Combining PDEVS and Modelica for describing Agent-Based Models

Journal Title
XX(X):1–20

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Victorino Sanz[1] and Alfonso Urquia[1]

Abstract

Modelica is a general-purpose modeling language mainly designed to facilitate the development, reusability and exchange of models. It represents the state-of-the-art in equation-based modeling of continuous-time systems. Modelica libraries facilitate the description of multi-formalism and multi-domain models. However, the description of agent-based models (ABMs) in Modelica is not currently supported, mainly due to the characteristics of the language and its simulation algorithm. The combination of ABMs with continuous-time equations provides a powerful tool for describing and analyzing complex systems. An approach for describing ABMs using the Modelica language is presented in this manuscript, with the objective of facilitating the combination of ABMs with the rest of Modelica functionality. Agent behavior is described using a process-oriented modeling approach. Agents are described as individual entities that move across a flowchart diagram, that represents the processes that agents undergo. Processes are formally described using the Parallel DEVS formalism, extended to describe the interface with other Modelica models. The environment where agents interact is described as a cellular automaton. This approach has been implemented in a free Modelica library, named ABMLib. Three case studies are discussed to illustrate the modeling functionality of the library and its combination with other models: a basic traffic model, a sheep-wolves predator-prey model and a consumer market model.

Keywords

Agent-based modeling, Modelica, Parallel DEVS, hybrid systems

1 Introduction

Agent-based modeling (ABM*) is a modeling methodology that has been widely adopted to study systems in multiple domains (e.g., ecology, economics, social science and biology, among many others)[1,2]. Systems in ABM are described as a combination of the individual behaviors of their components, named agents, and their interactions in a common environment. An agent may represent any component of a system, such as birds, cars, customers, molecules, grass, etc. The behavior of each individual agent is usually simple, but their combination sometimes leads to complex or emergent behavior of the system as a whole. The combination of continuous-time and agent-based models (ABMs) leads to a powerful and versatile approach for describing complex systems.

The objective of the work presented in this manuscript is to support the description of ABMs using the Modelica language. The main benefit for describing ABMs in Modelica is to facilitate the combination of ABM with the modeling functionality already supported by Modelica, which represents an advantage specially when the description of the continuous-time part is non-trivial. Modelica is an equation-based object-oriented modeling language[3] widely used in academia and industry. Modelica libraries have been developed to facilitate system modeling in different physical domains (multibody, electrical, thermo-fluid, etc.), and applying different modeling formalism (e.g., physical modeling, bond graphs, System Dynamics, Petri Nets, State Machines, PDEVS, etc.). Modelica models are described in terms of differential and algebraic equations (DAE), and discrete events[4]. Modelica modeling environments (e.g., Dymola and OpenModelica) automatically perform the analyses

[1]Dpto. de Informática y Automática, ETSI Informática, Universidad Nacional de Educación a Distancia (UNED), Spain

Corresponding author:
Victorino Sanz, Dpto. de Informática y Automática, ETSI Informática, Universidad Nacional de Educación a Distancia (UNED), C/ Juan del Rosal, 16, 28040, Madrid, Spain.
Email: vsanz@dia.uned.es

*When ABM is used in plural (ABMs) or with an article (an ABM) it stands for agent-based model instead of agent-based modeling.

and symbolic manipulations on the model equations (assignment of computational causality, reduction of the DAE index, tearing of the algebraic loops, etc.) required to translate the Modelica description of the model into efficient executable code[5]. The use of Modelica frees the modeler from these tedious and error-prone tasks. Also, the use of a single modeling language to describe the entire system, instead of a combination of multiple tools as in co-simulation approaches, facilitates the modeling task, and the exchange and maintenance of models.

The authors have previously proposed an approach for describing ABMs in Modelica[6]. The rationale of the proposal was the description of the behavior of agents in the form of flowchart diagrams. The components and structure of these flowchart diagrams are used to describe the particular behavior of agents. In each model, individual agents are represented by entities that flow from one component to another. Each agent that arrives to a component performs the process represented by that component and, when finished, continues to the next process. This approach complies with Modelica semantic, which imposes that the number and equations and variables can not vary during the simulation. Flowchart diagrams are represented as model variables and equations, while agents are represented by the information transported between modules (i.e., they are values assigned to model variables). The environment is represented as a cellular automaton, whose state is periodically updated using a transition function. Agents can read the state of the environment and are allowed to perform asynchronous changes on it. This proposal was implemented in a Modelica library named ABMLib.

The work presented in this manuscript extends the previous proposal as follows:

- The components of the flowchart diagram are formally specified using the Parallel DEVS (PDEVS) formalism, extended to include the interface with other Modelica models and the environment. PDEVS[7] is an extension of the Classic DEVS formalism[8]. It is focused on the collection of input messages into bags that are available for the external transition function, instead of individual messages, and the parallel execution of transitions, introducing the confluent transition function to manage the simultaneous occurrence of internal and external events. The use of PDEVS as underlying formalism defines clear semantics to describe the flowchart diagrams (i.e., processes and their inter-connections), thus facilitating the understanding of their behavior, the development of additional components, and their adaptation and maintenance in other applications.
- In the initial proposal, the components of the flowchart diagram only described instantaneous processes with only one component, named Tick, used to represent periodical advances in the simulated time. In this manuscript, the proposal is extended into a more general and versatile modeling approach. Modules that compose the flowchart diagram describe either transitory or delayed processes. The former describe instantaneous actions performed by agents, with no time delays (e.g., modify a variable, conditional divisions in the flow, etc.). The latter include time delays for agents (e.g., a queue).
- The proposed approach has been implemented in a new version of ABMLib. The new library constitutes a fully functional implementation that can be executed using Dymola without the requirement of any manual manipulation to the model or the generated code. ABMLib is freely distributed as a part of DESLib[9] (cf. `http://www.euclides.dia.uned.es/`).

The manuscript is structured as follows. Section 2 includes an analysis of related work previously performed by other authors, regarding the application of formal methods or the development of software tools, and their combination with continuous-time dynamics. Also, the support for PDEVS in Modelica using the DEVSLib library is explained. This discussion includes a description of the hybrid DAE model, its simulation algorithm and how it has been used to describe PDEVS operational semantics. The proposed approach for ABM is presented in Section 3. The design and implementation of the new ABMLib library is presented in Section 4. The functionality of ABMLib is illustrated by means of three examples. Each example has been included to illustrate some specific functionality of ABMLib, as discussed below:

1. A basic traffic model is presented in Section 5. This example does not include any continuous-time behavior, but serves to illustrate the description of agent behavior based on PDEVS and the interactions between agents since cars adjust their speeds observing the speed of the preceding car. This example can also serve as a test for the application of the proposed approach for ABM in any other PDEVS simulator.
2. A predator-prey model is discussed in Section 6. In this case, predators are described using a continuous-time model, and prey as an ABM using transitory processes and including independent environment behavior. This example serves to illustrate the versatility of combining PDEVS and Modelica, since the continuous-time behavior could be described using any available Modelica functionality. This model can also be used as a test for other simulators, since Lotka-Volterra equations are simple and can be manually implemented in a PDEVS simulator.

3. A consumer market model is discussed in Section 7. This example introduces the use of delayed processes to describe the behavior of consumers, while the supply chain of products is modeled with continuous-time equations. This model also serves to compare the description of continuous-time dynamics using System Dynamics with the Modelica functionality.

Finally, the conclusions of the presented work are summarized in Section 8.

2 Related Work

The description of continuous-time dynamical systems is commonly formalized in terms of systems of differential and algebraic equations (DAE). However, there is still no agreement for the description of ABMs and multiple approaches have been proposed, mainly focused on the application of formal methods or the development of specific software tools.

The use of formal methods to describe ABMs facilitates the description of systems due to the introduction of formal semantics to describe model behavior, structure and support multi-resolution hierarchical modeling and model reuse [10]. Multiple applications and extensions of formal methods have been proposed.

The application of BDI (Belief, Desire, Intention), Gaia, Tropos, Prometheus, Agent UML and the Z language are reviewed by Wooldridge [11]. The specification of ABMs using process algebras (e.g., CSP or the π-calculus), model-oriented approaches (e.g, the Z or B languages) and logic approaches (e.g., temporal or BDI logic) are also reviewed and discussed in Rouff *et al.* [12]. SLABS is a model-based formal specification language specially designed for ABM [13]. SysML can be used to describe agent-based models either directly creating an executable model [14] or translating it to DEVS [15]. Multiple extensions of Petri Nets have been proposed for describing ABMs, such as G-Nets [16], Object Petri Nets [17], Reflexive Petri Nets [18], and Resource-aware Petri Nets [19]. Other approaches include the application of X-Machines [20], Rewriting logic [21], and Continuous-time Markov-Chains [22].

The DEVS formalism has also been used as a base for the description of ABMs. The use of DEVS to describe ABMs could be a starting point towards a formal description of ABMs, as multiple different modeling formalisms can be described in terms of DEVS [23]. Multiple extensions of DEVS have been developed, such as Dynamic Structure DEVS (DSDEVS) [24,25], Dynamic DEVS (DynDEVS) [26], Mobile DEVS [27], ρ-DEVS [28], M-DEVS [29], Multi-level-DEVS (*ml*-DEVS) [30], Symmetric DEVS [31] and EB-DEVS [32]. All these extensions focus on the description of agents as DEVS models, and extend the DEVS formalism to support the particular characteristics and behavior of ABMs (e.g., dynamic structures, interfaces and couplings). Agent interactions are modeled by means of message communication. Other DEVS-based approaches include its combination with cognitive modeling, the *multi-agents model* [33] or the application of the Cell-DEVS formalism, where each cell represents an individual agent [34]. Examples of applications can also be found in [35–37].

On the other hand, multiple software tools have been developed to describe ABMs (cf. [38] and [39] for an extensive survey on ABM platforms and tools). The description of the model greatly depends on the functionality and design of the tools used for its implementation, ranging between spreadsheets (e.g., MS Excel), general-purpose programming languages (e.g., Java), mathematical software (e.g., Matlab) or ABM specific tools (e.g., NetLogo, AnyLogic, Repast, etc.) [40].

2.1 Combination of ABMs with Continuous-time Equations

Different approaches are used to describe the continuous-time part of models that combine equations with ABMs.

One approach is to use System Dynamics (SD) [41] to graphically describe ordinary differential equations in the form of rates, stocks, and their relationships. SD is supported by NetLogo [42] and AnyLogic [43], that facilitates the combination with ABM. For example, this approach has been applied to the analysis of the urban commuting CO_2 emissions in Beijing [44], the management of the water quality in lakes [45], the study of animal foraging [46] or the spread of pollution [47].

A second approach focuses on the development of custom models to combine the ABM part with the differential equations of the continuous-time part, since ABM tools do not facilitate the description of equation-based models. This combination is based on the discretization of the continuous-time equations, either using a time-based (e.g., Forward Euler [5]) or a state-based (e.g., QSS methods [48]) approach, in order to simulate it with a discrete-event tool. Although this approach allows to simulate hybrid systems in combination with formal methods or other discrete-event tools, the task of manually translating the model into a discrete form is left to the modeler increasing the difficulty of the development.

Model transformations have been also proposed as an alternative approach for multi-formalism modeling. DEVS has been proposed as a common denominator to unify the description continuous-time and discrete-event dynamics, using state-based discretization to transform the continuous-time equations into DEVS models [23]. The approach proposed in this manuscript, together with other available Modelica libraries that support other formalisms (e.g., System Dynamics, Petri Nets, Bond Graphs, etc.), can be seen as model transformations where the destination

formalism corresponds to the hybrid DAE supported by Modelica.

Custom models like the Global Epidemic Model (GEM)[49], MISS[50] or AADIS[51] have been developed to combine ABM with SEIR disease spread models, implementing them using a general-purpose programming language (e.g., Java). Other authors use Matlab to describe both the equations used to model intra-host bacteria reproduction, and the inter-host relationships described as an ABM[52]. NetLogo has also been used to combine ABM with continuous-time equations, but in this way the equations need to be manually transformed into difference equations in order to be simulated[53,54]. HyVisual is a domain-specific programming language specially designed for the description of hybrid systems supported in Ptolemy II[55]. In this case, continuous-time dynamics are represented using a block-diagram approach which is combined with finite-state machines used to describe different modes of operation and discrete-event behavior. The combination of ordinary differential equations (ODE) and ABM is also discussed in[56], where at each integration step the ABM is used to compute new parameters of the ODE system, and the values of the state variables of the ODE system are used as new parameters for the ABM. A hybrid ABM approach is discussed in[57], where time-based and state-based discretizations for the description cyber-physical systems are combined.

A third approach is based in the application of co-simulation[58–61], where multiple tools with different functionalities are combined to represent the final system[62–65].

These approaches present the following respective limitations: offering limited support for describing continuous-time equations by only using explicit ODE; requiring manual manipulations of the model equations in order to discretize them to apply any numerical integration approach for the simulation; and requiring to use a combination of different tools to describe the whole system, that usually involves learning how to use different tools, different modeling approaches, additional programming skills, etc. The Modelica language represents the state-of-the art in equation-based modeling of continuous-time systems. The use of Modelica, as an unified language for describing models that combine ABM and equations, would not exhibit the limitations previously described. The combination of Modelica functionality and PDEVS, as underlying formalism to describe the actions performed by agents, provides a general-purpose tool for the description of ABMs. Additionally, the ABM approach proposed in this manuscript could be applied to simulate ABMs in any other PDEVS simulator without requiring additional extensions. This is the aim of our work. To our best knowledge, the library presented in this manuscript (i.e., the ABMLib Modelica library) is the first full-fledged library for describing ABMs in Modelica.

2.2 An Implementation of PDEVS in Modelica

The authors have previously developed the DEVSLib library as an implementation of PDEVS in Modelica[66]. The description of atomic models is analogous to their formal description, following the elements of the tuple (e.g, interface ports, state, transition functions, output function and time advance function). The description of coupled models also follows their formal description, as a set of internal components, an interface and couplings between all these elements. In this section, the implementation of the DEVSLib library is presented. The limitations imposed by the design of Modelica and its simulation algorithm to support PDEVS and its operational semantics are discussed, together with their solutions implemented in DEVSLib. DEVSLib is used for the development of ABMLib.

Modelica models can be described either behaviorally, in terms of differential and algebraic equations, and discrete events, or structurally, in terms of a set of interconnected components (cf.[4] for a detailed description of the language and its functionality). In the latter case, component connections also follow an equation-based rationale, introducing additional equations to the model.

Modelica models are automatically transformed by the M&S tool into a mathematical description following the hybrid DAE formalism[67]. Hybrid DAE models include parameters or constant variables ($\mathbf{p}$), the time as independent variable (t), variables that appear differentiated in the equations ($\mathbf{x}(t)$), discrete variables whose value only change at event instants t_e ($pre(\mathbf{m}(te))$ represents the value of the $\mathbf{m}$ variable previous to the event execution, and $\mathbf{m}(te)$ represents the variable value after executing the event), algebraic variables ($\mathbf{y}(t)$), event conditions $c(t_e)$ and invariants ($relation(v)$ where v are the model variables). Modelica modeling environments automatically determine the order in which the model equations need to be evaluated, manipulating symbolically and sorting the model equations to generate the executable code of the simulation algorithm.

The simulation of the hybrid DAE model is graphically described in Fig. 1. Briefly, before starting the simulation consistent initial values are calculated for the variables of the model. After that, continuous-time integration is performed until any event is triggered. Conditions (c) and discrete variables ($\mathbf{m}$) are kept constant during numerical integration. The occurrence of events is detected using the invariants $relation(v)$ (i.e., zero-crossing functions). Modelica supports the definition of time and state events. When an event is triggered, the numerical integration is halted, the event instant is determined for state events and the actions associated with the event are executed (i.e.,

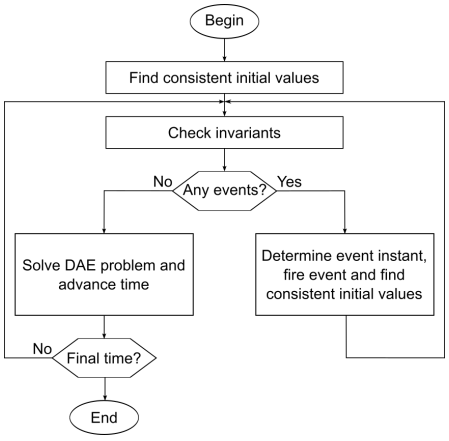

Figure 1. A simulation algorithm for hybrid DAE models.

solving a set of algebraic equations). After managing the event, an event iteration procedure is performed in order to manage newly triggered events (i.e., cascades of events), or otherwise, calculate new consistent initial values and resume the numerical integration.

PDEVS models are formally described as *atomic*, with the tuple $(X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$, or *coupled*, with the tuple $(X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC)$. $X = \{(p, v) | p \in InPort, v \in X_p\}$ and $Y = \{(p, v) | p \in OutPort, v \in Y_p\}$, where $InPort$ and X_p are the sets of input ports and values, and $OutPort$ and Y_p are the sets of output ports and values, respectively.

The behavior of PDEVS atomic models is described by the occurrence of internal events, scheduled in time using the time advance function ($ta : S \rightarrow \mathbb{R}^+_{0,\infty}$), or external events, triggered when an input is received. When an internal event is triggered and no inputs are received, the state S is updated using the internal transition function $\delta_{int} : S \rightarrow S$. When an input is received before reaching the time for the next scheduled internal event, the state is updated using external transition function $\delta_{ext} : Q \times X^b \rightarrow S$, with $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$, being e the time elapsed since the last transition and X^b a bag over the elements in X. When an internal and an external event are triggered simultaneously, the state is updated using the confluent transition function $\delta_{con} : Q \times X^b \rightarrow S$. The output function, $\lambda : S \rightarrow Y^b$ (where Y^b is a bag over the elements in Y), is evaluated before the internal or confluent transition functions, and may generate output messages that are synchronously transmitted to their destinations, triggering external events. After evaluating any transition function, a new internal event is scheduled using the time advance function. In the case of the external transition, this new internal event replaces the one currently scheduled.

The specification of coupled models includes a set of component names, D, and a set of PDEVS models M_d, for each $d \in D$. EIC, EOC and IC describe the couplings between the interface of the coupled model and their internal components, and among components themselves.

In order to support PDEVS in Modelica, the following characteristics were considered for the development of DEVSLib[66]. Communication between models follows a message-passing approach, which is conceptually different from the equation-based component connection supported by Modelica. Message transmissions are triggered by the occurrence of internal events, scheduled in time. A message-passing communication (MPC) approach was developed in DEVSLib to facilitate PDEVS model communication[68]. This MPC functionality has been also implemented in the MSGLib library, to be used in any Modelica model additionally to DEVSLib models[69]. This MPC mechanism is based on the following rules:

- Messages are transmitted only at time events, in order to fulfill the semantics of PDEVS models. Message transmissions are synchronous and are immediately considered as external inputs in their destinations. This MPC mechanism also guarantees that outputs are immediately routed to their destinations, following the defined couplings between models.
- The content of the transmitted messages can only depend on constants, parameters and *pre()* values of discrete-time variables. This guarantees that the content of the messages is known, as it only depends on constants, parameters or the state of the model previous to the event, which fulfills the requirement for the evaluation of the output function in PDEVS models.

The application of these rules guarantees that when the Modelica modeling environment sorts the equations of the complete model, the equations required to evaluate the output functions will be sorted first (since they only depend in known values of the model state), followed by the equations required to transmit output messages to their destinations, and finally, the rest of the equations required to modify the state of the model. The event iteration mechanism is used to manage cascades of events triggered at the same instant, until no additional events are triggered and the next scheduled event requires an advance in time. In this way, the generation of output messages, their transmission and the modification of the model state satisfy the PDEVS specification. Simultaneous occurrences of internal and external events were also taken into account. Additionally, the support for bags of messages was implemented in the message-passing mechanism, since Modelica does not provide support for dynamic data structures. The simulation of DEVSLib models was validated and compared with equivalent models described using PowerDEVS[70] and Arena[71].

3 A Proposal for ABM in Modelica

The approach proposed in this manuscript allows to describe ABMs using PDEVS, since no structural changes in the model at simulation runtime are required. In addition, this proposal allows to describe a variable number of agents during the simulation run in Modelica. This is achieved without interfering with the model analyses and transformations performed by the Modelica modeling environment (e.g., Dymola and OpenModelica) in order to generate the executable code, which require to describe the model using a number of equations and variables that can not change during the simulation run. Our proposal is also inspired in NetLogo, where the behavior of agents is described as a sequence of commands. The components of the flowchart diagrams can be seen as sequences of commands or procedures, and the structure of the diagram defines the flow of execution.

The ABMs considered in this manuscript are composed of agent behavior and continuous-time behavior. These components are graphically shown in Fig. 2.

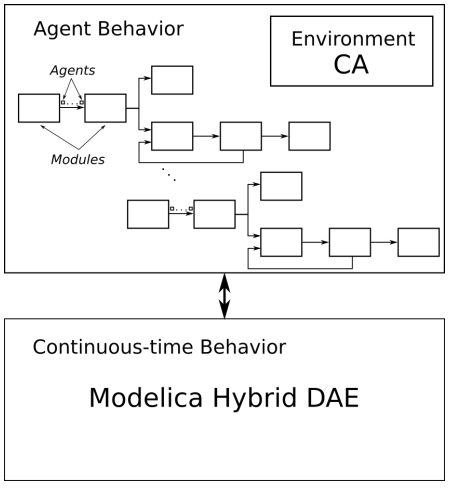

Figure 2. Proposed ABM approach.

3.1 Agent Behavior

The behavior of agents is described as flowchart diagrams, at least one, composed of interconnected components, named *modules*. These diagrams are used to describe one or multiple types of agents, and their interactions. Modules are formally described as PDEVS models, extended to include input and output continuous-time ports as in DEV&DESS models[72]. The continuous-time inputs influence the transition functions. The continuous-time outputs are generated depending on the state of the model.

Messages sent between modules represent the individual agents of the system. The information that composes each message represents the state of each agent, usually in the form of state variables or *attributes*. Different types of agents can be described using different types of messages.

Each module represents a process performed by the agents. Modules are either transitory (e.g., used to represent instantaneous actions performed agents) or delayed, in which agents will remain until some conditions are satisfied (e.g., a process where agents must wait for some time or where only a percentage of agents is able to continue while the rest remains waiting).

Transitory processes are usually described using the external transition function of an atomic model. This represents that agents arrived to the module perform an action, are stored in an internal queue, and trigger an immediate internal event (i.e., the time advance function after the external transition returns zero). During the management of the internal event, the output and internal transition functions are used to send agents to the next process, empty the internal queue of agents and return the module to idle state. Also, more complex processes can be described as a combination of simpler modules in a coupled model.

On the other hand, modules that describe delayed processes can also be described as atomic models. These must include the evaluation of the conditions for agents to leave the module, usually in the internal transition function, and the management of the time the agents spend in this process, using the time advance function. Both types of processes, transitory and delayed, can be combined in the same flowchart diagram.

The environment where the agents live is defined as a bi-dimensional cellular automaton (CA). The transition function of the CA is used to represent the behavior of the environment, synchronously updating the state variables of each cell. This transition function is periodically evaluated, by triggering internal events, and independently of the behavior of agents described in the flowchart diagram. The state variables of the environment cells are global variables of the model, so all modules have access to them. This way, the behavior of agents may be influenced by the state of the environment and also the environment can be asynchronously modified by the agents. The state and behavior of the environment can be seen as belonging to all the modules included in the flowchart diagram, however its representation as a single cellular automaton facilitates its management (e.g., any change in the environment will be immediately shared to all modules present in the system, thus allowing the agents to observe and modify it). This approach is similar to the state of coupled models in EB-DEVS[32], or macro-devs models in multi-level DEVS[30].

The interaction between the environment and the modules included in the flowchart diagram complies with the state encapsulation imposed by the PDEVS

formalism (i.e., only the transition functions of the PDEVS atomic component can modify the state of the component). The state of the environment is an argument of the transition functions of the PDEVS components that describe the modules. The transition functions of the PDEVS components can modify the state of the environment. An example of this behavior is shown in the Sheep-Wolves model, where sheep eat the grass of the environment decreasing its content in the current location of the sheep. Simultaneous modifications to the environment may be performed and their order depends on the ordering of the model equations performed by the Modelica tool when translating the model into executable code. This ordering may affect to the order in which modules are evaluated, and the ordering of the agents received in the ports of the modules. Modelers do not have control over this ordering. Specific orderings in the evaluation of agents may be described within a module, using a custom ordered queue of agents. This situation is similar to the random order obtained when reading agents from agentsets in NetLogo.

Interactions between agents are described through the environment. Some of the state variables of the environment can be used to store part of the state of individual agents. Then, these state variables can be used to influence the behavior of other agents, since the environment is accessible from all the modules. This is illustrated in the Basic Traffic model included in Section 5, where a car can adjust its speed depending on the speed of the preceding car in order to avoid collisions. This approach provides simple, flexible and versatile way for describing agent interactions. However, possible inconsistencies between agent's states and the states stored in the environment have to be properly managed by model developers.

3.2 Interface with Continuous-time Behavior

The continuous-time behavior of the model can be defined using any functionality provided by Modelica. This continuous-time part, together with the agent behavior, will be translated into a hybrid DAE model, as specified in Modelica[67], and simulated using the algorithm described in Section 2.2.

The interface between the continuous-time part and the ABM part can be defined using input and output connectors in the modules of the flowchart diagram. These interface connectors can be connected with other Modelica models using the available connect sentences, which, as mentioned above, describe equation-based communication between model components. These inputs and outputs, and their connections with other models, will be considered as variables and equations of the hybrid DAE model.

The input connectors included in the modules can be used as parameters for the transition functions, additionally to the state, the input bag and the elapsed time. When an event is triggered in a module, the current value of these continuous-time input connectors could be used to describe the behavior of the module when executing the corresponding transition function. The value of an output connector included in a module can be defined using equations that depend on the state of the module. Since the state of modules, as PDEVS models, only changes at event instants, this approach generates outputs as piecewise-constant signals. This approach is demonstrated in the market hybrid model described in Section 7, where the stock of a product influences the purchases from the customers and the number of purchases influences the manufacturing rate of additional products.

Another approach to describe the interface between continuous-time and ABM is to translate the flow of agents into a continuous-time signal, that could be connected with other models, and viceversa. For example, the flow of agents in a traffic model could be aggregated in a continuous-time variable that represents the mean of cars per unit of time. Similarly, aggregated variables could be transformed back into a flow of agents. This approach is demonstrated in the Sheep-Wolves model described in Section 6, where the number of sheep agents present in the model is used in the Lotka-Volterra equation that describes the behavior of wolves.

4 The ABMLib Library

In this section, the support and implementation for the elements of ABMs following the proposed approach are presented.

4.1 Agents

Agents in ABMLib are described as an array of `Real` variables, named *attributes*. Agents must have a minimum of three attributes (i.e., `Real[3]` that correspond to $\{port, type, value\}$), because these are required by DEVSLib to manage them as DEVS messages. Any other attribute can be used to describe additional characteristics of the agents. For example, agents able to move around the environment require attributes to describe their position in the space ($xpos$ and $ypos$) and their orientation for moving forward ($head$).

4.2 Modules

ABMLib includes several pre-defined modules used to describe some common processes performed by agents. These modules include the interface ports required to receive and send agents. The behavior of these modules represents simple processes and is implemented as atomic PDEVS models. The specification of these models is similar to the simple examples discussed in[8]. The `Create`

module is analogous to the generator (cf.[8] p. 99), the `Dispose` module is analogous to the passive (cf.[8] p. 96), the `Assign`, `Modify`, and `Count` modules are analogous to the processor (cf.[8] p. 101) with zero processing time (since these models represent transitory processes that are performed without an advance in the simulation time), the `Tick` module is analogous to the processor with positive processing time (since it represents a delayed process, so agents are stored until the *tick* time is elapsed), the `Duplicate` module is analogous to a processor where the outputs are sent simultaneously through two output ports (e.g., the original agent and its duplicate), and the `Decide` module is also analogous to a processor with two output ports, and outputs are sent through each port depending on the value of the condition. Also, many of these modules include input and output connectors to interface with continuous-time models. The included modules an their descriptions are:

`Create` This module is used as the starting point for agents. It generates agents in periodical batches, that can be configured using the module parameters. This module assigns to every agent a unique identification number, stored in its $value$ attribute. Random inter-arrival times can be easily described by modifying this module.

`Dispose` This module is used to remove agents from the simulation and the graphical animation.

`Assign`, `Modify` These modules are included to modify the values of the attributes of agents. The `Assign` module changes the current value of an attribute, while the `Modify` module sums a quantity to the current value. The new value, or the modifier, can be set as a constant parameter (`value`), a continuous-time Real input (`CVAL` port) or a random variate, using the `randomD` and `randomP` parameters to configure the desired distribution and its parameters.

`DecideC`, `DecideA`, `DecideP` These modules can be used to divide the flow of agents in the diagram. `DecideC` divides the flow depending on an conditional expression given as input, either as constant parameter (`inputCondition`) or continuous-time Boolean value (`COND` port). `DecideA` divides the flow depending on the value of an attribute compared with an input value. The module can be configured to evaluate if the attribute is equal, greater or smaller than another value, or a combination of these conditions. The input value can be a constant parameter (`value`) or a continuous-time Real input (`CVAL` port). `DecideP` divides the flow depending on a probability, which can be set as a constant parameter (`inputProbability`) or a continuous-time Boolean input (`PROB` port).

`Duplicate` This module generates clones of agents. Each agent received is cloned, assigning to the clone a new unique identification number.

`Count` This module can be used to account quantities based on the flow of agents in the diagram. A constant value (`value`) or a continuous-time Real input (`CVAL` port) is summed to the value of the counter every time a new agent is received. The value of the counter can be observed using the `COUT` port.

`Tick` This module is used to represent periodical simulation steps, by defining the time duration of a *tick*. All agents received in this module wait until the *tick* time is elapsed. Similarly to the `tick` command in NetLogo, usually located at the end of the `go` procedure, this module is usually located at the end of a loop in the flowchart diagram that represents the end of the set of processes performed by agents every simulation step. The module is also used to count the number of agents waiting in each step, that can be observed using the `NAGENTS` port.

These modules can be easily extended or modified to adjust their behavior to other requirements. Additional or more complex processes can be implemented as new atomic PDEVS models, or as a combination of several modules in a coupled PDEVS model.

ABMLib only includes one delayed process, the `Tick` module. Additional delayed processes can be described as a combination of the `Tick` and other modules (e.g., any `Decide` module used to describe the condition to leave the process), or as new atomic PDEVS models, where the delay is managed using the time advance function. In the latter case, the conditions to leave the module need to be expressed in the transition functions of the model.

4.3 Environment

In ABMLib the environment is described as a discrete bi-dimensional space of cells. Each cell can store multiple state *variables* that are used to describe characteristics of the space (e.g., amount of grass, wind velocity, etc.) or attributes of agents (e.g., agent ID, energy, speed, etc.) located in that cell. These variables are used to describe the interactions of agents with the environment or with other agents. Agents in a module can access the variables of the environment as a part of their behavior, and can update or modify their values. For example, consider that one variable of the environment is used to store the amount of agents present in each cell. An agent that wants to move its position in the space needs to decrease the value of that variable in its

current position and increase the value in its next position. The access between the environment and the modules is described using the Modelica `inner/outer` variable modifiers, and performed using the functions described below.

The components of ABMLib used to describe the environment are:

`EnvObj` This is the external object used to manage the data required to describe the environment. External objects in Modelica are used to manage special data structures using C code instead of Modelica. Multiple functions are included in ABMLib to manage the information stored in the `EnvObj` object:

- `ENV_plot`, used to plot one of the variables of the environment using GnuPlot to generate the graphical animation of the simulation.
- `ENV_addplot`, generates a new plot that allows to simultaneously generate graphical animations for multiple variables of the environment.
- `ENV_clear`, resets to zero the value of one variable in all the cells of the environment.
- `ENV_modify`, can be used to modify the value of a variable in one cell. The new value is summed to the current value of the variable.
- `ENV_set`, can be used to set the value of a variable in one cell.
- `ENV_read`, is used to observe the value of a variable in one cell.
- `ENV_newID`, when a new agent is created this function can be used to assign a new unique ID. This is specially relevant when multiple agent creation or reproduction modules are present in the model, in order to guarantee unique IDs for all the agents.

A three-dimensional matrix is used to describe the space and the variables of each cell. Two dimensions are used as coordinates for the cells and the third is used to store the variables. `EnvObj` can be configured to automatically generate the graphical animation with the values from one of its variables.

`EnvPort` Is the Modelica connector used to make the environment available to other models. The connector contains a variable of `EnvObj` type that represents the environment.

`Environment` It is the model used to represent the environment. It includes an `EnvObj` object and an `EnvPort`. This model also includes the periodical updates of the graphical animation, if generated.

`NewPlot` This model can be used to generate additional animations for other variables of the environment. It has to be connected to one `Environment` model using their `EnvPort` connectors.

The behavior of the environment can be described using one or more modules, like the ones used to describe agent behavior in the flowchart diagrams. However, these environment modules have to be disconnected from the rest of modules of the flowchart diagrams in order to describe the transition function of the environment independently from the behavior of agents. Since all modules have access to the environment, these environment modules can be used to update the state of the environment as required. The `grassgrowth` module included in the sheep agent of the Sheep-Wolves model is an example of this independent environment behavior.

4.4 Comparison with other ABM tools

In this section the modeling functionality of ABMLib is compared with other tools, such as Swarm, Java Swarm, Repast, MASON and NetLogo, following the analysis performed by Railsback *et al.*[73]. Different versions of a very simple model (named StupidModel) were implemented to compare the functionality and versatility of these ABM tools. Each version of StupidModel is focused on the analysis of a particular modeling functionality. The support of these functionalities in ABMLib is discussed next:

Mobile agents: agents in StupidModel live in a toroidal bi-dimensional grid of cells. The `Environment` module included in ABMLib can be used to describe a non-toroidal bi-dimensional grid space. Additional code for describing a toroidal space can be included in the modules that describe interactions between agents and the environment (e.g., when the position of an agent is modified, etc.). As mentioned, the environment model automatically generates a graphical animation displaying one of the variables stored in the environment. Additional variables can be displayed in other animation windows using the `NewPlot` model, also included in ABMLib. The animation is autonomous, which means that it is periodically refreshed without any dependence on the processes performed by agents. Animations are generated using GnuPlot, and their customization is still very limited (e.g., colors, shapes, forms, etc.). For example, in the Sheep-Wolves model two animations are generated to show the amount of grass and the position of sheep. The position of each sheep agent is controlled assigning values to its `xpos` and `ypos` attributes and using the `forward1Step` model to move them around the environment. This

model also updates the variables of the environment used to display the position of each sheep in the animation.

Agent growth: the state of each agent is stored in their attributes. Changes, such as growth, must be performed to any of these attributes. If these changes need to be displayed in the graphical animation, a specific model needs to be included in the diagram to perform this interaction between the attributes of the agent and the variables of the environment. For example, the Sheep-Wolves model displays the position of each sheep, however the animation could display their energy by storing it in one variable of the environment and use it to generate a new animation.

Habitat cells: these can be described using the variables of the environment model. In the Sheep-Wolves model one variable is used to describe the amount of grass in each cell, which is periodically updated by the `grassgrowth` model to represent grass growing and is reduced when any sheep in that cell eats the grass. The same variable is used to update the animation showing the amount of grass.

Probes: in ABMLib, agents can only be observed at processes or using variables of the environment. Custom modules can be implemented to observe agents as additional processes in the flowchart diagram. Also, all interactions between agents are performed using the variables of the environment. These interactions are not present in the Sheep-Wolves model, but are performed between cars in the Basic Traffic model to adjust the speed of each car with relation to the car ahead in order to avoid collisions.

Parameters: in Modelica, parameters are a special class of variables and are described using the `parameter` modifier when declaring a variable. Parameter values are defined in the code of the models and must remain constant during the simulation. Multiple simulation runs with different parameter values can be automatically programmed using the Modelica script language. Also, Dymola supports parameter sweep that allows to automatically explore a range of parameters for the simulations.

Histogram output: ABMLib does not support the creation of histogram information. Data required for the generation of the histograms needs to be generated adding the required code or modules to the models or flowchart diagrams.

Stop: Modelica includes a function named `terminate()` that allows to stop the simulation before reaching the programmed simulation time. This function can be combined with a `when` statement to define the condition required for the simulation to stop.

File output: ABMLib does not generate any output files. The required outputs can be generated using custom code and using the functions provided by the Modelica Standard Library to open, write and manage files. Similarly, these functions were used in SIMANLib and ARENALib[71] to generate statistical outputs.

Random actions: processes in ABMLib are ordered following the structure and couplings between modules in the flowchart diagram. A random execution of these processes is not supported. It could be described by randomly routing the agents to different sets of processes, each set with the same processes coupled in different order, using a probabilistic division in the flow of agents in the diagram (e.g., the `decideP` module). Also, random transitions between processes, or sets of processes, can be implemented. The RandomLib library is distributed together with ABMLib, as a part of DESLib, to facilitate the generation of random numbers and variates[71].

Size-ordering movement: each module of the flowchart diagram receives in its input ports a list of agents (i.e., the bag of input messages for PDEVS models). The ordering of the list of agents is not controlled by ABMLib, since it depends on the order of the send functions imposed by the analysis and symbolic manipulation of the whole Modelica model before generating the executable code. However, an internal queue of agents could be included in any module to implement the desired order (e.g., based on the value of an attribute).

Optimal movement: ABMLib does not provide functionality to manage neighbors or adjacent cells. All the interactions with other agents and the space are performed using the variables stored in the environment. Custom code and modules can be implemented to find optimal solutions for movement or any other kind of interaction (e.g., the closest cell with the biggest amount of grass).

Mortality and reproduction: agent reproduction is supported in ABMLib by the `Duplicate` module, that generates clones of agents. New agents are assigned with a unique ID, that can be used to distinguish them from the original agent. Agent mortality can also be described using the `Dispose`

module, that removes from the simulation all the received agents. These two actions are shown in the Sheep-Wolves model.

Population graph: the `Tick` module automatically counts the number of agents in the simulation and assigns it to its `NAGENTS` port. This port can be used to generate the population graph.

Random normal initial size: the `Assign` and `Modify` modules support the generation of random values from multiple continuous and discrete distributions. These modules can be used after the creation of agents to set the desired initial random values to agent attributes.

Habitat data from file: the use of input data from external files is not supported in ABMLib, but it is supported by Modelica and could be easily included in an ABMLib model.

Predators: additional types of agents can be included in the models using additional flowchart diagrams to describe their behavior. For example, Wolf agents could be described in the Sheep-Wolves model using an additional flowchart diagram analogous to the Sheep (e.g., including modules for wiggle, move, hunt, reproduce, die, etc.).

5 Basic Traffic Model

This model replicates the basic car traffic model described by Wilensky and Rand[42]. It is presented here to illustrate the ABM functionality of ABMLib with a very simple model that includes interactions between agents.

The model is composed of a fixed number of cars that move along a single-lane circular road. The initial number of cars and the length of the road are parameters of the model. The state of each car is defined with two attributes: its position on the road ($pos \in \mathbb{R}$); and its current speed ($speed \in \mathbb{R}$). The initial position and speed of each car is randomly assigned. The minimum and maximum values for the speed ($minSpeed, maxSpeed \in \mathbb{R}$) are also parameters of the model. The road is divided into 200 discrete segments, and each of them can be occupied by a single car. The segment occupied by each car is defined as the integral part of its position ($\lfloor pos \rfloor$). After the initial assignment of positions, cars are separated to avoid several cars in the same road segment.

The behavior of each car, after the initialization is as follows:

- Each car observes the segment of the road ahead of its current position to see if it is occupied by another car. If not, the car increases its speed, and otherwise, it slows down to avoid collision.
- The speed of the car is adjusted between the defined minimum and maximum values.
- The car advances its position, depending on its current speed.
- This behavior is iterated every time step until the final simulation time is reached.

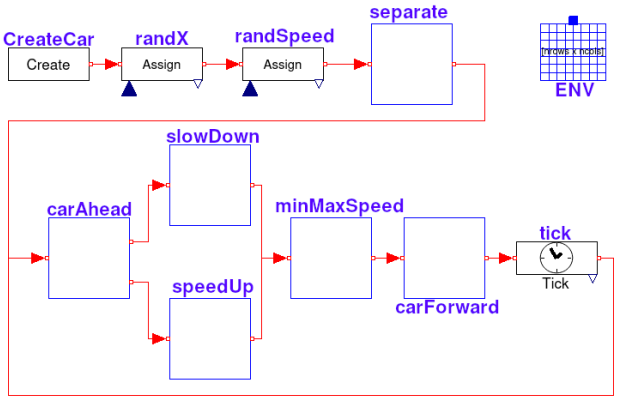

Figure 3. Flowchart diagram of the Basic Traffic model.

The flowchart diagram that represents the behavior of the cars and the environment, that represents the road, is shown in Figure 3. The diagram is composed of a combination of pre-defined modules from ABMLib (i.e., the Create, Assign, Tick and Environment modules), and custom-made modules implemented to describe other processes performed by the cars (i.e., separate, carAhead, slowDown, speedUP, minMaxSpeed and carForward). The behavior of these components is not detailed since the actions performed are very simple and their names are self-explanatory. An example of graphical animation is shown in Fig. 4, where each colored line represents a car and its current position. Note that cars move from left to right in the animation.

The performance and scalability of the simulations has been evaluated. The model has been simulated, during 1000s and with a road length of 2000 segments, with an increasing initial number of cars, between 100 and 2000. The simulations have been executed on an Intel Core i7-4720HQ CPU running Linux 5.15.1 and Dymola 2021. The execution times, obtained without the generation of the graphical animation to focus on the simulation of the agents, are shown in Table 1. Since the execution time depends on the number of agents present in the model and the size of the model, the execution time per module is also included. Per module execution time corresponds to the total execution time divided by the number of agents times the number of modules present in each simulation cycle (i.e., the size of the loop between ticks in the flowchart diagram). The evolution of both indicators is graphically

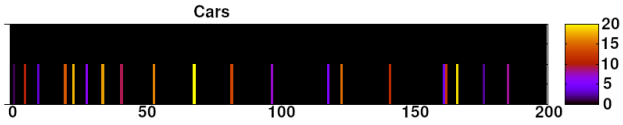

Figure 4. Graphical animation of the Basic Traffic model.

presented in Fig. 5. The execution time per module grows almost linearly with the number of agents in the model.

Table 1. Performance of the Basic Traffic model.

Cars	Exec. Time	Exec. Time per Module
100	$0.9087s$	$0.0018s$
500	$4.27s$	$0.0017s$
1000	$11.2s$	$0.0022s$
1500	$22.7s$	$0.0030s$
2000	$38.6s$	$0.038s$

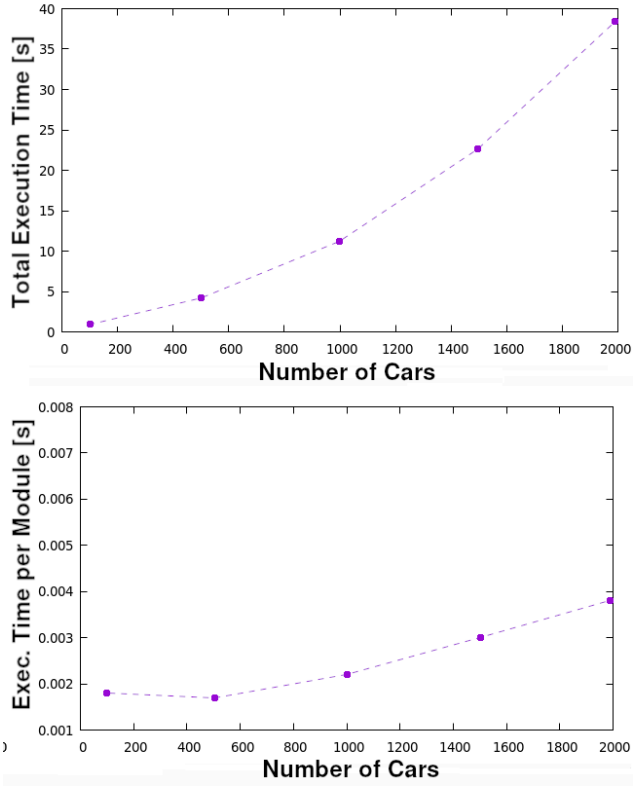

Figure 5. Evolution of total and per module execution times for the Basic Traffic model.

6 Sheep-Wolves Hybrid Model

A modified Sheep-Wolves model is presented as a demonstration of the modeling functionality of ABMLib in combination with other Modelica models. The model is based on the sheep-wolves model from NetLogo[42]. In NetLogo, sheep and wolves are modeled as agents. In our version, only sheep are described as agents, while wolves are described using one of the Lotka-Volterra equations for predator-prey systems. This model is used to illustrate the combination of ABM with continuous-time dynamics when both parts are clearly separated, sheep on one side and wolves on the other. Also, the description of agents mainly involves transitory processes and the simulation steps are controlled using the tick module.

Briefly, the model consists on a bi-dimensional discrete space, divided into $N \times N$ cells, and populated by sheep and wolves. Each cell of the space is covered with an initial amount of grass, that serves as food for sheep. Grass in each cell grows every time step until a maximum amount of grass is reached. Each individual sheep living in the space is represented by an agent, whose behavior is detailed below. Wolves are represented as a whole quantity across the space.

6.1 Sheep

The state of each sheep is described using the tuple of attributes: $\langle ID \in \mathbb{N}, head \in [0,7], xpos \in [1,N], ypos \in [1,N], energy \in \mathbb{R} \rangle$, where:

- ID is the unique identifier of the sheep.
- $head$ is the orientation of the sheep (e.g., 0 = E, 1 = S-E, 2 = S, 3 = S-W, 4 = W, 5 = N-W, 6 = N, 7 = N-E).
- $xpos$ is the position of the sheep in the X axis.
- $ypos$ is the position of the sheep in the Y axis.
- $energy$ is the current energy of the sheep.

The behavior of sheep is graphically described with the flowchart diagram shown in Fig. 6. Note that the Wiggle, Move, SheepEat and Reproduce modules are coupled PDEVS models, and their internal components are also shown in the figure to better illustrate the structure of the diagram. A fixed number of sheep are created at the beginning of the simulation, and random values are assigned to $xpos$, $ypos$ and $head$. After this initialization of sheep agents, their behavior in each simulation step is as follows:

1. The sheep wiggles, which means that turns its head a random angle between $[-90°, 90°]$ and is modeled as a random turn to the right followed by a random turn to the left.
2. After the wiggle, the sheep moves one cell forward, in the direction of its current head. During the

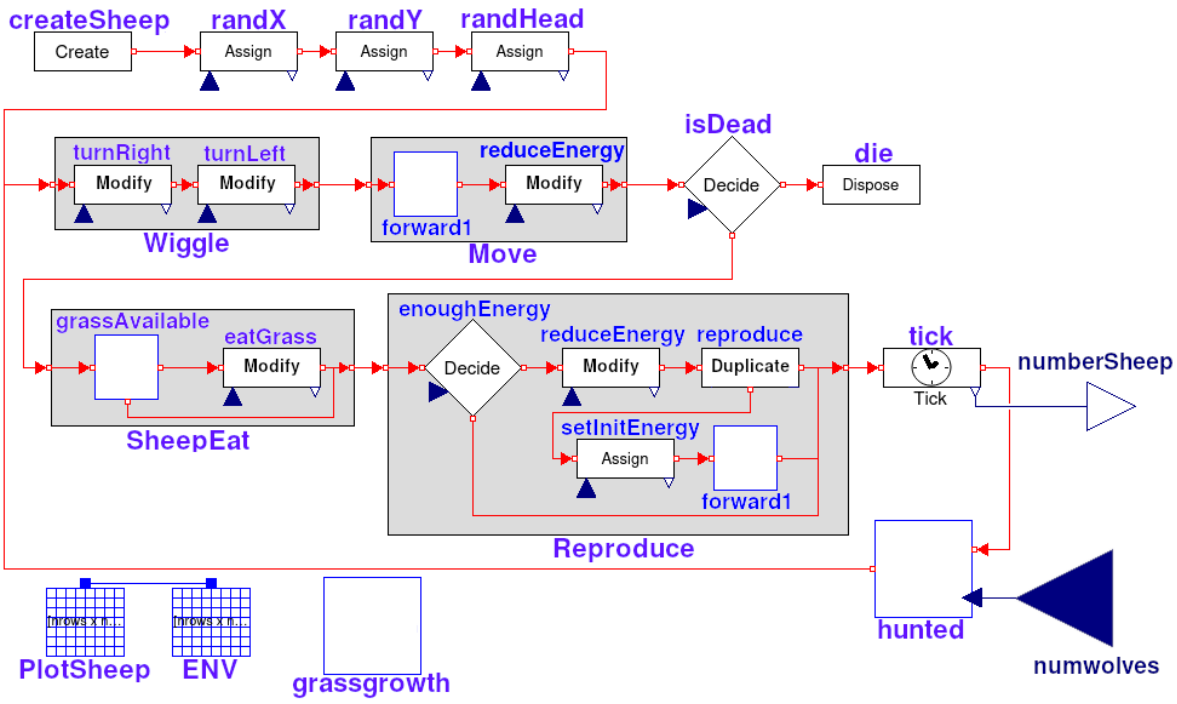

Figure 6. Flowchart diagram of the Sheep agent model.

movement the sheep spends some energy, defined by the parameter `moveCost`.

3. If the energy of the sheep reaches zero it dies. Otherwise, the sheep continues its behavior.
4. The next step for the sheep is eating grass. If some grass is available in its current position, the sheep increases its energy. Otherwise, no energy is gained.
5. Sheep reproduction is performed making copies of the current agents to create new sheep. If the sheep has enough energy to reproduce, a new copy of the sheep is created and its energy is reduced by `repCost`. The initial energy of the new sheep is also assigned.
6. After these steps, all sheep are gathered in the Tick module and wait for the next time step. The Tick module counts the number of sheep agents in the model, that is used as an input for the wolves model (i.e. `numberSheep` connector).
7. When the *tick* time is elapsed a new step is started. Before starting the new step, sheep can be hunted by wolves. The probability of being hunted is proportional to the number of wolves, which is received as input from the wolves model (i.e. the `numwolves` connector). Hunted sheep die and the rest continue to the next step starting with the wiggle action.

Model components used to describe the environment are also shown in Fig. 6. Each cell has two variables, one for the grass and another for the number of sheep present in the cell. The `ENV` module describes the bi-dimensional space and automatically generates the animation for the amount of grass in each cell. The `PlotSheep` module is added to automatically generate the animation of the sheep, showing their current positions in the space. Note the connection between the `ENV` and `PlotSheep` modules, as they generate animations for different variables of the same environment. The `grassgrowth` module is used to define the behavior of the environment, and represents the periodical growth of grass in each cell of the space. Note that this module is disconnected from the flowchart diagram, and thus it can only interact with the environment as it will never receive any agent.

6.2 Wolves

As mentioned above, wolves are represented using one of the differential equations, Eq. (1), described by Lotka and Volterra for predator-prey models,

$$\dot{y} = \delta x y - \gamma y \tag{1}$$

where, x is the number of sheep, y is the number of wolves, and δ and γ are the parameters describing their interactions.

The translation of Eq. (1) into Modelica code is straight forward, as shown in Listing 1. Note that `sheep` is an input

Listing 1: Modelica code of wolves model.

```
model WolvesEQ
  import Modelica.Blocks.Interfaces.*;
  parameter Real C, D;
  parameter Integer initWolves=5;
  RealInput sheep;
  RealOutput wolves (start=initWolves);
equation
  der(wolves) = C * sheep * wolves - D * wolves;
end WolvesEQ;
```

Table 2. Parameters of the Sheep-Wolves model.

Parameter	Value
moveCost	0.4
grassGrowthRate	0.03
energyGain	1.7
repCost	200
initEnergy	100
initSheep	100
initWolves	5
δ	0.00075
γ	0.005

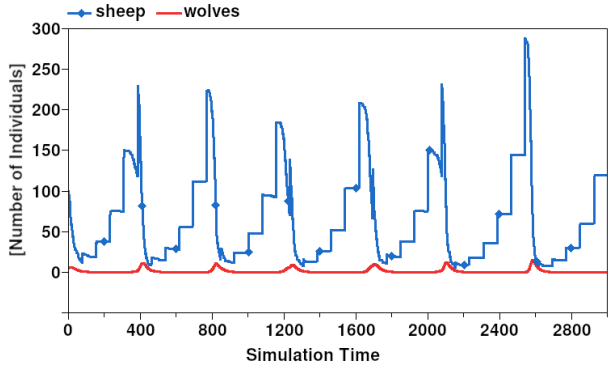

Figure 7. Sheep-Wolves simulation results.

to the model, computed as the count of sheep agents from the ABM, and `wolves` is used in the ABM to calculate the probability of wolves hunting sheep.

6.3 Simulation Results

The whole model is composed of the Sheep and Wolves models, connected to communicate the number of sheep and wolves. The `NumberSheep` port of the Sheep model is connected to the `sheep` port of the WolvesEQ model. The `wolves` port of the wolvesEQ model is connected to the `numwolves` port of the Sheep model.

The values of the parameters used for the simulation are shown in Table 2. The model is simulated during 3000 time steps. The evolution in the number of sheep and wolves is shown in Fig. 7. The staircase shape shown in the variation of the number of sheep is due to the initial generation of sheep agents as a single batch, that generates a synchronization in their reproduction. Examples of animations, showing the amount of grass and the position of sheep in the space, are shown in Fig. 8.

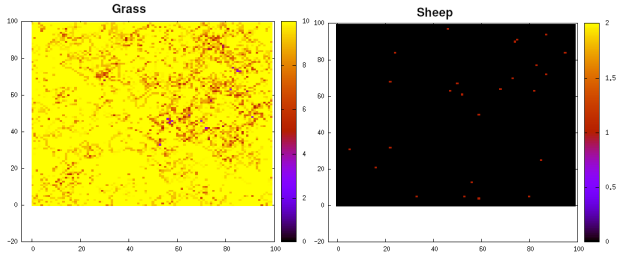

Figure 8. Examples of animations showing the amount of grass and the location of sheep agents.

In this case, an analyses of the performance is not included since the number of agents is variable in each simulation run. The simulation has been executed on an Intel Core i7-4720HQ CPU running Linux 5.15.1 and Dymola 2021. The execution time for the model with the described parameters is $7.53s$.

7 Market Hybrid Model

An example extracted from AnyLogic[74] is presented and implemented using ABMLib. This model represents the marketing of two products, A and B, in a population of individuals. Thus, the model is composed of: consumers, described as individual agents, and products, described used stock amounts and manufacturing rates. In this case, both parts, ABM and continuous-time, are more integrated. This model is used to demonstrate how different variables of the continuous-time part can influence different ABM modules, and viceversa. Also, alternatively to the `Sheep` model previously presented, in this case agent behavior is described using delayed processes.

7.1 Products

Products are manufactured and supplied to retailers. In AnyLogic, this supply chain is modeled using System Dynamics (SD) as shown in Fig. 9.

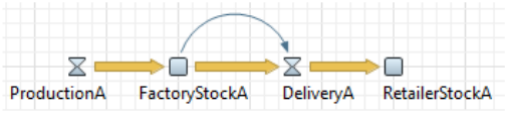

Figure 9. AnyLogic System Dynamics model of product manufacturing.

Listing 2: Modelica code of Product model.

```
model Product
  import Modelica.Blocks.Interfaces.*;
  parameter Real deliveryRate = 0.5;
  parameter Integer initialStock = 100;
  RealInput productionRate(start=15);
  RealInput purchases( start=0);
  Real Fstock( start=initialStock);
  Real Rstock( start=0);
  RealOutput stockOut(start=initialStock);
equation
  when sample(0,1) then
    stockOut =  pre(Rstock);
  end when;
  // Production
  der(Fstock) = productionRate - deliveryRate*Fstock;
  // Delivery
  der(Rstock) = deliveryRate * Fstock - purchases;
end Product;
```

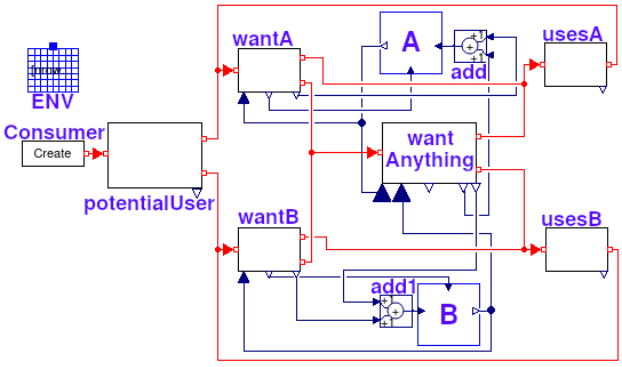

Figure 10. Flowchart diagram of Market model showing consumer processes, transitions and interactions with Product models.

The simple differential equations that correspond to the described SD model can be easily coded in Modelica, as shown in Listing 2. More complex models can also be graphically described in Modelica using the System Dynamics library[75].

The variation of the factory stock (`Fstock`) depends on the newly produced items (`productionRate`) minus the products delivered to retailers (`deliveryRate*Fstock`). The variation of the retailer stock (`Rstock`) depends on the products received from the factory (`deliveryRate*Fstock`) minus the products sold to customers (`purchases`). Note that `productionRate` and `purchases` are inputs to the model, that will be calculated in the `Consumers` model, and `stockOut` is the output, calculated as a periodical sample of the retailer stock.

7.2 Consumers

Consumers are represented as individual agents. Alternatively to the `Sheep` model previously presented, in this case agent behavior is described using delayed processes. Each consumer can be in any of the following processes: *potentialUser*, *wantX*, *wantAnything* and *usesX* (where *X* can be either *A* or *B*, depending on the selected product). The flowchart diagram is shown in Fig. 10. This diagram also includes two `Product` models, `productA` and `productB` that correspond to products A and B, and the interactions between them and the agents.

A fixed number of agents, or consumers, is initially created in the model. All these agents start in the *potentialUser* process, as potential consumers of any of the products. They decide between both products due to advertisement or to interactions with other agents that already use any of the products. Once the product type is decided, agents change to the corresponding *wantX* process. While in the *wantX* process, agents try to buy the product from the retailer. Availability of a product is defined by the value of the `stockOut` variable calculated by the corresponding `Product` model. It the product is available, the agent becomes a user and moves to the *usesX* process. Otherwise, the agent waits for the product to be available. Agents waiting for a product during more than 2 days will change to the *wantAnything* process, meaning that they give up waiting for a particular product and now they will buy the first product available of any type. After using the decided product for a variable amount of time, between 17 and 23 days, agents decide to discard the product and move to the *wantX* process to buy a new one.

The modules used to describe the processes of the diagram (`potentialUser`, `wantA`, `wantB`, `wantAnything`, `usesA` and `usesB`) are implemented as atomic PDEVS models. Each module is used to represent one process. Agents remain in the processes until any of the conditions for the transitions to other processes are met. The evaluation of these conditions is described inside the internal transition function of the modules. Agents received by a module are stored in an internal queue. Periodically, the conditions for process transition are evaluated for all agents in the internal queue. If an agent meets a condition it leaves the internal queue and is sent to corresponding new process using the output function.

The interactions between consumers and products are defined as follows. The `wantA` process receives as input the current stock of the product A (i.e., the `stockOut` variable), which is used to evaluate the availability of products for consumers. Its outputs are: a) the number of consumers waiting for product A, that is used as `productionRate` since production equals demand in this model, and; b) the number of consumers that have purchased available products, that is used to calculate the variation in the stock of the retailers. The same interactions

are required for the processes `wantB` (involving product B instead of A) and `wantAnything` (involving both products A and B at the same time). Also, two `add` models are included in the diagram to calculate the total amount of purchases for each product, summing the purchases from the *wantX* and the *wantAnything* processes.

7.3 Simulation Results

The system has been simulated for 100 days, with an initial number of 1000 consumers. Process transitions are periodically updated every simulated day. The evolution of the number of consumers in the different processes, *wait* and *use*, is shown in Fig. 11. A higher priority is given to the evaluation of the conditions for product A, that results in a bigger adoption by consumers when compared to product B. The oscillations shown in the figure correspond to the periodical product discard after use. Otherwise the results will correspond to a typical logistic curve.

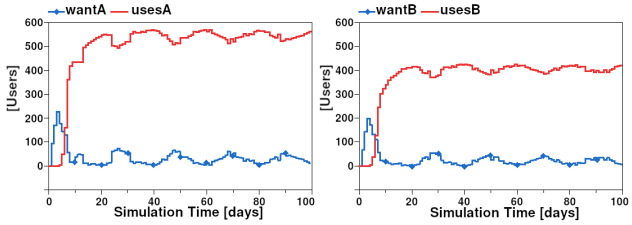

Figure 11. Simulation results for the Market model.

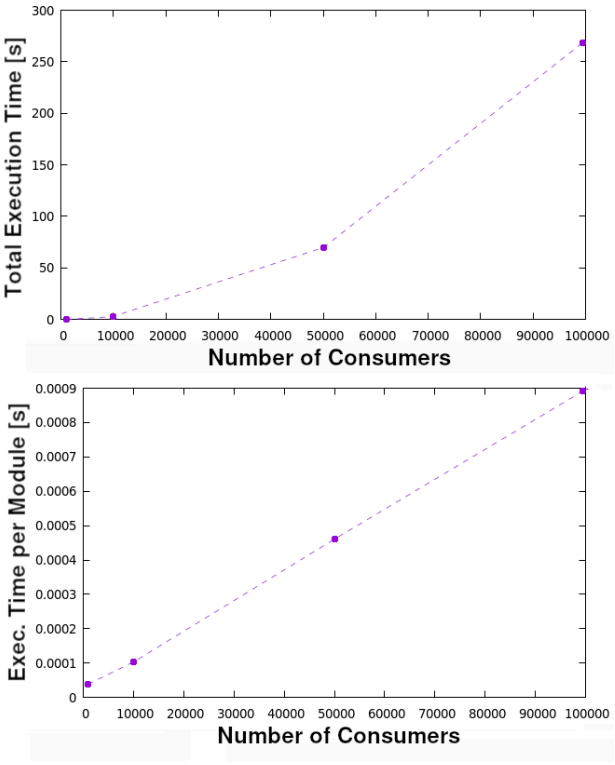

Figure 12. Evolution of total and per module execution times for the Market model.

Table 3. Performance of the Market model.

Consumers	Exec. Time	Exec. Time per Module
1000	$0.114s$	$0.000038s$
10000	$3.1s$	$0.000103s$
50000	$69.2s$	$0.000462s$
100000	$269s$	$0.000896s$

The performance and scalability have been also evaluated in this model. The model has been simulated with a variable number of consumers (e.g., 1000, 10000, 50000 and 100000). The simulations have been executed on an Intel Core i7-4720HQ CPU running Linux 5.15.1 and Dymola 2021. As with the basic traffic model, the total execution time and the execution time per module are shown in Table 3. The results are analogous to the basic traffic model, with a linear evolution of the per module execution time. These results are also graphically shown in Fig. 12.

8 Conclusions

An approach for describing ABMs using the Modelica language has been presented. Previous contributions from other authors oriented to combine ABM with continuous-time equations present several limitations, such as the description of continuous-time dynamics only using explicit ODE, the requirement to perform manual manipulations to the model equations in order to generate the simulation code, or the use of multiple tools to describe the entire system in a co-simulation approach. The Modelica language represents the state-of-the-art in equation-based modeling of cyber-physical systems. However, Modelica does not facilitate the description of ABMs, since Modelica does not allow variations in the number of variables or equations of the model during the simulation, nor the use of dynamic data structures. This functionality is basic to describe the variations in the behavior of agents during the simulations. The proposed approach facilitates describing ABMs using the functionality provided by Modelica, and combining the ABMs described in Modelica with other Modelica models. Our solution is grounded in two concepts: the description of agents using a process-oriented approach, as entities flowing across a flowchart diagram; and the use of PDEVS as a formalism to provide clear semantics for the description of the processes performed by agents. Since Modelica is a general-purpose object-oriented equation-based modeling

language, this combination provides a powerful framework for the study of complex systems using a multi-formalism modeling approach.

Acknowledgements

This research was supported by the Ministerio de Economía y Competitividad of Spain, DPI2013-42941-R grant.

References

1. Railsback SF and Grimm V. *Agent-Based and Individual-Based Modeling - A Practical Introduction*. 2nd ed. Priceton, NJ, USA: Priceton University Press, 2019. ISBN 9780691190839.
2. Dam KV, Nikolic I and Lukszo Z (eds.) *Agent-Based Modelling of Socio-Technical Systems*. Springer Netherlands, 2013. ISBN 9789401782685.
3. Åström KJ, Elmqvist H and Mattsson SE. Evolution of continuous-time modeling and simulation. In *Proceedings of the* 12^{th} *European Simulation Multiconference*. Manchester, UK, 1998. pp. 9–18.
4. Fritzson P. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley-IEEE Computer Society Pr, 2014. ISBN 9781118859124.
5. Cellier FE and Kofman E. *Continuous System Simulation*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 0387261028.
6. Sanz V, Bergero F and Urquia A. An approach to agent-based modeling with Modelica. *Simul Model Pract Theory* 2018; 83: 65–74. DOI:10.1016/j.simpat.2017.12.012.
7. Chow ACH and Zeigler BP. Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In *Proceedings of the* 26^{th} *Winter Simulation Conference*. Orlando, FL, USA, 1994. pp. 716–722.
8. Zeigler BP, Muzy A and Kofman E. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*. 3rd ed. New York: Academic Press, 2018. ISBN 9780128133705.
9. Sanz V, Urquia A and Dormido S. Parallel DEVS and process-oriented modeling in Modelica. In *Proceedings of the* 7^{th} *International Modelica Conference*. Como, Italy, 2009. pp. 96–107.
10. Zhang M, Seck M and Verbraeck A. A DEVS-based M&S method for lange-scale multi-agent systems. In *Proceedings of the 2013 Summer Computer Simulation Conference*. Toronto, Ontario, Canada, 2013.
11. Wooldridge M. *An Introduction to Multiagent Systems*. Wiley, 2009. ISBN 9780470519462.
12. Rouff CA, Hinchey M, Rash J et al. (eds.) *Agent Technology from a Formal Perspective*. NASA Monographs in Systems and Software Engineering, Springer, 2006. ISBN 9781846282713.
13. Zhu H. Slabs: A formal specification language for agent-based systems. *Int J Softw Eng Know* 2001; 11(05): 529–558. DOI:10.1142/S0218194001000657.
14. Maheshwari A, Kenley CR and DeLaurentis DA. Creating executable agent-based models using SysML. In *Proceedings of the* 25^{th} *Annual INCOSE International Symposium*. 2015. pp. 1263–1277.
15. Tsadimas A, Kapos GD, Dalakas V et al. Integrating simulationcapabilities into SysML for enterprise information system design. In *Proceedings of the* 9^{th} *International Conference on System of Systems Engineering*. 2014. pp. 272–277.
16. Xu H and Shatz SM. An agent-based Petri Net model with application to seller/buyer design in electronic commerce. In *Proceedings of the* 5^{th} *International Symposium on Autonomous Decentralized Systems*. 2001. pp. 149–154.
17. Pouyan AA, Beigi AH and Kadkhoda M. An agent-based model for virtual tourism using Object Petri Nets. In *Proceedings of the* 5^{th} *International Conference on Circuits, Systems, Electronics, Control and Signal Processing*. 2006. pp. 149–154.
18. Köhler M, Langer R, von Lüde R et al. Socionic multi-agent systems based on reflexive petri nets and theories of social self-organisation. *J Artif Soc Soc Simul* 2007; 10(1): 3.
19. Plà A, Gay P, Meléndez J et al. Petri net-based process monitoring: a workflow management system for process modelling and monitoring. *J Intell Manuf* 2014; 25: 539–554. DOI:10.1007/s10845-012-0704-z.
20. Holcombe M, Coakley S and Smalwood R. A general framework for agent-based modelling of complex systems. In *Proceedings of the European Conference on Complex Systems (ECCS'06)*. 2006.
21. Boucheri A, Khebaba A and Belala F. Rewriting logic based approach for the formalization of critical systems based on multi-agent system. *Int J Comput Appl* 2011; 13(2): 6–13. DOI:10.5120/1755-2392.
22. Reinhardt O and Uhrmacher AM. An efficient simulation algorithm for continuous-time agent-based linked lives models. In *Proceedings of the* 50^{th} *Annual Simulation Symposium*. 2017.
23. Vangheluwe HLM. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*. IEEE Computer Society Press, 2000. pp. 129–134.
24. Barros FJ. Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation. In *Proceedings of the* 27^{th} *Conference on Winter simulation*. Arlington, VA, USA, 1995. pp. 781–785.
25. Barros FJ. Modelling formalisms for dynamic structure systems. *ACM Trans Model Comput Simul* 1997; 7(4): 501–515. DOI:10.1145/268403.268423.

26. Uhrmacher AM. Dynamic structures in modeling and simulation: A reflective approach. *ACM Trans Model Comput Simul* 2001; 11(2): 206–232. DOI:10.1145/268403.268423.
27. Kim JH and Kim TG. DEVS-based framework for modeling/simulation of mobile agent systems. *Simulation* 2001; 76(6): 345–357. DOI:10.1177/003754970107600603.
28. Uhrmacher AM, Himmelspach J, Röhl M et al. Introducing variable ports and multi-couplings for cell biological modeling in DEVS. In *Proceedings of the 2006 Winter Simulation Conference*. 2006. pp. 832–840.
29. Müller JP. *Towards a Formal Semantics of Event-Based Multi-Agent Simulations*. Berlin, Heidelberg: Springer-Verlag. ISBN 9783642019906, 2009. p. 110–126. DOI: 10.1007/978-3-642-01991-3_9.
30. Steiniger A, Krüger F and Uhrmacher AM. Modeling agents and their environment in multi-level-DEVS. In *Proceedings of the 2012 Winter Simulation Conference*. 2012. pp. 2629–2640.
31. Goldstein T, Breslav S and Khan A. A symmetric formalism for discrete event simulation with agents. In *Proceedings of the 2018 Winter Simulation Conference*. Gothenburg, Sweden, 2018.
32. Foguelman D, Henning P, Uhrmacher A et al. Eb-devs: A formal framework for modeling and simulation of emergent behavior in dynamic complex systems. *J Comput Sci* 2021; 53: 101387. DOI:10.1016/j.jocs.2021.101387.
33. Bae JW, Lee G and Moon IC. Formal specification supporting incremental and flexible agent-based modeling. In *Proceedings of the 2012 Winter Simulation Conference*. 2012. pp. 1–12.
34. Bouanan Y, Zacharewicz G and Vallespir B. DEVS modelling and simulation of human social interaction and influence. *Eng Appl Artif Intell* 2016; 50: 83–92. DOI:10.1016/j.engappai.2016.01.002.
35. Uhrmacher AM and Schattenberg B. Agents in discrete event simulation. In *Proceedings of the* 10^{th} *European Simulation Symposium*. Nottingham, UK, 1998.
36. Camus B, Bourjot C and Chevrier V. Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems. In *Proceedings of the Spring Simulation Multi-Conference*. Alexandria, VA, USA, 2015. pp. 85 – 90.
37. Bouanan Y, Zacharewicz G, Ribault J et al. Discrete event system specification-based framework for modeling and simulation of propagation phenomena in social networks: application to the information spreading in a multi-layer social network. *Simulation* 2019; 95(5): 411–427. DOI: 10.1177/0037549718776368.
38. Kravari K and Bassiliades N. A survey of agent platforms. *J Artif Soc Soc Simul* 2015; 18(1): 11. DOI:10.18564/jasss.2661.
39. Abar S, Thodoropoulos GK, Lemarinier P et al. Agent based modelling and simulation tools: A review of the state-of-art software. *Comput Sci Rev* 2017; 24: 13–33. DOI: 10.1016/j.cosrev.2017.03.001.
40. Macal CM. Tutorial on agent-based modeling and simulation: ABM design for the zombie apocalypse. In *Proceedings of the 2018 Winter Simulation Conference*. Gothenburg, Sweden, 2018. pp. 207– 221.
41. Forrester JW. *Principles of Systems*. Waltham, MA, USA: Pegasus Communications, 1969. ISBN 9781883823412.
42. Wilensky U and Rand W. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with Netlogo*. Cambridge, MA, USA: MIT Press, 2015. ISBN 9780262731898.
43. Company TA. Anylogic simulation software website, 2021. URL http://www.anylogic.com. Accessed May 2021.
44. Wang H, Cao R and Zeng W. Multi-agent based and system dynamics models integrated simulation of urban commuting relevant carbon dioxide emission reduction policy in China. *J Clean Prod* 2020; 272. DOI:10.1016/j.jclepro.2020.122620.
45. Martin R and Schlüter M. Combining system dynamics and agent-based modeling to analyze social-ecological interactions—an example from modeling restoration of a shallow lake. *Front Environ Sci* 2015; 3. DOI:10.3389/fenvs.2015.00066.
46. Vincenot CE, MAzzoleni S, Moriya K et al. How spatial resource distribution and memory impact foraging success: A hybrid model and mechanistic index. *Ecol Complex* 2015; 22: 139–151. DOI:10.1016/j.ecocom.2015.03.004.
47. Swinerd C. *On the Design of Hybrid Simulation Models, Focussing on the Agent-Based System Dynamics Combination*. PhD Thesis, Cranfield Defence And Security Centre For Simulation And Analytics, Cranfield, UK, 2014.
48. Kofman E. Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing* 2004; 25(5): 1771–1797.
49. Bobashev GV, Goedecke DM, Yu F et al. A hybrid epidemic model: Combining the advantages of agent-based and equation-based approaches. In *Proceedings of the* 39^{th} *Winter Simulation Conference*. Washington, DC, USA, 2007. pp. 1532–1537.
50. Gräbner C, Bale CSE, Furtado BA et al. Getting the best of both worlds? developing complementary equation-based and agent-based models. *Comput Econ* 2019; 53(2): 763 – 782. DOI:10.1007/s10614-017-9763-8.
51. Bradhurst RA, Roche SE, East IJ et al. A hybrid modeling approach to simulating foot-and-mouth disease outbreaks in australian livestock. *Front Environ Sci* 2015; 3. DOI: 10.3389/fenvs.2015.00017.
52. Caudill L and Lawson B. A hybrid agent-based and differential equations model for simulating antibiotic resistance in a hospital ward. In *Proceedings of the 2013*

Winter Simulation Conference. Washington, DC, USA, 2013. pp. 1419–1430.
53. Hunter E, Mac Namee B and Kelleher J. A hybrid agent-based and equation based model for the spread of infectious diseases. *J Artif Soc Soc Simul* 2020; 23(4): 14. DOI: 10.18564/jasss.4421.
54. Banos A, Corson N, Lang C et al. 2 - multiscale modeling: Application to traffic flow. In Banos A, Lang C and Marilleau N (eds.) *Agent-based Spatial Simulation with NetLogo, Volume 2*. Elsevier, 2017. pp. 37–62. DOI:10.1016/B978-1-78548-157-4.50002-9.
55. Lee EA and Zheng H. Operational semantics of hybrid systems. In *Proceedings of the International Workshop on Hybrid Systems: Computation and Control*. 2005. pp. 25–53.
56. Duboz R, Éric Ramat and Preux P. Scale transfer modeling: using emergent computation for coupling an ordinary differential equation system with a reactive agent model. *Syst Anal Model Simul* 2003; 43(6): 793–814.
57. Novak P, Kadera P and Wimmer M. Agent-based modeling and simulation of hybrid cyber-physical systems. In *Proceedings of the 3^{rd} IEEE International Conference on Cybernetics*. 2017. pp. 1–8.
58. Gomes C, Thule C, Broman D et al. Co-simulation: A survey. *ACM Comput Surv* 2018; 51(3): 49:1–49:33. DOI: 10.1145/3179993.
59. Modelica Association Project FMI. Functional Mock-up Interface for model exchange and co-simulation (v2.0.2), 2020. URL http://www.fmi-standard.org. Accessed May 2021.
60. IEEE Computer 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. Accessed May 2021.
61. Palmintier B, Krishnamurthy D, Top P et al. Design of the helics high-performance transmission-distribution-communication-market co-simulation framework. In *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. 2017. pp. 1–6. DOI:10.1109/MSCPES.2017.8064542.
62. Tang Y, Tai W, Liu Z et al. A hardware-in-the-loop based co-simulation platform of cyber-physical power systems for wide area protection applications. *Appl Sci* 2017; 7(12). DOI: 10.3390/app7121279.
63. Constantin A, Löwen A, Ponci F et al. Dymola-JADE co-simulation for agent-based control in office spaces. In *Proceedings of the 12^{th} International Modelica Conference*. Prague, Czech Republic, 2017. pp. 345–351.
64. Marilleau N, Lang C and Giraudoux P. Coupling agent-based with equation-based models to study spatially explicit megapopulation dynamics. *Ecol Modell* 2018; 384: 34–42. DOI:10.1016/j.ecolmodel.2018.06.011.
65. Größler A, Stotz M and Schieritz N. A software interface between system dynamics and agent-based simulations: Linking Vensim and Repast. In *Proceedings of the 21^{st} International Conference of the System Dynamics Society*. New York City, NY, USA, 2003.
66. Sanz V, Urquia A, Cellier FE et al. System modeling using the Parallel DEVS formalism and the Modelica language. *Simul Model Pract Theory* 2010; 18(7): 998–1018. DOI: 10.1016/j.simpat.2010.03.004.
67. Modelica Association. Modelica – A unified object-oriented language for systems modeling. Language spec. v. 3.5, 2021. URL https://modelica.org/documents/MLS.pdf. Accessed May 2021.
68. Sanz V and Urquia A. Modelica extensions for supporting message passing communication. In *Proceedings of the 7^{th} International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. 2016. pp. 21–28.
69. Sanz V and Urquia A. Cyber–physical system modeling with Modelica using message passing communication. *Simul Model Pract Theory* 2022; 117: 102501. DOI:10.1016/j.simpat.2022.102501.
70. Bergero F and Kofman E. PowerDEVS: a tool for hybrid system modeling and real-time simulation. *Simulation* 2011; 87(1-2): 113–132. DOI:10.1177/0037549710368029.
71. Sanz V, Urquia A, Cellier FE et al. Hybrid system modeling using the SIMANLib and ARENALib Modelica libraries. *Simul Model Pract Theory* 2013; 37: 1–17. DOI:10.1016/j.simpat.2013.05.005.
72. Zeigler BP. Embedding DEV&DESS in DEVS. In *Proceedings of the DEVS Integrative M&S Symposium*. Huntsville, AL, USA, 2006.
73. Railsback SF, Lytinen SL and Jackson SK. Agent-based simulation platforms: Review and development recommendations. *Simulation* 2006; 82(9): 609–623. DOI: 10.1177/0037549706073695.
74. Borshchev A. How to build a combined Agent-Based System Dynamics model in AnyLogic. In *Proceedings of the 26^{th} International Conference of the System Dynamics Society*. 2008. pp. 402–4023.
75. Cellier FE. World3 in Modelica: Creating System Dynamics models in the Modelica framework. In *Proceedings of the 6^{th} International Modelica Conference*. Bielefeld, Germany, 2008. pp. 393–400.

Author biographies

Victorino Sanz received his M.S. in Computer Science in 2004 from Universidad Politecnica de Madrid and his Ph.D. in Computer Science from Universidad Nacional de Educación a Distancia (UNED). He is assistant professor at Dpto. Informática y Automática, Universidad Nacional de Educación a Distancia (UNED) since 2010. His research interests focus on hybrid system modeling and simulation using the Modelica language.

Alfonso Urquia received his M.S. degree in Physics in 1992 from Universidad Complutense de Madrid and his Ph.D. in Physics in 2000 from Universidad Nacional de Educación a Distancia (UNED). Since 2002, he has been working as an Associate Professor at Dpto. Informática y Automática, UNED, in Madrid, Spain. His research is focused on mathematical modeling and computer simulation.