol{e}_{n-1}}{T_s} \tag{16}$$

$$\boldsymbol{\Gamma}_n = \max(-\alpha, \min(\alpha, K_c(\boldsymbol{e}_n + \frac{1}{T_i}\boldsymbol{I}_n + T_d\boldsymbol{D}_n))) \tag{17}$$

6.3. Network

The Ethernet network constitutes the cyber part of the system. Its implementation is encapsulated in the `Network` package (cf. Fig. 5), that includes:

- The `packet` model is used to represent the data sent through the network. Each packet includes the address of the node that sent the message, the address of its destination, the packet size, the role (i.e., the type of packet) and the payload.

Table 2
Parameters of the controller.

Description	Notation	Value
Proportional gain	K_c	2000 N m
Integral gain	T_i	300 s
Derivative gain	T_d	0.05 s
Sample time	T_s	0.001 s
Saturation	α	50 N m

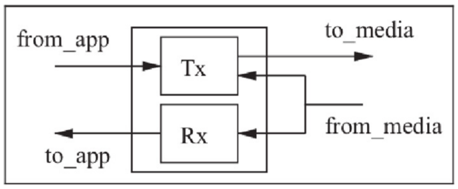


Fig. 7. Scheme of the Card model [6].

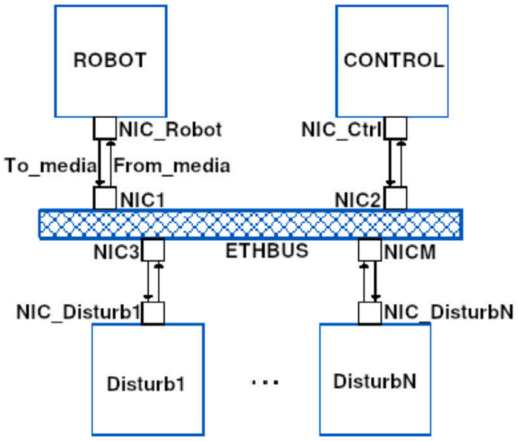


Fig. 8. Complete robot-controller system including disturbance nodes.

- The TX_port is used to transmit packets from the application level (e.g., the robot or the controller) to the network, and observe the state of the network (e.g., busy media, collisions, etc.). It is implemented as a PDEVS atomic model with two input ports (from_media and from_media) and one output port (to_media). These interface ports are used to communicate with the application level and with the media, represented by the Ethernet bus. This model implements the media access control (MAC) protocol used by Ethernet networks.
- The RX_port is used to receive completed packets from the network and transmit them to the applications. Analogously to TX_port, it is also implemented as a PDEVS atomic model with one input port (from_media) and one output port (to_app).
- The Card represents a Network Interface Controller (NIC). It is implemented as a PDEVS coupled model that includes one TX_port and one RX_port models and the four ports required to communicate with the media and the applications (cf. Fig. 7).
- The EthBus model is used to describe connections between multiple Card models. It broadcasts all packets received from one NIC to the rest of NICs in the bus. It is also implemented as an atomic PDEVS model.
- The NetworkDisturbRC model is used as a source of disturbances in the network. It is implemented as an application that generates packets with a time interval that follows an exponential random distribution with mean $1/125$ s by default. It is included to evaluate the effect of collisions in the network over the control of the robot.

6.4. Simulation results

The complete system is composed of a robot (RobotNetwork), a controller (ControlDiscNetwork), an Ethernet bus (EthBus), a variable number of NetworkDisturbRC models that introduce disturbances in the bus, and the couplings between

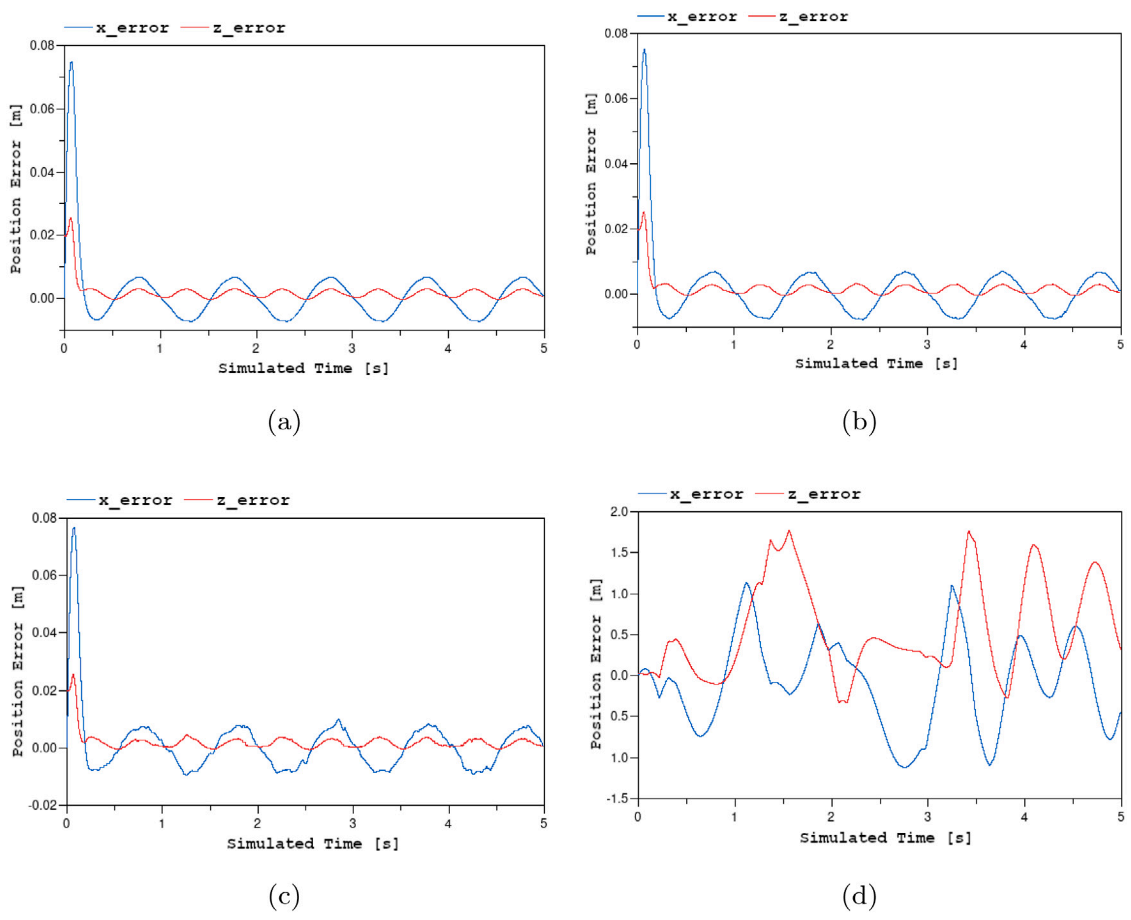


Fig. 9. Simulation results for the pick and place system: (a) 0 disturbance nodes, (b) 5 disturbance nodes, (c) 8 disturbance nodes, and (d) 10 disturbance nodes.

Table 3
Simulation results for Experiments 1A and 1B.

Experiment	Nodes	Execution time	Time events	Collisions	Packets
A (rand)	10	3.24 s	2098	0	1049
B (sync)	10	6.2 s	2638	5754	1000

all components (cf. Fig. 8). The system has been simulated during 5 s with a different number of disturbance nodes (0, 5, 8 and 10) in order to evaluate the impact of collisions in the control. The initial position of the traveling plate is set to $(x_0, z_0) = (0\,\mathrm{m}, -0.38\,\mathrm{m})$.

The simulation results are presented in Fig. 9. It can be observed that the quality of the control degrades with the increment of nodes transmitting over the network, due to delays introduced by packet collisions.

6.5. Simulation performance

Three experiments are discussed in order to demonstrate the performance of the simulated network.

- Experiment 1 demonstrates the impact of network collisions in the simulation. The system is composed of 10 disturbance nodes connected to the same Ethernet bus. Two cases are considered: (A) all nodes transmit their packets at random intervals, exponentially distributed with mean 1 and; (B) all nodes synchronously transmit 1 packet per second. Packet size is 100 B and the system has been simulated for 100 s. The simulation results are shown in Table 3. As expected, the second case shows a large amount of collisions due to the synchronized transmissions. Thus, the execution time and the number of time events triggered during the simulation are increased, due to the management of retries.
- Experiment 2 demonstrates how the performance of the simulation evolves when simulating a loosely congested network. In this experiment the system is composed of an increasing number of nodes, from 10 to 40, that transmit packets analogously

Table 4
Simulation results for Experiment 2.

Nodes	Execution time	Time events	Collisions	Packets
10	3.24 s	2098	0	1049
20	19.3 s	4155	7	2076
30	56.6 s	5942	0	2971
40	144 s	8072	0	4036

Table 5
Simulation results for Experiment 3.

Nodes	Execution time	Time events	Collisions	Packets
10	2.67 s	1461	652	527
20	36.4 s	8661	17900	971
30	86.4 s	10237	29611	1540

to experiment 1 A. The results of the simulations are presented in Table 4. Note that the number of packets and the number of time events evolve linearly with the size of the system. However, the execution time evolves exponentially because the time required to manage each event depends on the size of the system, since Modelica evaluates the whole model during each event.

- Experiment 3 represents a congested network. The system is also composed of an increasing number of nodes, from 10 to 30. In this case, the packet size is increased to the Ethernet MTU (1500 bytes), whose transmission time is around 1 ms, and the average transmission interval is set to 20 ms. The system is simulated for 1 s, in each case, and the simulation results are presented in Table 5. The system with 10 nodes transmits an average of 10 packets every 20 ms, that take 10 ms to be transmitted, without generating congestion in the network. However, systems with 20 and 30 nodes congest the network since the time required to transmit the packets is greater that the 20 ms interval between transmissions. This leads to a large increment in the number of collisions, time events and the execution time, in these cases.

7. Case study: Demo robot from the modelica standard library

This example is introduced to demonstrate the combination of the physical modeling paradigm and MPC for the description of CPS. The original system is composed of a robotic arm with six degrees of freedom that includes path planning, communication bus, axes and mechanics (cf. package `RobotR3` of the MSL [7]). In this case, the communication bus between the path planning and the 6 axis models has been replaced with an Ethernet network, using the model previously developed for the Pick and Place system. The structures of the original and modified models are shown in Fig. 10. The reference for the movement of the robotic arm is transmitted to the robot across the network, and simultaneously received by the multiple components of the robot. The description of the communication network does not require any additional intermediate models to transport the messages to the multiple destinations. Also, in this example, network packets are represented using an array of eleven Real numbers without requiring any modification in the library. Axis and mechanics models are described using an object-oriented modeling approach, as a combination of components

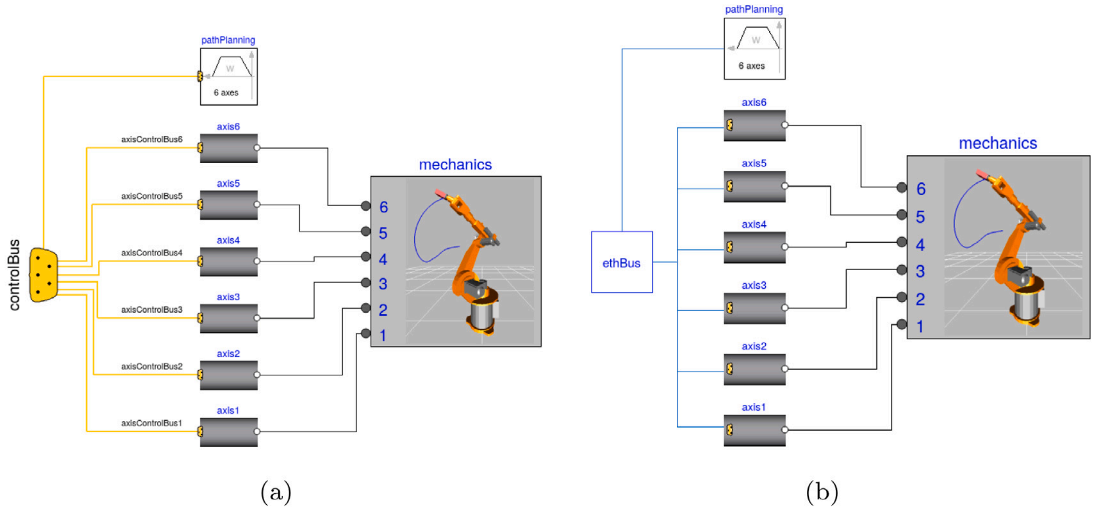
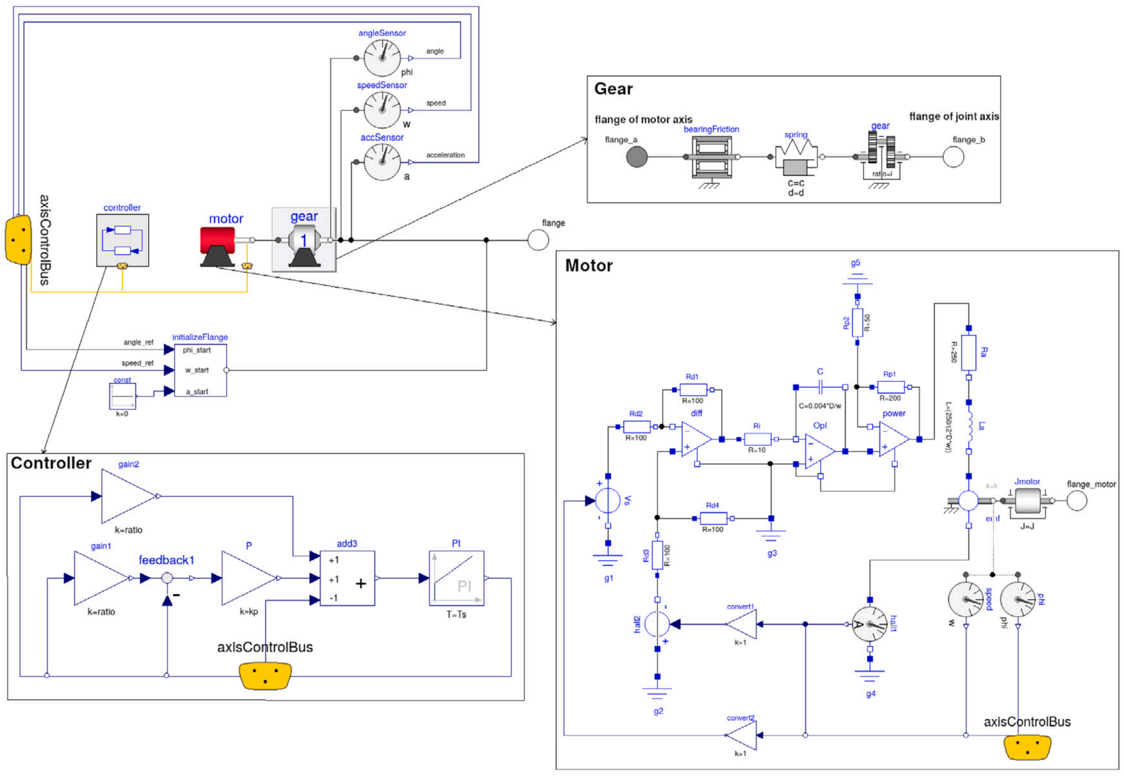
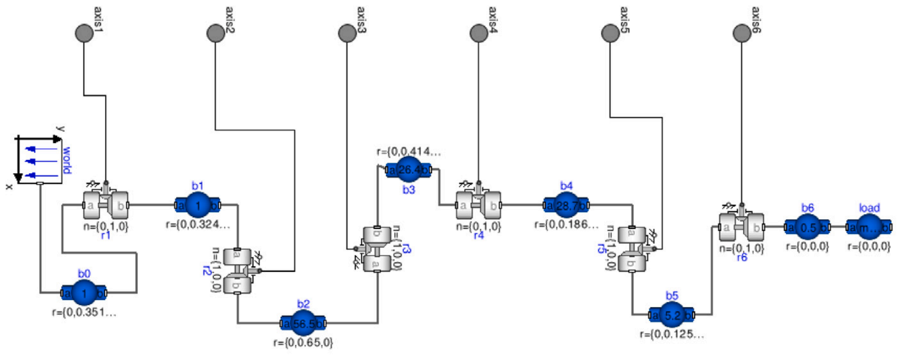


Fig. 10. Structure of the 3D robot: (a) original, and (b) modified models.



(a)



(b)

Fig. 11. Internal structure of the 3D robot components: (a) axis (including controller, motor and gear), and (b) mechanics.

from the MSL. The internal structure of the axis model, including the internal details of the controller, motor and gear, and the mechanics model are shown in Fig. 11.

In order to integrate the robot with the Ethernet network, the PathPlanning and Axis models also required modifications. An Ethernet NIC model, `Card`, is included in both models. Both models also include the required code to generate and send messages, in the case of the PathPlanning model, and to receive these messages and communicate them to the rest of the model, in the case of the Axis model.

The modified robot has been simulated using the same parameters as the original robot in order to compare both results. The path planning signals are sampled at 100 Hz and sent through the Ethernet network to the robot. The simulation of both robots is shown in Fig. 12, where the evolution of the angles of each axis is displayed. Note that in this case, since the introduction of

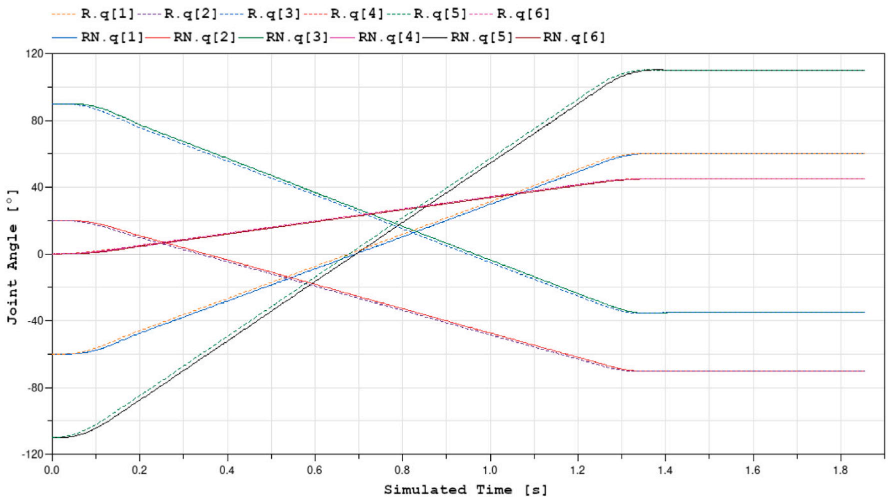


Fig. 12. Simulation results of the original ($R.q[X]$, dashed lines) and modified ($RN.q[X]$, solid lines) robots.

the Ethernet network does not include additional disturbances in the network the results for both robots is almost identical. This example serves as an illustration of how MSGLib models, and the proposed MPC mechanism, can be integrated with the rest of Modelica functionality and libraries, and in particular the Modelica Standard Library.

8. Conclusions and future work

An approach to support message passing communication in Modelica has been presented, together with its practical implementation in the form of the MSGLib library. This new library supports message passing communication and extends Modelica functionality to facilitate the description, for instance, of computer communication networks. This new functionality can be combined with the rest of Modelica and provides a powerful tool for the development of CPS models. The use of a single language, such as Modelica, to describe CPS facilitates the modeling task. Modelica and MSGLib have been shown as a well suited tool for modeling and simulation of CPS. MSGLib is freely distributed under the Modelica License 2 (cf. www.euclides.dia.uned.es), and has been developed and tested using Dymola 2021 [44].

The main difference and contribution of this work is that MSGLib provides additional functionality for the Modelica language, namely MPC communication and the description of dynamic data structures as buffers of messages. The presented proposal for MPC communication facilitates the description of models without the limitations present in the previous DEVSLib, SIMANLib and ARENALib libraries developed by the authors. In this new proposal, different types of messages can be described using arrays of Real numbers without further modifications to the library, no additional models are required to describe communication channels (e.g., for describing collective communication or loops in the structure of the model), and the messages are synchronously transmitted following the PDEVS communication approach. The proposed MPC communication approach can be used to model networking components, but it also can be used to describe multiple discrete-event models in Modelica (e.g., DEVS models, actor-oriented models, cellular automata, and agent-based models). Future work will include the development of a Modelica library of components to support different communication networks, such as Ethernet, CAN, PROFIBUS, etc.

Acknowledgment

This research was supported by the Ministerio de Economía *y* Competitividad of Spain, DPI2013-42941-R grant. UNED funding for open access publishing.

References

[1] R. Alur, Principles of cyber-physical systems, The MIT Press, Cambridge, MA, USA, 2015.
[2] I. Graja, S. Kallel, N. Guermouche, S. Cheikhrouhou, A.K. Kacem, A comprehensive survey on modeling of cyber-physical systems, Concurr. Comput. (2018) e4850, http://dx.doi.org/10.1002/cpe.4850.
[3] V. Sanz, A. Urquia, F.E. Cellier, S. Dormido, System modeling using the Parallel DEVS formalism and the Modelica language, Simul. Model. Pract. Theory 18 (7) (2010) 998–1018, http://dx.doi.org/10.1016/j.simpat.2010.03.004.
[4] V. Sanz, A. Urquia, F.E. Cellier, S. Dormido, Hybrid system modeling using the SIMANLib and ARENALib Modelica libraries, Simul. Model. Pract. Theory 37 (2013) 1–17, http://dx.doi.org/10.1016/j.simpat.2013.05.005.
[5] U. Pohlmann, M. Tichy, Modelica code generation from ModelicaML state machines extended by asynchronous communication, in: Proceedings of the 4^{th} International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, 2011, pp. 75–84.

[6] J. Nutaro, An extension of the OpenModelica compiler for using Modelica models in a discrete event simulation, Simulation 90 (12) (2014) 1328–1345, http://dx.doi.org/10.1177/0037549714554480.

[7] Modelica Standard Library (v3.2.3+build.3), URL https://github.com/modelica/ModelicaStandardLib, 2019.

[8] K.J. Åström, H. Elmqvist, S.E. Mattsson, Evolution of continuous-time modeling and simulation, in: Proceedings of the 12^{th} European Simulation Multiconference, ESM'98, Manchester, UK, 1998, pp. 9–18.

[9] H. Elmqvist, S.E. Mattsson, M. Otter, Modelica – A language for physical system modeling, visualization and interaction, in: Proceedings of the IEEE International Symposium on Computer-Aided Control System Design, CACSD'99, Kohala Coast, HI, USA, 1999, pp. 630–639.

[10] H. Elmqvist, F.E. Cellier, M. Otter, Object-oriented modeling of hybrid systems, in: Proceedings of the European Simulation Symposium, Delft, The Netherlands, 1993.

[11] M. Otter, H. Elmqvist, S.E. Mattsson, Hybrid modeling in Modelica based on the synchronous data flow principle, in: Proceedings of the 10^{th} IEEE International Symposium on Computer Aided Control System Design, 1999, pp. 151–157, http://dx.doi.org/10.1109/CACSD.1999.808640.

[12] F.E. Cellier, H. Elmqvist, M. Otter, J.H. Taylor, Guidelines for modeling and simulation of hybrid systems, in: Proceedings of the IFAC World Congress, Sydney, Australia, 1993.

[13] M. Association, Modelica – An unified object-oriented language for physical systems modeling. Language spec. v. 3.4, 2017, URL https://www.modelica.org/documents/ModelicaSpec34.pdf. (Accessed 8th February 2022).

[14] S.E. Mattsson, M. Otter, H. Elmqvist, Modelica hybrid modeling and efficient simulation, in: Proceedings of the 38^{th} IEEE Conference on Decision and Control, 1999, pp. 3502–3507.

[15] M. Otter, B. Thiele, H. Elmqvist, A library for synchronous control systems in modelica, in: Proceedings of the 9^{th} International Modelica Conference, 2012, pp. 27–36.

[16] Modelica Tools, Modelica free and comercial tools, 2022, URL http://www.modelica.org/tools.html (Accessed 8th February 2022).

[17] B.P. Zeigler, A. Muzy, E. Kofman, Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations, third ed., Academic Press, New York, 2018.

[18] W.D. Kelton, R.P. Sadowski, D.T. Sturrock, Simulation with Arena, fourth ed., McGraw-Hill, Inc., New York, NY, USA, 2007.

[19] F.E. Cellier, E. Kofman, Continuous System Simulation, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[20] J. Nutaro, P.T. Kuruganti, L. Miller, S. Mullen, M. Shankar, Integrated hybrid-simulation of electric power and communications systems, in: 2007 IEEE Power Engineering Society General Meeting, 2007, pp. 1–8, http://dx.doi.org/10.1109/PES.2007.386202.

[21] E. Kofman, S. Junco, Quantized-state systems: A DEVS approach for continuous system simulation, Trans. Soc. Comput. Simul. Int. 18 (3) (2001) 123–132.

[22] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K.-E. Årzén, How does control timing affect performance? Analysis and simulation of timing using jitterbug and TrueTime, IEEE Control Syst. Mag. 23 (3) (2003) 16–30, http://dx.doi.org/10.1109/MCS.2003.1200240.

[23] F. Wagner, L. Liu, G. Frey, Simulation of distributed automation systems in Modelica, in: Proceedings of the 6^{th} International Modelica Conference, 2008, pp. 113–122.

[24] P. Reuterswärd, J. Åkesson, A. Cervin, K.-E. Årzén, TrueTime network – A network simulation library for Modelica, in: Proceedings of the 7^{th} International Modelica Conference, 2009, pp. 657–662.

[25] D. Henriksson, H. Elmqvist, Cyber-physical systems modeling and simulation with Modelica, in: Proceedings of the 8^{th} International Modelica Conference, 2011, pp. 502–509.

[26] M. Askarpour, C. Ghezzia, D. Mandrioli, M. Rossi, C. Tsigkanos, Formal methods in designing critical cyber-physical systems, in: M.H. ter Beek, A. Fantechi, L. Semini (Eds.), From Software Engineering to Formal Methods and Tools, and Back: Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday, Springer International Publishing, Cham, 2019, pp. 110–130, http://dx.doi.org/10.1007/978-3-030-30985-5_8.

[27] J. Michael, D. Drusinsky, D. Wijesekera, Formal methods in cyberphysical systems, Computer 54 (09) (2021) 25–29, http://dx.doi.org/10.1109/MC.2021.3089267.

[28] M. Amrani, D. Blouin, R. Hinrich, A. Rensink, H. Vangheluwe, A. Wortmann, Towards a formal specification of multi-paradigm modelling, in: Proceedings of the 22^{nd} International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C, 2019, pp. 419–424.

[29] D. Broman, J.G. Siek, Modelyze: A Gradually Typed Host Language for Embedding Equation-Based Modeling Languages, Tech. Rep. UCB/EECS-2012-173, EECS Department, University of California, Berkeley, 2012, URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-173.html.

[30] M.W. Aziz, M. Rashid, Domain specific Modeling language for cyber physical systems, in: Proceedings of the 2016 International Conference on Information Systems Engineering, ICISE, 2016, pp. 29–33.

[31] B. Wood, A. Azim, Triton: A domain specific language for cyber-physical systems, in: Proceedings Fo the 22^{nd} IEEE International Conference on Industrial Technology, ICIT, vol. 1, 2021, pp. 810–816.

[32] A.T. Al-Hammouri, A comprehensive co-simulation platform for cyber-physical systems, Comput. Commun. 36 (2012) (2012) 8–19, http://dx.doi.org/10.1016/j.comcom.2012.01.003.

[33] S. Centomo, J. Deantoni, R. de Simone, Using SystemC cyber models in an FMI Co-simulation environment: Results and proposed FMI enhancements, in: Proceedings of the 2016 Euromicro Conference on Digital System Design, DSD, 2016, pp. 318–325, http://dx.doi.org/10.1109/DSD.2016.86.

[34] T. Nägele, J. Hooman, Co-simulation of cyber-physical systems using HLA, in: Proceedings of the 2017 IEEE 7^{th} Annual Computing and Communication Workshop and Conference, CCWC, 2017, pp. 1–6, http://dx.doi.org/10.1109/CCWC.2017.7868401.

[35] P. Palensky, A. van der Meer, C.D. Lopez, A. Joseph, K. Pan, Applied cosimulation of intelligent power systems: Implementing hybrid simulators for complex power systems, IEEE Trans. Ind. Electron. Mag. 11 (2) (2017) 6–21, http://dx.doi.org/10.1109/MIE.2017.2671198.

[36] I.C. Society, IEEE 1516-2010 – IEEE standard for modeling and simulation (M&S) high level architecture (HLA), 2010, URL https://standards.ieee.org/standard/1516-2010.html.

[37] Modelica Association Project FMI, Functional mock-up interface for model exchange and Co-simulation (v2.0), 2014, URL http://www.fmi-standard.org.

[38] H. Neema, J. Gohl, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, C. Sureshkumar, Model-Based Integration Platform for FMI Co-simulation and Heterogeneous Simulations of Cyber-Physical Systems, in: Proceedings of the 10^{th} International Modelica Conference, 2014, pp. 235–245.

[39] M.U. Awais, P. Palensky, W. Müller, E. Widl, A. Elsheikh, Distributed hybrid simulation using the HLA and the functional mock-up interface, in: IECON Proceedings, Industrial Electronics Conference, 2013, pp. 7556–7561.

[40] G. Liboni, J. Deantoni, A. Portaluri, D. Quaglia, R. de Simone, Beyond time-triggered Co-simulation of cyber-physical systems for performance and accuracy improvements, in: Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, ACM, New York, NY, USA, 2018, pp. 2:1–2:8, http://dx.doi.org/10.1145/3180665.3180668.

[41] A. Tolk, F. Barros, A. D'Ambrogio, A. Rajhans, P.J. Mosterman, S.S. Shetty, M.K. Traoré, H. Vangheluwe, L. Yilmaz, Hybrid Simulation for Cyber Physical Systems - A panel on where are we going regarding complexity, intelligence, and adaptability of CPS using simulation, in: Proceedings of the Symposium on Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems, MSCIAAS'18, 2018, pp. 1–19.

[42] A.C.H. Chow, Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator, Simulation 13 (2) (1996) 55–67.

[43] N.S. Nedialkov, N. Ramdani, Towards Integrating Hybrid DAEs with a High-Index DAE Solver, Tech. Rep. RR-6834, INRIA, 2009, inria-00360999.

[44] Dassault Systèmes AB, Dymola system engineering: Multi-engineering modeling and simulation based on Modelica and FMI, 2021, URL https://www.3ds.com/products-services/catia/products/dymola/.