Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

SIMULATION OF DECENTRALIZED COORDINATION STRATEGIES FOR NETWORKED MULTI-ROBOT SYSTEMS WITH EMERGENT BEHAVIOR-DEVS

Ezequiel Pecker-Marcosig[1,3], J. Francisco Presenza[3,4], Ignacio Mas[5], J. Ignacio Alvarez-Hamelin[4,6], Juan I. Giribet[5], and Rodrigo Castro[1,2]

[1]Instituto UBA-CONICET de Ciencias de la Computación, Buenos Aires, ARGENTINA
[2]Depto. de Computación, FCEyN, Univ. de Buenos Aires, Buenos Aires, ARGENTINA
[3]Lab. de Automática y Robótica, FI, Univ. de Buenos Aires, Buenos Aires, ARGENTINA
[4]Instituto UBA-CONICET INTECIN, Buenos Aires, ARGENTINA
[5]Lab. de Inteligencia Artificial y Robótica, Univ. de San Andrés and CONICET, Buenos Aires, ARGENTINA
[6]Facultad de Ingeniería, Univ. de Buenos Aires, Buenos Aires, ARGENTINA.

ABSTRACT

The design of distributed control strategies for multi-robot systems (MRS) relies heavily on simulations to validate algorithms prior to real-world deployment. However, simulating such systems poses significant challenges due to their dynamic network topologies and scalability requirements, where full inter-robot communication becomes computationally prohibitive. In this paper, we extend the applications of the Emergent Behavior DEVS (EB-DEVS) formalism by developing an agent-based model (ABM) to address key distributed control challenges in networked MRS. The proposed approach supports both direct and indirect interactions between agents (robots) via event messages and through macroscopic-microscopic states sharing, respectively. We validate the model using a challenging cooperative target-capturing scenario that demands dynamic multi-hop communication and robust coordination among agents. This complex use case highlights the strengths of EB-DEVS in managing asynchronous events while minimizing communication overhead. The results demonstrate the formalism's effectiveness in supporting decentralized control and simulation scalability within a hierarchical micro-macro modeling framework.

1 INTRODUCTION AND RELATED WORKS

The growing complexity of autonomous missions for mobile robots has led to the adoption of networked multi-robot systems (MRS), due to their advantages in adaptability, coverage, robustness, and scalability across a wide range of applications. MRS are typically employed to tackle complex tasks by dividing them into simpler sub-tasks executed by each robot. In addition, many applications demand keeping some structural properties, such as network connectivity or rigidity, while avoiding inter-robot collisions. These constraints require the implementation of coordination strategies between robots, commonly achieved through the use of onboard control systems and peer-to-peer communication. The development of distributed controllers for these applications demands efficient design methodologies to accelerate development cycles. Heuristic approaches, such as trial-and-error methods applied directly to operational systems, can incur substantial costs and safety risks—particularly in safety-critical contexts (Hentati et al. 2018). Alternatively, analytical techniques are often limited by the complexity of the underlying models, requiring significant simplifications. As a result, numerical experimentation becomes essential for simulating such systems.

In the control community, MRS are typically modeled as graphs, where nodes represent robots and edges denote communication links or sensing relationships between them. As observed by (Testa et al. 2021), robotics simulators frequently oversimplify the communications dynamics between agents or consider unrealistic full mesh topologies (leading to very compute-intensive models). In addition, due to limited communication ranges and sensing constraints, the topology of multi-robot networks evolves over time as the robots move across space to perform a given task. Therefore, any attempt to model such systems must be able to jointly represent multiple robots communicating over arbitrary dynamic networks. A simple approach to simulate dynamic topologies is to consider all-to-all links that can be enabled/disabled during the simulation, as in Hu et al. (2005). Testa et al. (2021) presents a more efficient graph-based approach, where it is left to the agents to explicitly know the neighbors with which they are connected at a given time.

Existing simulation platforms for cooperative MRS typically use co-simulation to integrate inter-robot communication aspects with the simulation of vehicle physics, due to the fundamentally different nature of these two types of dynamics (Calvo-Fullana et al. 2021; Acharya et al. 2023) (and references therein). However, the necessary synchronization mechanisms between the clocking and robots' states of these tools introduce complexity and potentially loss of performance. An alternative approach is to develop a unified simulation tool capable of representing all the dynamics within the same framework. The most straightforward solution takes the form of application-specific discrete-time simulators, which are often used for preliminary validation of control strategies. This approach provides the advantage of rapid deployment and ease of use. However, the resulting ad hoc simulators are typically non-reusable, entangle the model specification with the simulation engine, and struggle to handle discrete events over a continuous time axis.

The Discrete-EVent System specification (DEVS) formalism (Zeigler et al. 2018) has proven successful for the modeling and simulation (M&S) of hybrid dynamic systems under a unified framework. DEVS has been used in a wide variety of applications involving robots and autonomous mobile systems, ranging from pure discrete models for the robot and the controllers (Moallemi and Wainer 2013) to complex hybrid and non-linear models (Pecker-Marcosig et al. 2020; Bordón-Ruiz et al. 2021), and multi-robot systems (Hu et al. 2005). In addition, discrete-event models (with a continuous-time base) inherently capture stochastic phenomena—such as robot clock drifts and path-dependent propagation delays in peer-to-peer communications—that are otherwise extremely difficult to deal with in conventional frameworks based on time discretization. DEVS models have the additional advantage that they are built in a modular and hierarchical fashion, fostering code adaptability and reusability, a key aspect for the development of model libraries. Additionally, the strict separation between the models and the underlying simulator algorithm fostered by DEVS ensures that a modeler does not need to worry about the overall orchestration of events produced by each component of the system, as this is a responsibility of the abstract simulator. This approach clearly contrasts with the use of custom simulation code where there is no separation between model and simulator.

In the interest of scaling the distributed controller algorithms to larger networks, we favor using a simulation framework specifically built for complex systems. Various approaches have been developed within the DEVS formalism to handle models with dynamically changing structures. Among them, the recently introduced EB-DEVS formalism (Foguelman et al. 2021) is specifically targeted to deal with complex systems, which exploits a microscopic-macroscopic communication mechanism offered by the simulator for agents to interact via global states (complementary to using explicit communication channels) helping to reduce the number of connections and consequently the number of exchanged events.

Following this discussion, this paper presents a unified simulation framework where models are specifically designed for simulating complex networked MRS, leveraging the capabilities of EB-DEVS. This tool aims to balance the strengths of the existing approaches by combining ease of use with the ability to model complex multi-robot behaviors. To deal with dynamic communication networks, we extend the ideas proposed in (Testa et al. 2021) by creating an atomic model that represents the transmission medium, which is responsible for routing messages from each robot to its neighbors according to their physical state—positions, orientations, etc.—and sensor specifications—range, visual range, etc. This allows the system to mimic the real-world behavior of robots broadcasting messages without explicit knowledge of their neighbors. The work is organized as follows. Section 2 provides the background on the EB-DEVS formalism and its simulation toolkit. Section 3 presents the library of EB-DEVS models for robotic applications, while Section 4 makes use of them in a real application involving a decentralized controller for a MRS. Finally, Section 5 concludes the paper.

2 THE EMERGENT BEHAVIOR DEVS (EB-DEVS) FORMALISM AND SIMULATION TOOLKIT

The Emergent Behavior DEVS (EB-DEVS) formalism, recently introduced by Foguelman et al. (2021), extends Classic DEVS (Zeigler et al. 2018) to model, simulate, and identify emergent properties. It incorporates multi-level feedback loops between macro and micro-states, allowing parent models to exchange values that are a function of their global states with their children models (and vice versa), enabling indirect communication via upward and downward channels. These channels support information flow while preserving the DEVS's internal state-hiding principle. EB-DEVS remains fully interoperable with standard DEVS models.

An *Atomic* EB-DEVS model is defined formally as $M_A = \langle X, Y, S, \delta_{int},\ \delta_{ext}, \lambda, ta, Y_{up}, S_{macro}\rangle$ where X is the set of input events, Y is the set of output events, S is the set of state values, Y_{up} is the set of output events directed to the parent model, and S_{macro} is the set of parent's states. Four dynamic functions define the behavior of the model: $\delta_{int} : S \times S_{macro} \rightarrow S \times Y_{up}$ (*internal transition*, for autonomous behavior in absence of external events), $\delta_{ext} : AtomicS \times \{Rformally\}_0^+ \times X \times S_{macro} \rightarrow S \times Y_{up}$ (*external transition*, for reactive behavior after receiving external events), $\lambda : S \rightarrow Y$ (*output*, for emitting output events) and $ta : S \rightarrow \mathbb{R}_0^+ \cup \infty$ (*time advance*, for the state duration). Two major changes are introduced in the EB-DEVS atomic models compared to Classic DEVS: (a) the state transition functions δ_{int} and δ_{ext} can use the parent's model state S_{macro} as a parameter for state changes, and (b) they can communicate their outputs to the parent model through Y_{up}, which may use these outputs to compute δ_G (defined below).

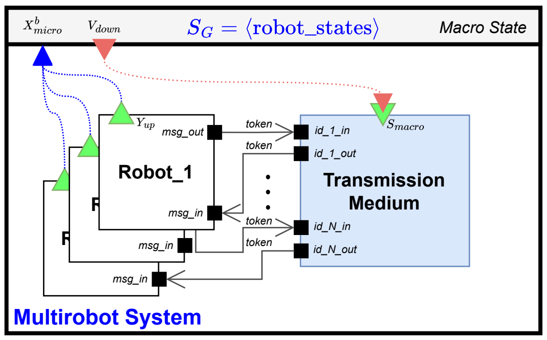
A *Coupled* EB-DEVS model is an ensemble of interconnected models, and is defined formally as $M_C = \langle X_{self}, Y_{self}, D, \{M_d\}, \{I_d\}, \{Z_{d,j}\}, Select, X^b_{micro}, Y_{G_{up}}, S_{G_{macro}}, S_G, V_{down}, \delta_G\rangle$ where $self$ is the coupled model itself, X_{self} and Y_{self} are the sets of input and output values of the coupled model (respectively), D is the dictionary of connected components belonging to $self$, M_d ($d \in D$) is any other model, coupled or atomic. For each $d \in D \cup \{self\}$, I_d is the set of influencees models of subsystem d. For each $j \in I_d$, $Z_{d,j} : Y_d \rightarrow X_j$ is the d to j translation function, while $Select : 2^D \rightarrow D$ is a tie-breaking function for simultaneous events. X^b_{micro} is a mailbox input port for the information events sent by the children atomic models (via δ_{int} and δ_{ext}). $Y_{G_{up}}$ is an output port for the information events sent towards its parent model. EB-DEVS coupled models have their own global state s_G, with S_G representing the set of states of all possible global states. $V_{down} : S_G \rightarrow S_{macro}$ is the *downward value coupling* function that provides the global information to its children. The global transition function $\delta_G : S_G \times \mathbb{R}_0^+ \times X^b_{micro} \times S_{G_{macro}} \rightarrow S_G \times Y_{G_{up}}$ computes a new global state $s_G \in S_G$ based on its own state, the time elapsed in its last state, the messages X^b_{micro} arrived from its micro components and its parent's macro state $S_{G_{macro}}$. It also computes the upward-causation event (a value in the $Y_{G_{up}}$ set) towards its parent. The cascade of upward-causation events can eventually climb up in the system hierarchy, possibly (but not necessarily) until the topmost (root) coupled model.

In this work, the library of models for multi-robot networks (Section 3) as well as the simulations presented (Section 4) were developed using the EB-DEVS simulator (Foguelman, D.J. 2022) which was designed using the abstract simulator for EB-DEVS (Foguelman et al. 2021). The latter, which defines how EB-DEVS models are executed, was implemented as an extension of the PythonPDEVS (Van Tendeloo and Vangheluwe 2015) simulation toolkit.

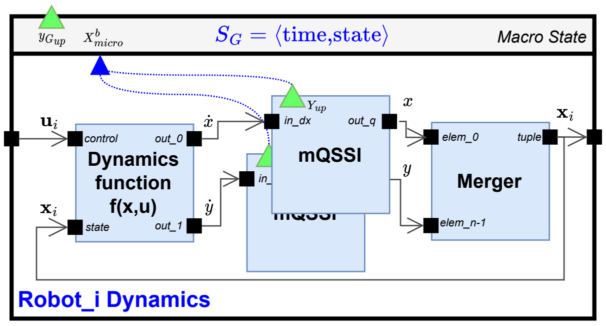
3 EB-DEVS LIBRARY FOR MULTI-ROBOT NETWORKS

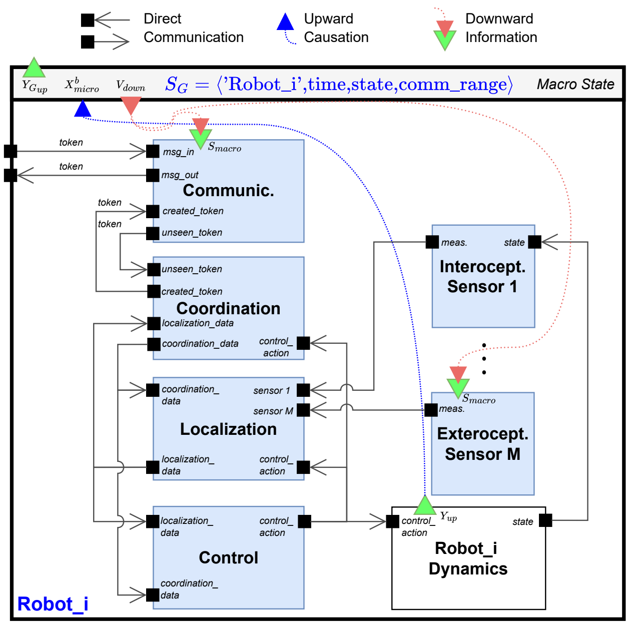
This section presents an extensible library comprising EB-DEVS atomic and coupled models, aimed at supporting applications involving MRS and distributed control strategies (SEDLab and LAR 2025). These models were designed with modularity in mind, allowing them to be used out-of-the-box for constructing a wide range of complex applications, leveraging the inherent modular structure of DEVS. We adopted an ABM approach, where each robot is an agent that executes a local set of rules and interacts with neighboring agents to exchange information. The agents' models include the robots' physics, the controller algorithms that execute the control rules, the communication modules that exchange data, and the measurement and/or estimation of the robot state.

Figure 1 illustrates the modular and hierarchical composition of EB-DEVS models. *Atomic* EB-DEVS models are represented with light-blue boxes and *Coupled* EB-DEVS models with white boxes (with the topmost bar in light gray showing their *global state* S_G between angle brackets). The small black squares on sides of the boxes represent input and output ports for direct communication connected by solid arrows, while upward and downward facing triangles (linked with dotted lines) represent indirect communication channels (Y_{up} and V_{down} for atomics, and X^b_{micro} and $Y_{G_{up}}$ for coupled). Notice that a DEVS model is a particular case of an EB-DEVS model. Therefore, in what follows, DEVS models will be presented as EB-DEVS models that simply ignore any upward/downward channels.



(a) Multi-robot system (root coupled model).





(b) Robot i Dynamics (coupled model).

(c) Robot i (coupled model).

Figure 1: EB-DEVS library for multi-robot networks. Figures (b), (a) and (c) illustrate the hierarchical composition of the models, progressing from micro to macro-level structures. Blue-light boxes represent atomic EB-DEVS models, while white boxes represent coupled EB-DEVS models.

3.1 EB-DEVS Coupled Models

3.1.1 Multi-Robot System

The *MultiRobotSystem* coupled model (Figure 1(a)) is the top coupled model illustrating the complete architecture. It consists of N instances of the *Robot* coupled model (Section 3.1.2), each with bidirectional links to the *Transmission Medium* atomic model (Section 3.2.9). The *MultiRobotSystem*'s global state S_G, updated upon receiving information from the *Robots* through the $Y_{G_{up}}$ ports, is the dictionary

$$S_G = \langle \ldots, \langle \texttt{<id>}, \texttt{<time>}, \texttt{<state>}, \texttt{<comm\_range>} \rangle, \ldots \rangle,$$

where `id` identifies the robot, `time` is the time stamp, `state` comprises the *QSS* polynomials representing the dynamical state of the robot (Section 3.1.3), and `comm_range` is the communication range. During initialization, the *MultiRobotSystem* loads configuration parameters from some `json` files containing individual dictionaries for each robot.

3.1.2 Robot

The *Robot* coupled model (Figure 1(c)) includes atomic models for subsystems typically used in multi-robot applications. This coupled model has one input port (`msg_in`) and one output port (`msg_out`) indicating the ports for communication with other robots. The global state of the *Robot* coupled model, updated when receiving information from *Robot Dynamics* (Section 3.1.3) through the Y_{Gup} ports, is the dictionary

$$S_G = \langle \texttt{<id>}, \texttt{<time>}, \texttt{<state>}, \texttt{<comm\_range>} \rangle.$$

A generic *Robot* configuration is shown in Figure 1(c), including the coupled model *Robot Dynamics*, and atomic models like *Localization*, *Control*, *Coordination*, *Communication*, *Interoceptive Sensor*, and *Exteroceptive Sensor* (Sections 3.2.1 through 3.2.9).

3.1.3 Robot Dynamics

The *Robot Dynamics* coupled model (Figure 1(b)) is used to numerically solve the ordinary differential equation that models the robot dynamics, $\dot{\mathbf{x}}_i(t) = f_i(\mathbf{x}_i(t), \mathbf{u}_i(t))$, where $\mathbf{x}_i \in \mathbb{R}^{d_x}$ denotes the state of the robot i, $\mathbf{u}_i \in \mathbb{R}^{d_u}$ its control input and $f_i : \mathbb{R}^{d_x} \times \mathbb{R}^{d_u} \to \mathbb{R}^{d_x}$ the dynamics function. This model includes the *Dynamics Function* atomic (Section 3.2.1), d_x *multilevel QSS Integrators (mQSSI)* (Section 3.2.2), a *Merger* atomic model (Section 3.2.3) and the appropriate interconnections. The *Robot Dynamics*'s global state, updated when receiving information from the *mQSSI* through the Y_{Gup} ports, is the dictionary

$$S_G = \langle \texttt{<time>}, \texttt{<state>} \rangle,$$

where `<state>` comprises one *QSS* polynomial segment for each of the d_x state variables.

3.2 EB-DEVS Atomic Models

The atomic models in this library are designed to closely represent common components present in networked MRS. Their design includes the typical required inputs and outputs, which can be modified to suit the modeler's needs. In addition, they provide *hooks* for calling external Python libraries during both internal and external transitions. This design choice fosters re-usability and adaptability, which are key for developing different control strategies. To simplify the following description, those atomic models that start passivated (time advance equal to infinity) and trigger an output event in zero time after receiving an external event, and then return to passive, are called *reactive*.

3.2.1 Dynamics Function

The *Dynamics Function* atomic model is a reactive model that implements the mapping $f_i : \mathbb{R}^{d_x} \times \mathbb{R}^{d_u} \to \mathbb{R}^{d_x}$. It includes two input ports (`state` and `control`) and d_x output ports (`out_<i>`) where it emits d_x simultaneous outputs events carrying one element of the state derivative vector $\dot{\mathbf{x}}_i$.

3.2.2 QSS Integrator and multilevel *QSS* Integrator

The family of Quantized State System integration methods (*QSS*) can naturally be represented using DEVS (Kofman and Junco 2001). The integration result x is approximated by a sequence of polynomial segments, each valid between consecutive events. The *QSS* Integrator DEVS atomic model (*QSSI*) has one input port `dx` for the time derivative $\dot{x}$, and one output port `q` for the quantized *QSS* polynomial q that approximates the time evolution of x. In this work, we present the multilevel *QSS* Integrator (*mQSSI*) for EB-DEVS, an extension of the *QSSI* in which every change in the atomic state is communicated to the parent model via the Y_{up}. This event carries both the polynomial for the quantized state q and the current time t. The *QSSI* and *mQSSI* atomic models have four parameters: the relative quantum `dQRel`, the minimum quantum `dQMin`, the input gain `gain` and the initial condition `x0`.

3.2.3 Splitter and Merger

The complementary *Splitter* and *Merger* atomic models perform reactive splitting and merging of lists of any data type. A *Splitter* receives a tuple with n elements on its `tuple` port and emits n simultaneous outputs on ports `elem_<i>` ($i = 0,\dots,n-1$), each carrying one element. Conversely, a *Merger* has n input ports (`elem_<i>`) and one output port (`tuple`). It maintains a tuple `data` indexed by port, updates the relevant entry upon receiving an input, and emits the complete tuple.

3.2.4 Interoceptive and Exteroceptive Sensors

Typical sensors found onboard robots can be classified as interoceptive and exteroceptive. The *Interoceptive Sensor* atomic model aims to represent various types of sensors that measure variables that depend only on the robot's internal state, such as accelerometers, gyroscopes, and odometers. This atomic model has one input port (`state`) for receiving the list of *QSS* polynomials corresponding to the d_x state variables. The atomic model periodically schedules an internal transition to simulate a new measurement evaluating the *QSS* polynomials. which are valid between consecutive input events, therefore they can be evaluated at any time.

The *Exteroceptive Sensor* atomic model aims to represent various types of sensors that perceive parameters which depend on external references, such as GNSS receivers, magnetometers, cameras, and lidars. This atomic model possesses no input; instead, it simulates measurements by periodically retrieving information from *MultiRobotSystem*'s global state S_G via the V_{down} function.

Both sensor atomic models have one output port (`measurement`) for emitting events carrying the scalar valued measurements corrupted with white Gaussian noise. Three configuration parameters are required: the measurement period (`<sensor_id>_period`), the noise's mean value (`<sensor_id>_bias`) and covariance matrix (`<sensor_id>_covariance`).

3.2.5 Localization

The *Localization* atomic model is designed to implement information fusion algorithms that combine data from multiple sources at different rates such that each robot estimates its own state. A generic specification of this atomic model has $M+2$ input ports, where M is the number of onboard sensors (see Figure 1(c)). Sensor measurements are received via M inputs (`<sensor_id>`). Another input corresponds to `coordination_data`, involving information received from the other robots which, depending on the coordination strategy, might be other robots' estimated states or other relevant data. The remaining input is `control_action` which contains the robot's last computed control action. This model has one output port (`localization_data`) where output events are emitted containing the robot's own state estimation possibly accompanied by additional data such as uncertainty measures.

3.2.6 Control

The *Control* atomic model provides the essential tools to implement control strategies for MRS. It is intended to contain the implementation of the control law from two inputs: `localization_data` containing the state estimate, and `coordination_data` that incorporates relevant information communicated by other robots. Based on these values, this atomic model periodically executes the control law and schedules an internal transition in zero time to emit the control action through its output port (`control_action`).

This atomic model takes the execution period (`control_period`) as a configuration parameter. With this generic atomic model, a wide variety of multi-robot control schemes can be implemented. If needed, when handling complex control operations, such as path planning, optimization solvers, etc., this atomic model can be replaced by the modular composition of simpler atomic models in a coupled model that maintains the same external interface (inputs and output ports).

3.2.7 Coordination

This atomic model is intended to encapsulate the coordination strategy among robots. The coordination among robots is assumed to be based on the exchange of *tokens*. This atomic has three input ports (`unseen_token`, `localization_data` and `control_action`); and two output ports (`coordination_data` and `created_token`), see Figure 1(c). The particular way of creating and processing tokens depends on the application. However, all tokens are considered to be of the form

$$\langle \texttt{<id>},\ \texttt{<type>},\ \texttt{<order>},\ \texttt{<data>},\ \texttt{<max\_hops>},\ \texttt{<hops>} \rangle, \tag{1}$$

where `id` identifies the source robot, `type` indicates the type of information it carries, `order` determines the order in the sequence of tokens emitted by the source, `data` carries the relevant information or *payload*, `max_hops` is the maximum number of hops that it must traverse, and `hops` the number of hops already traversed. All fields must remain immutable through the token's propagation except for `hops`, which starts at 1 and gets incremented each time it is forwarded. In Section 4, we describe the token handling in detail for the proposed case study.

3.2.8 Communication

The *Communication* atomic model is intended to contain the logic that manages peer-to-peer communications (see Figure 1(c)). This reactive atomic has two input ports (`msg_in` and `created_token`); two output ports (`unseen_token` and `msg_out`), see Figure 1(c); and a configuration parameter, `forward`, a boolean flag that determines whether the robot forwards received tokens. This atomic keeps, as part of the atomic's internal state, a double-key dictionary $\langle (\texttt{<id>},\ \texttt{<type>}) = \texttt{<order>} \rangle$ that identifies received tokens by source and type. This dictionary is used to check if a received token has not been seen before, since forwarded tokens might arrive multiple times. Each time a token (1) arrives at a robot via `msg_in`, the *Communication* module outputs it through `unseen_token` if it has not been seen before; otherwise, the token is discarded. The token is forwarded through `msg_out` if `hops` < `max_hops` and `forward` = `true`. The *Communication* atomic model includes the capability to store communication metrics (`comm_metrics`) when exchanging tokens with other robots. This feature can be used to simulate the measurement of indicators such as Time-of-Arrival (TOA) or Received Signal Strength Indicator (RSSI), which can be used, for example, to estimate inter-robot distances. This is implemented by retrieving information from *MultiRobotSystem*'s global state S_G via the V_{down} function upon the reception of a token.

3.2.9 Transmission Medium

The *Transmission Medium* atomic model simulates the physical environment through which inter-robot communication occurs. It is designed to enable peer-to-peer communication that adapts to the dynamic topology of the multi-robot network (see Figure 1(a)). This reactive atomic has N input ports `<id>` and N output ports `<id>`. When the *Transmission Medium* model receives an input event carrying a token from a robot, it is routed to the neighboring robots through the corresponding output ports by scheduling multiple output events in zero time. The event value consists of the transmitter's `id` and the `token`. To determine the set of neighbors, the *Transmission Medium* queries the downward value coupling function $V_{down} : S_G \rightarrow S_{macro}$. This function checks the global state S_G of the *Multi-Robot System*, which contains the polynomials that approximate each robot's state $\mathbf{x}_i(t)$ at the current time.

Observe that in the absence of a global state, DEVS models would demand connecting all the robots with each other, which requires $O(N^2)$ links, where N is the total number of robots, affecting the scalability of this simulation scheme. For example, (Hu et al. 2005) leverages the Dynamic Structure DEVS formalism to dynamically enable/disable links between robots based on their distances. In contrast, the presented library connects all robots with the *Transmission Medium* atomic using bidirectional static links. This approach requires $O(N)$ links, reducing the number of events and resulting in improved simulation performance.

4 CASE STUDY: RIGIDITY CONTROL OF A NETWORKED MULTI-ROBOT SYSTEM

There exist different types of distributed control strategies for MRS that require the intensive exchange of data between robots, where this library can be applied. This section describes a challenging application involving a multi-robot network of $N = N_h + N_t$ robots engaged in a cooperative task: a group of N_h mobile robots, called hunters, aims to capture N_t stationary targets that are randomly distributed throughout the environment. While targets have known positions, hunters must estimate their own locations using distance measurements from each other. To enable distance-based localization, hunters communicate with each other, forming a network where links serve both for message exchange and distance measurements. Distance-based localization is possible if the hunter's communication network preserves the structural property of *rigidity*, see (Anderson et al. 2008). This approach is common in multi-robot control strategies, particularly useful in GPS-denied environments such as underwater or indoor settings.

The cooperative mission considered assigns each hunter the goal of approaching the nearest uncaptured target, resulting in trajectories that naturally lead to the dispersion of hunters. In this work, we assume that the communication network among the hunters is determined by their relative distances and their `comm_range`. As a consequence, inter-hunter links and therefore rigidity might be lost, leading to the failure of the distance-based localization scheme. Therefore, a decentralized rigidity maintenance controller that relies on local interactions between neighboring hunters is required. To this end, we adopt the control strategy proposed in Presenza et al. (2022a) and Presenza et al. (2022b). This control strategy requires a multi-hop exchange of tokens, configuring a challenging scenario that helps to evaluate the network capabilities of the simulation library. The coupled models for the targets and the hunters are described as follows.

4.1 Targets

A *Target* model is an instance of a *Robot* (Section 3.1.2) that includes the models *Robot Dynamics*, *Coordination*, *Communication* and *Control*. The dynamical system of a target is defined by its position $\mathbf{x}_i = \mathbf{p}_i \in \mathbb{R}^2$ satisfying $\dot{\mathbf{p}}_i = 0$ (i.e. remain stationary). Hence, *Robot Dynamics* model includes a *Dynamics Function* atomic model implementing $f_i(\mathbf{x}_i, \mathbf{u}_i) = 0$ and two *mQSSI* models for the x and y coordinates. The *Communication* model has the `forward` flag set to `false`. The *Coordination* model incorporates an additional flag called `status` that is initially set to `active`. A *Control* model, with no inputs, periodically sends empty messages to *Coordination* (at `control_period`) through `control_action`. When receiving these messages, the latter issues a `status` token, with fields `type` and `data` filled with the target's status (flag `status`) and position

$$\langle \texttt{id} = i,\ \texttt{type} = \textbf{status},\ \texttt{<order>},\ \texttt{data} = \mathbf{p}_i,\ \texttt{max\_hops} = 1,\ \texttt{hops} = 1 \rangle, \tag{2}$$

which is sent by *Communication*. These tokens are received by the *Transmission Medium* which delivers them to all targets and hunters within a predefined communication range `comm_range_<id>` $\in \mathbb{R}^+$. When *Communication* receives a token from a hunter (Section 4.2), it measures the distance to it as `comm_metrics` and sends it along with the token to the *Coordination*. If the distance is less than a predefined range `capture_range_<id>` $\in \mathbb{R}^+$, the target is considered hunted and the `status` flag is changed to `inactive`. This flag remains `inactive` for the rest of the mission, during which it continues emitting the token in (2) with status set to **`inactive`**.

4.2 Hunters

A *Hunter* model is an instance of a *Robot* that includes the models *Robot Dynamics*, *Coordination*, *Communication*, *Localization*, *Control*, and *Interoceptive Sensor*. The dynamical system of a hunter is defined by its position $\mathbf{x}_i = \mathbf{p}_i \in \mathbb{R}^2$ satisfying $\dot{\mathbf{p}}_i = \mathbf{u}_i$, i.e., it is controlled through velocity commands. Hence, *Robot Dynamics* model includes a *Dynamics Function* atomic model implementing $f_i(\mathbf{x}_i, \mathbf{u}_i) = \mathbf{u}_i$ and two *mQSSI* models for the x and y coordinates. The *Communication* model has the `forward` flag set to `true`. All hunters include an *Odometry Interoceptive Sensor* that periodically triggers linear velocity measurements $\tilde{\dot{\mathbf{p}}}_i$. Also, to enable absolute position estimation, 2 hunters are equipped with an *GPS Exteroceptive Sensor* which periodically outputs the measurement $\tilde{\mathbf{p}}_i$.

The *Localization* model produces the estimate $\hat{\mathbf{p}}_i$ using a distributed Kalman-filter-based localization algorithm (Presenza et al. 2022a) that requires the aforementioned $\tilde{\dot{\mathbf{p}}}_i$ (and $\tilde{\mathbf{p}}_i$ if available) along with distance measurements with the 1-hop neighbors $\{\tilde{z}_{ij} : j \in \mathcal{N}_i\}$ and their estimated positions $\{\hat{\mathbf{p}}_j : j \in \mathcal{N}_i\}$. The value of $\hat{\mathbf{p}}_i$ is updated and outputted through `localization_data` (Figure 1(c)) each time a new measurement is obtained.

The i^{th} hunter's *Control* model has three objectives: (a) commanding the robot to pursue the nearest target, (b) avoiding collisions with its hunter neighbors, and (c) maintaining the network's rigidity. In order to achieve (a), *Coordination* maintains a variable `nearest_target` $= \langle$`<id>, <dist>`$\rangle$, representing the id and the distance to the nearest active target. This pair is initialized as $\langle$`none`, `infinity`$\rangle$; `dist` is updated whenever the hunter receives a token from its nearest target, and `id` when a token is received from a closer active target. After each update, hunter's *Coordination* shares the nearest target position (included in token (2)) to hunter's *Control* through `coordination_data` (Figure 1(c)) which computes the action $\mathbf{u}_{i,T}$, to attract the hunter i to the nearest target. For control objectives (b) and (c), the controller of i needs the estimated positions of each $j \in \mathcal{H}_i$, defined as the set of hunters within η_i (a configuration parameter) hops from the i^{th}-robot, see (Presenza et al. 2022a). This control action is updated at a fixed rate determined by `control_period` and outputted by scheduling an internal transition in zero time.

The following part details the configuration of hunter's *Coordination*, required to implement the control and localization strategy in a decentralized fashion. To do so, each robot issues two tokens (`state` and `action`) at each control period, directed by the controller (when an input event in *Coordination* model occurs in the `control_action` input). The tokens' content is defined as follows

$$\begin{aligned} &\langle \texttt{id} = i,\ \texttt{type} = \textbf{\texttt{state}},\ \texttt{<order>},\ \texttt{data} = \hat{\mathbf{p}}_i,\ \texttt{max\_hops} = \sigma_i,\ \texttt{hops} = 1\rangle, \\ &\langle \texttt{id} = i,\ \texttt{type} = \textbf{\texttt{action}},\ \texttt{<order>},\ \texttt{data} = \{\mathbf{u}^j_{i,R} : j \in \mathcal{H}_i\},\ \texttt{max\_hops} = \eta_i,\ \texttt{hops} = 1\rangle, \end{aligned} \tag{3}$$

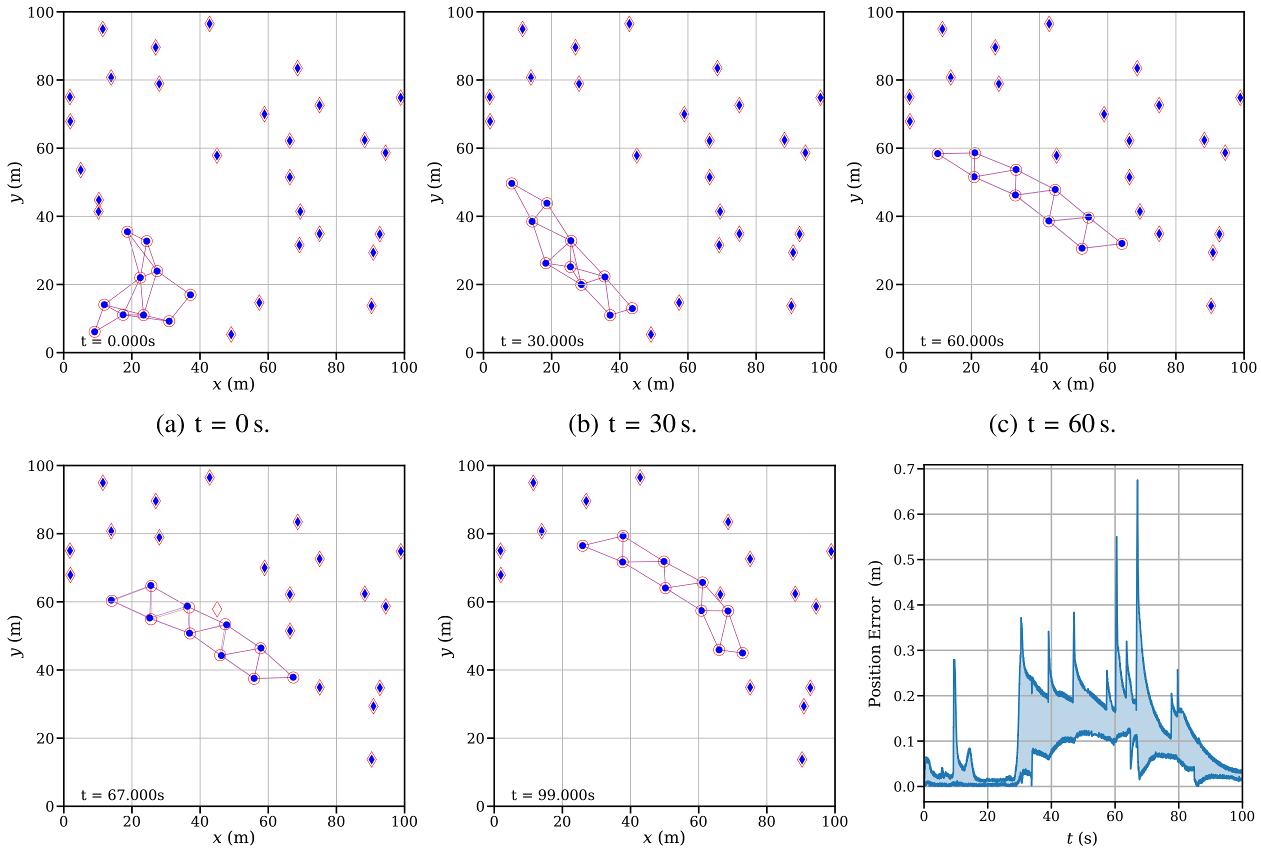
where η_i and σ_i are configuration parameters stored by hunter's *Coordination*, and $\mathbf{u}^j_{i,R}$ is the set of control actions computed for $j \in \mathcal{H}_i$. When hunter i receives a `state` token emitted by j, the *Coordination* model sends $\hat{\mathbf{p}}_i$ to the *Localization* atomic if `hops` $\leq \eta_i$; otherwise, it is discarded. On the other hand, when a j receives an `action` token from i, the *Coordination* model sends $\mathbf{u}^j_{i,R}$ to the *Controller* atomic. This protocol ensures that all robots gather the necessary information to apply the corresponding control action.

4.3 Experimental Results

We consider the following scenario: $N_h = 10$ hunters and $N_t = 30$ targets were randomly distributed in an area of $100\,\mathrm{m} \times 100\,\mathrm{m}$ (Figure 2(a)). All hunters were configured with `comm_range` $= 15\,\mathrm{m}$; targets had `comm_range` $= 50\,\mathrm{m}$, and `capture_range` $= 5\,\mathrm{m}$. The `control_period` for hunters was set to $0.02\,\mathrm{s}$ and for targets to $0.1\,\mathrm{s}$. The *Odometry Interoceptive Sensor* measurements were synchronized with the control (`odometry_period` $= 0.02\,\mathrm{s}$) while the *GPS Exteroceptive Sensor* measurements had `gps_period` $= 1\,\mathrm{s}$. The hunter's *Coordination* parameters were set as $\eta_i = 1$ for 7 hunters, and $\eta_i = 2$ for the rest; additionally, $\sigma_i = 2$ for all.

We took advantage of a preexisting discrete-time simulation (DTS) model, specifically designed for the control strategy described in Section 4.2 and validated in (Presenza et al. 2022a; Presenza et al.

2022b). This model was used for cross-validation against the present EB-DEVS simulation (EBDS). For this comparison, the sensors' noise biases and covariances were set to zero. Finally, the *mQSSI* were configured with `dQMin` $= 0.01$ (fixed), `gain` $= 1$. On the other hand, the DTS considers Euler integrators with time step 1 ms (this value was set according to the value of `dQMin` to obtain comparable errors).



(a) t = 0 s. (b) t = 30 s. (c) t = 60 s.

(d) t = 67 s. (e) t = 99 s. (f) Position Error between hunters.

Figure 2: Simulation snapshots of the multi-robot system using the EBDS (red) and DTS (blue) models for time instants: (a) t = 0 s, (b) t = 30 s, (c) t = 60 s, (d) t = 67 s, (e) t = 90 s. The evolution of the maximum and minimum absolute error between the hunters' positions (EBDS vs. DTS) is shown in (f).

The simulation results are shown in Figure 2. Blue dots and red circles represent the hunters' positions for the DTS and EBDS respectively, while blue filled diamonds and red empty diamonds correspond to the active targets (removed from the picture as they become inactive). Connections between neighboring robots are represented by edges (blue for DTS and red for EBDS). It can be noticed that, as the mission evolves, targets are captured while the rigidity of the hunters' network is maintained. Qualitatively both simulators collect the same targets in the same order, achieving the same control objective. Figure 2(f) shows the evolution of the absolute position error of the hunters resulting from discrepancies between the two simulators. The maximum deviation observed between the trajectories of the robots in both simulations is upper bounded by 0.7 m (or 0.7 %). This deviation can be observed in Figure 2(d) which also shows a target that was collected first in the DTS than in the EBDS (red empty diamond). In terms of performance, the execution times are comparable, although in the EBDS there is significant room for improvement (performance was not a main concern in this work). These results confirm that the proposed approach performs as intended, validating both the robots' dynamic behavior and the token-based communication mechanism essential for the distributed control strategy.

5 CONCLUDING REMARKS AND FUTURE WORK

This paper introduces a library of models designed for MRS and distributed control strategy development, built under the EB-DEVS formalism and simulation tool. It offers a collection of ready-to-use models representing the most common components in robotic and multi-robot systems. These models were designed with modularity in mind, allowing complex operations, such as localization and control algorithms, to be tailored to specific applications. To facilitate this, the corresponding atomic models delegate core computations to external functions (e.g., a Kalman filter or a rigidity controller), making them easy to adapt. To address dynamic communication networks, we introduce the *Transmission Medium* atomic model that captures the transmission medium, responsible for routing messages from each robot to its neighbors based on their physical state. This enables the system to emulate the real-world behavior of robots broadcasting information without explicitly knowing their neighbors.

The proposed library also implements a communication mechanism based on the exchange of tokens, which embeds the number of hops a token must traverse and the hop count into the token's payload, allowing recipient agents to autonomously decide whether to forward or discard them. In constrast, existing simulation frameworks for MRS often require developers to manually implement communication protocols such as multi-hop token passing—a task that demands advanced programming skills. A side contribution of this paper is the multilevel QSS Integrator atomic model, which takes advantage of the micro-macro communication mechanism offered by EB-DEVS.

To demonstrate the library's capabilities, this paper implements a multi-robot experiment comprising N_h hunters tasked with capturing N_t targets, while ensuring network rigidity and inter-agent collision avoidance. The simulation results were validated against a discrete-time simulator tailored to this application originally used to test this control strategy.

This paper demonstrates that the EB-DEVS formalism allows to build a unified simulation tool capable of representing robotic and network dynamics in an event-based manner. While EB-DEVS may appear complex at first, it offers clear advantages over building custom discrete-time models. First, users do not need to implement or manage the simulation algorithm, as it is shared across all models. This allows modelers to focus solely on defining the behavior of submodels, without explicitly handling their interactions—synchronization of discrete events across is entirely handled by the underlying abstract simulator. Second, discrete-event models naturally support random and asynchronous events, which are often difficult—or even impossible—to represent with discrete-time approaches.

As for future work, we plan to continue expanding the proposed library in several ways. Firstly, by incorporating additional atomics models for a broader variety of dynamical models and sensors commonly found in robotic applications. Secondly, by including higher order *QSS* integrators and exploring alternative integration methods. In addition, we plan to incorporate into the *Transmission Medium* atomic model a physical representation of the surrounding environments to enable more realistic modeling of communication channels. Future releases will also extend the *Localization* atomic model description to include mapping capabilities for SLAM implementations, widely used in robotic applications.

REFERENCES

Acharya, S., M. Bharatheesha, Y. Simmhan, and B. Amrutur. 2023. "A Co-Simulation Framework for Communication and Control in Autonomous Multi-Robot Systems". In *Proc. of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 01st-05th, Detroit, MI, USA, 11087-11094. https://doi.org/10.1109/IROS55552.2023.10342407.

Anderson, B. D., C. Yu, B. Fidan, and J. M. Hendrickx. 2008. "Rigid Graph Control Architectures for Autonomous Formations". *IEEE Control Systems Magazine* 28(6):48–63.

Bordón-Ruiz, J. B., E. Besada-Portas, J. L. Risco-Martín, and J. A. López-Orozco. 2021. "DEVS-based Evaluation of UAVs-based Target-search Strategies in Realistically-modeled Missions". In *Proc. of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '21)*. May 31th- June 02nd, New York, NY, USA, 141–152. https://doi.org/10.1145/3437959.3459253.

Calvo-Fullana, M., D. Mox, A. Pyattaev, J. Fink, V. Kumar, and A. Ribeiro. 2021. "ROS-NetSim: A Framework for the Integration of Robotic and Network Simulators". *IEEE Robotics and Automation Letters* 6(2):1120–1127.

Foguelman, D.J. 2022. "EB-DEVS simulation toolkit.". https://git-modsimu.exp.dc.uba.ar/elanzarotti/eb-devs/., accessed 10th March 2025.

Foguelman, D., P. Henning, A. Uhrmacher, and R. Castro. 2021. "EB-DEVS: A Formal Framework for Modeling and Simulation of Emergent Behavior in Dynamic Complex Systems". *Journal of Computational Science* 53:101387.

Hentati, A. I., L. Krichen, M. Fourati, and L. C. Fourati. 2018. "Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis". In *Proc. of the 2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*. June 25th-29th, Limassol, Cyprus, 1495-1500. https://doi.org/10.1109/IWCMC.2018.8450505.

Hu, X., X. Hu, B. P. Zeigler, and S. Mittal. 2005. "Variable Structure in DEVS Component-Based Modeling and Simulation". *SIMULATION* 81(2):91–102.

Kofman, E., and S. Junco. 2001. "Quantized-State Systems: a DEVS Approach for Continuous System Simulation". *Transactions of the Society for Computer Simulation International* 18(3):123–132.

Moallemi, M., and G. Wainer. 2013. "Modeling and Simulation-driven Development of Embedded Real-time Systems". *Simulation Modelling Practice and Theory* 38:115–131.

Pecker-Marcosig, E., S. Zudaire, M. Garrett, S. Uchitel, and R. Castro. 2020. "Unified DEVS-based Platform for Modeling and Simulation of Hybrid Control Systems". In *2020 Winter Simulation Conference (WSC)*, 1051–1062 https://doi.org/10.1109/WSC48552.2020.9384025.

Presenza, J. F., J. I. Alvarez-Hamelin, I. Mas, and J. I. Giribet. 2022a. "Subframework-Based Rigidity Control in Multirobot Networks". In *Proc. of the 2022 American Control Conference (ACC)*. June 08th-10th, Atlanta, GA, USA, 3431-3436. https://doi.org/10.23919/ACC53348.2022.9867693.

Presenza, J. F., J. I. Giribet, J. I. Alvarez-Hamelin, and I. Mas. 2022b. "Control de Rigidez por Subframeworks en Sistemas Multirobot". In *XI Jornadas Argentinas de Robótica*. March 09th-11th, Bariloche, Argentina. (In Spanish.).

SEDLab and LAR 2025. "EB-DEVS Multi-Robot Library". https://git-modsimu.exp.dc.uba.ar/epecker/ebd-multirobot-subframeworks, accessed 11th April 2025.

Testa, A., A. Camisa, and G. Notarstefano. 2021. "ChoiRbot: A ROS 2 Toolbox for Cooperative Robotics". *IEEE Robotics and Automation Letters* 6(2):2714–2720.

Van Tendeloo, Y., and H. Vangheluwe. 2015. "PythonPDEVS: a Distributed Parallel DEVS Simulator". In *Proc. of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, edited by F. Barros, M. H. Wang, H. Prähofer, and X. Hu, 91–98. San Diego, CA, USA: Society for Computer Simulation International.

Zeigler, B. P., A. Muzy, and E. Kofman. 2018. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*. 3rd ed. San Diego, CA, USA: Academic Press.

AUTHOR BIOGRAPHIES

EZEQUIEL PECKER-MARCOSIG has been appointed Assistant Researcher in CONICET (National Research Council of Argentina) and holds a Professor position at UBA. His email address is emarcosig@dc.uba.ar.

J. FRANCISCO PRESENZA is a Post-Doctoral Researcher at INTECIN (Institute of Engineering Sciences and Technology "Hilario Fernández Long") and CONICET (National Research Council of Argentina). His e-mail address is jpresenza@fi.uba.ar.

IGNACIO MAS is Professor at University of Buenos Aires, Argentina and CONICET Scientific Researcher at University of San Andrés. His e-mail address is imas@udesa.edu.ar.

J. IGNACIO ALVAREZ-HAMELIN is Professor at University of Buenos Aires, Argentina and CONICET Scientific Researcher at INTECIN (Institute of Engineering Sciences and Technology " Hilario Fernández Long"). His e-mail address is ihameli@fi.uba.ar.

JUAN I. GIRIBET is Associate Professor at University of San Andrés, Argentina and Scientific researcher at CONICET. His e-mail address is jgiribet@conicet.gov.ar.

RODRIGO CASTRO is Professor with the UBA and Head of the Laboratory of Discrete Event Simulation at the Research Institute of Computer Science (ICC) of CONICET (National Research Council of Argentina). His e-mail address is rcastro@dc.uba.ar.