Simulation Modelling Practice and Theory 83 (2018) 65–74

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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

An approach to agent-based modeling with Modelica

Victorino Sanz[a,*], Federico Bergero[b], Alfonso Urquia[a]

[a] Departamento de Informática y Automática, UNED, Juan del Rosal, 16, Madrid, 28040, Spain
[b] French-Argentine International Center for Information and Systems Sciences (CIFASIS-CONICET), 27 de Febrero 210 bis, S2000EZP, Rosario, Argentina

ARTICLE INFO

Article history:
Available online 26 December 2017

Keywords:
Agent-based model
Modelica
Hybrid systems

ABSTRACT

Modelica is a free, general-purpose object-oriented equation-based modeling language. It is mainly designed to describe systems using the physical modeling approach. Our proposal to describe Agent-Based Models (ABMs) in Modelica is discussed in this manuscript. The contribution of the presented work is twofold: firstly, to analyze the conceptual requirements to describe ABMs in Modelica; and secondly, to develop a prototype implementation following the previous analysis. Agents are described using a message passing communication mechanism previously proposed by the authors. Additional extensions to this mechanism are proposed in order to describe agent interactions. The environment, where the agents live, is described as a two-dimensional cellular automaton. A new Modelica library, named ABMLib, developed to support this functionality, is presented. A prototype implementation of the message passing mechanism and ABMLib models has been performed to demonstrate the functionality of the library as a proof-of-concept for this proposal. The library is freely available at www.euclides.dia.uned.es/vsanz.

1. Introduction

Agent-Based Models (ABMs) are discrete-event models composed of a variable number of “living” objects, named agents, that behave following a pre-defined set of rules (i.e., agent behavior), and interact among them and with their environment (i.e., the physical space where the agents “live”) [1]. The individual behavior of each agent is defined using simple rules, or algorithms, but the simulation of the whole model may lead to complex and emergent behaviors. In this manuscript, the description of ABMs using the Modelica language is discussed.

Modelica supports the description of mathematical models following the physical modeling paradigm [2]. Modelica models are described as a combination of acausal equations, algorithms and events, using the hybrid DAE formalism (cf. [3] for a detailed description of the formalism). The causality of the model is automatically computed by means of symbolic manipulations of the equations before generating the executable code [4].

The description of ABMs in Modelica could be used to perform a qualitative description of models, or parts of models, in contrast with the quantitative approach given by equation-based modeling [5]. ABMs can be used to represent heterogeneous objects in the model (i.e., agents of the same type are used to represent different individuals in the model with different characteristics or even different behavior) while equations are used to represent homogeneous quantities. Adaptive

* Corresponding author.
E-mail addresses: vsanz@dia.uned.es (V. Sanz), bergero@cifasis-conicet.gov.ar (F. Bergero), aurquia@dia.uned.es (A. Urquia).

https://doi.org/10.1016/j.simpat.2017.12.012
1569-190X/© 2017 Elsevier B.V. All rights reserved.

or learning behaviors can also be described in terms of ABMs. In this way, the combination of ABMs with other Modelica models enhances the functionality of the language and the description of more complex hybrid systems.

Other authors have made efforts to combine equation-based models and ABMs. The LEADSTO language combines dynamic systems of equations with ABMs [6]. Another approach has been to combine ABMs with System Dynamics, using Anylogic, to describe health-care systems [7]. Humans are described using agents, while System Dynamics is used to describe disease dynamics. A similar approach is used to simulate antibiotic resistance in hospital wards [8]. Intra-host dynamics (i.e., bacterial-level processes inside individuals) are described using differential equations, while inter-host dynamics (i.e., relations between humans) are described using ABMs. Also, Dymola and JADE have been combined using a co-simulation approach to describe control for office spaces [9].

The proposal presented in this manuscript is to describe agents as individual messages flowing between components of a flowchart diagram, which is analogous to a coupled DEVS model [10]. Agent behavior is described by the components of the flowchart diagram independently of the environment, but agents can interact with it. The environment is represented as a two-dimensional cellular automaton, using CellularAutomataLib2 [11]. Some extensions to the message passing communication mechanism, previously proposed by the authors [12], are presented to describe agent's interactions. All this functionality is included in a new Modelica library, named ABMLib, designed and developed by the authors. ABMLib models can be also combined with other Modelica models. ABMLib approach is similar to the description of systems using process calculus, where the processes communicate using messages [13]. However, in this proposal agent actions are not synchronized by means of communication rendezvous, but at discrete points in time [14]. ABMLib approach is also similar to actor-oriented models, but in this case messages represent the agents themselves and not the data flowing between actors.

A prototype implementation of the message passing mechanism and the new library has been developed and tested, and an application example using a Lotka–Volterra model is presented in this manuscript. The presented model combines ABMs with continuous-time equations in order to illustrate the benefits of supporting ABMs in Modelica.

The structure of the manuscript is as follows. The requirements to describe ABMs in Modelica are discussed in Section 2. The message passing communication mechanism is briefly described in Section 3, together with the proposed extensions required for ABM development. The ABMLib library is described in Section 4, and the combination of ABMLib models with other Modelica models is described in Section 5. The Lotka–Volterra model described using ABMLib and other Modelica functionality is presented in Section 6. Finally, some conclusions and future work ideas are given in Section 7.

2. Requirements for describing ABMs in Modelica

Modelica and ABM are conceptually different. ABMs are composed of agents, environments and interactions [1]. Modelica models are mainly described by means of equations, while the behavior of agents is described using rules. Agents can be created or removed during the simulation run, while the number of variables and equations in Modelica has to remain constant. ABMs are usually dependent on the spatial coordinates, with the agents moving and interacting around the environment, while the only independent variable in Modelica is the time. The functionality of the Modelica language that can be used to describe these characteristics is discussed below.

- Agents are described by means of their state and behavior. Modelica simple data types and complex data structures can be used to represent the state of the agents. Usually, the behavior of agents is described as a set of rules or actions. Modelica algorithm sections can be used to describe the behavior of the agents.
  However, Modelica does not support changes in the number of variables and/or equations during the simulation. These changes occur during the creation or removal of agents from the model. An additional mechanism needs to be used to represent the variation of agents during the simulation.
  The proposed approach is to graphically represent the behavior of agents as a flowchart diagram. Agents are represented as messages that are sent from one component to another in the diagram. The components represent the individual actions performed by agents. The diagram includes components to create agents and to remove them from the simulation. The components of the diagram communicate using the message passing communication mechanism previously proposed by the authors [12], which is capable of dealing with a variable number of messages during the simulation run.
- The environment represents the physical space where the agents behave and interact, and can be defined in multiple ways depending on the necessities of the model. Some authors consider the environment as a set of passive agents, since they can also have state and behavior [15]. Also, the environment could only represent feasible agent interactions (e.g., links in a social network).
  As a first approach, our proposal is to represent the environment as a cellular automaton. This is a two-dimensional square lattice, where the states of the cells are represented using some variables and all the cells share the same behavior defined using a transition function. The state of the cells is periodically updated using the transition function. The CellularAutomataLib2 library, developed by the authors, supports the description and efficient simulation of cellular automata in Modelica [11].
- Agents can interact with other agents or with the environment. Modelica provides functionality to access models and variables in the hierarchy of models and libraries of models. This functionality includes the dot-notation, that allows to access models in other libraries or packages, or the `inner/outer` variable modifiers that can be used to access variables and models defined in another part of the hierarchy. Additionally, CellularAutomataLib2 includes interface models

to combine cellular automata with other Modelica models. These interface models, combined with the other Modelica functionality, can be used to observe or modify the state of the environment depending on the behavior of the agents.
On the other hand, since each agent is defined as a message and they move between the components of the flowchart diagram, the state of the agents is not accessible. Additional functionality is required to describe agent interactions. An extension to the message passing communication mechanism, with the concepts of sets and subsets of messages, is proposed. This new extension is detailed next.

3. Message passing communication mechanism

Coupling relationships between Modelica components are described using: the `connector` class to define the ports of their interfaces, and the `connect` sentence to define the connection between connectors. Each connector is composed of a set of variables that by default are defined as *effort*, and may be defined as *flow* using the `flow` modifier. The *effort* variables of the connected connectors are equaled, and the *flow* variables are summed up and the sum is equaled to zero. These connect-equations are added to the model and taken into account with the rest of the model in order to perform the causality analysis.

Message passing corresponds to the transference of one or multiple impulses of information between models, in contrast with the equation-based connections previously described. Both communication approaches are conceptually different, and thus the authors proposed to include new functionality in Modelica in order to support message passing communication. A new class, named `buffer`, and a new sentence, named `couple`, are introduced to describe communication buffers and their relationships. The proposed extensions are based on the P-DEVS model communication approach [10]. These extensions are briefly introduced in Section 3.1, and a detailed description can be found in [12].

Message passing communication can serve as a general purpose extension to Modelica, since it can be applied to describe systems using multiple approaches, such as DEVS, process-oriented models and distributed models, among others. Message passing libraries already exist for general purpose programming languages, like the Message Passing Interface (MPI) [16]. In this manuscript the use of message passing communication is focused on the description of ABMs. However, the mechanism proposed in [12] needs to be extended to describe the interactions between agents. These additional extensions are presented in Section 3.2.

3.1. Previous proposal

The elements of the communication mechanism are the messages, the buffers and the communication channel. A brief description of the proposed message passing communication mechanism is presented.

- *Messages* represent the information transported from one model to another. They can be defined using current Modelica functionality, such as basic data types (e.g., `Integer`, `Real` or `Boolean`) or more complex data structures using `record` classes.
- *Buffers* constitute the data structures used to store messages. They can be used to store a variable number of messages in a model, or as interface ports for communicating with other models using the `input` and `output` Modelica modifiers. The messages in a buffer can be read using array-like and dot-notation (e.g., `buffer[1]` for accessing the first message in the buffer, or `buffer[1].var` for accessing a variable in the first message). Two special functions, named `put` and `pop`, can be used to insert and extract messages to and from the buffer respectively. The number of messages in a buffer is automatically computed and stored in a variable named `size` (e.g., `buffer.size`).
- The *Channel* is used to define the communication relationships between models. Similarly to the `connect` sentence, a new sentence, named `couple`, is introduced to define relationships between input and output buffers. Point to point and collective (i.e., 1-N, N-1 and N-N) communication are allowed.

As described above, messages can be used to represent agents. The actions required to describe the behavior of agents can be defined as the actions associated with the management of input messages in a model. This modeling approach can be described in terms of the P-DEVS formalism and is similar to the implementation of SIMANLib and ARENALib, which are two Modelica libraries developed by the authors that support the process-oriented modeling worldview [17].

The behavior of each agent type is defined using a flowchart diagram. All the agents of a particular type will be stored in the buffers of the components of their diagram. An additional extension to this message passing communication mechanism is presented next to allow agent interactions.

3.2. Additional language extensions

In order to allow the interaction between agents, two additional data structures are proposed as new Modelica classes: `msgset` and `msgsubset`. These new classes constitute language extensions, since their functionality cannot be described using Modelica.

The `msgset` class represents the set of messages that are stored in the buffers of a model and its components, including interface ports and internal buffers. The contents of the `msgset` object are automatically updated by the simulator, adding or removing messages as required. In order to avoid duplicities between `msgset` objects and the buffers of the model, the

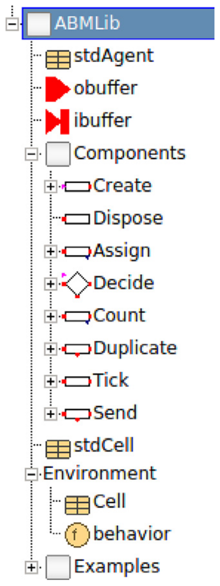


Fig. 1. ABMLib library architecture.

msgset object should only store references to the actual messages. The msgset class can be used to access the information carried by the messages using array-like and dot-notations (e.g., set[1].a defines the value of the variable a of the first message in the set). Note that the index of the array is not representative since buffers and sets are unordered data structures, so additional mechanisms have to be used to identify individual messages (e.g., unique message identification numbers). The number of messages in the set is automatically computed and stored in a pre-defined size variable. In ABMs, a msgset object can be used to have access to all the agents flowing in a flowchart diagram.

However, the interaction between agents is usually restricted to a particular reduced set of agents. The msgsubset class represents a subset of messages that share a given condition (e.g., agents located in the same position or in the neighborhood). The subset has two parameters: the sources that are the set or sets from where the subset is composed of; and the condition that defines the common characteristic for all the messages in the subset. The condition is used to compare the values of the messages in the sources with other values. The ':' Modelica operator could be used to index all the messages in the sources (e.g., sources[:].X == x, meaning that all the messages in the sources with a value of X equal to x will belong to the subset). The authors propose to simplify the notation and remove the reference to the array. In that case the same condition will be written as sources.X == x.

An additional problem arises when the condition is based on the values of the variables of a message that is currently being processed. For example, an agent that needs to interact with other agents located in the same spatial position. In this case the condition for the msgsubset could be written as sources.X == agent.X and sources.Y == agent.Y, where X and Y are the spatial coordinates of the agents. That condition will be different for each agent, depending on their spatial position, so a different subset has to be computed for each agent. The introduction of a new Modelica operator, named msg, is proposed to facilitate the description of such conditions. The value of msg corresponds to the value of the last message extracted from one buffer using the pop function (i.e., msg corresponds to the last active message in a model). Different models may have different values for msg, since messages can be simultaneously received and managed. Thus, the condition defined above could be written as sources.X == msg.X and sources.Y == msg.Y. The msg operator can also be used to access the variables of the active message, to define other conditions or parameter values in the components of the flowchart diagram.

4. The ABMLib library

ABMLib is a new Modelica library that facilitates the description of ABMs in Modelica, and their combination with other Modelica models. The architecture of the library is shown in Fig. 1. The library is composed of:

- stdAgent model that is used to describe a standard agent. It includes some variables common to all agents.
- obuffer and ibuffer that represent the output and input buffers used to describe the interfaces of the components of the flowchart diagram.
- Components package that include some basic components that can be used to describe the behavior of agents.
- stdCell and Environment that can be used to represent a basic agent environment. As described above, the CellularAutomataLib2 library provides better functionality to describe the environment and its use is encouraged.
- Examples package that includes some examples of use.

```
partial record stdAgent
  Integer X; // X coordinate
  Integer Y; // Y coordinate
  Integer head; // head orientation in degrees
  Real color; // color for graphical animation
  Integer ID; // identification number
  String name; // agent type name
  buffer origin; // buffer of origin before send
end stdAgent;

record car extends stdAgent(name="car");
  Real kms; // kilometer count
  Real fuel; // amount of fuel
  Integer passengers; // number of passengers
end car;
```

Listing 1. ABMLib agent partial record code.

Since ABMLib is based on the proposed message passing communication mechanism and that is not yet part of the Modelica specification, ABMLib models cannot be simulated with standard Modelica tools. The implementation proposed by the authors is described in Section 6. Briefly, the support for flattening ABMLib models has been included in the ModelicaCC compiler [18]. This implementation includes support for describing buffers and couple sentences. The process generates Modelica flat code that can be simulated using a standard Modelica tool (OpenModelica in our case). The generated code includes calls to external functions in C that contain the actual implementation of the message passing communication. Some manual modifications have to be performed to the model in order to correctly generate the flat code. As a result, ABMLib should be considered as a proof-of-concept library for supporting ABM in Modelica.

An ABM described using ABMLib is composed of *agent types*, at least one, that defines the characteristics and behavior of a kind of agent that populates the model (i.e., cars, humans, sheep, wolves, ants, etc.), and one *environment* model that describes the characteristics and behavior of the world where agents live. Agent types may include functionality to define their behavior and the interaction between different agent types (e.g., humans and cars). All these components are detailed next.

4.1. Agent type

An agent type is a Modelica model that includes the characteristics of the agent and its behavior.

The record that describes the agent characteristics is used as data structure to create the messages that represent the agents flowing in the diagram. ABMLib includes a partial record, named `stdAgent`, that contains some common variables for all agent types: position in the environment, orientation, color, identification number, agent type name and the buffer where the agent is located before being moved to another destination. The `stdAgent` record can be extended, and thus its variables inherited, by other records to facilitate the description of the characteristics of other agents. The Modelica code for the `stdAgent` record and the declaration of a new agent, named `car`, is shown in Listing 1.

4.2. Agent behavior

The behavior of an agent type is described by means of a flowchart diagram. Agents are created in the diagram and flow through its components following the structure of its links and performing the actions defined by the components. All the inter-connected components of a flowchart diagram have to manage the same agent type, which is specified as a parameter of each component additionally to other parameters. In order to describe the flowchart diagram, ABMLib includes the components listed below. A graphical description of their interfaces is shown in Fig. 2. Ports with two stripes represent buffers for message communication and the rest are Boolean inputs and Real outputs.

- *Create*, represents the starting point for the agents in the diagram. Agents are created in batches of a given size.
- *Dispose*, represents the ending point for the agents. Agents that arrive to this component are removed from the model.
- *Assign*, represents an assignment to a variable of the agent or the model, in response to the arrival of an agent.
- *Decide*, represents a bifurcation in the flow of agents based on the value of a boolean condition. If multiple conditions need to be meet, multiple Decide components should be consecutively nested.
- *Count*, represents a point where basic statistical information of the model is computed based in the flow of agents.
- *Duplicate*, represents the creation of a duplicate of an agent. The duplicate agent is an exact copy of the original agent, but it has a different identification number.
- *Tick*, represents the point where the agents wait for a time unit to be elapsed (i.e., a tick of time).

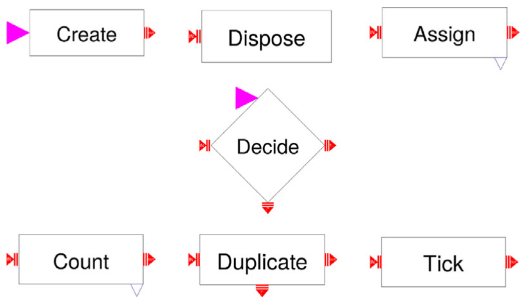


Fig. 2. Flowchart component models in ABMLib.

Note that some variables have been included in the components to facilitate the analysis of the simulations (e.g., number of disposed agents, number of agents in each decision branch, counters, etc.). This information can be used to observe the evolution of the model and perform a basic analysis of the simulation results. More detailed and complex information, such as statistical indicators, can be included in these or other custom made components to perform more complex analyses.

4.3. Agent interaction

Agent interaction happens when the actions performed by an agent depend on the state of other agents, or when they directly modify the state of other agents. These two kind of interactions can be described using the proposed extensions to the message passing communication mechanism: the `msgset` and `msgsubset` classes.

The `msgset` and `msgsubset` objects can be used to access the state of other agents, like any other model variable. The values of the state variables of other agents can be used to configure the parameters or conditions in the flowchart diagram of an agent type.

On the other hand, an additional flowchart component has to be included in order to influence the state of other agents. The *Send* component has been included in ABMLib for this purpose. This component can be used to extract an agent from its current buffer and send it to a new destination (i.e., another flowchart diagram) that represents the new actions to be performed by that agent. For example, when an agent *policeman* finds an agent *criminal* (e.g., both agents are located in the same coordinates in the environment) a Send component can be used to send the criminal to another flowchart diagram that represents the stay in jail. The parameters of the Send component are the agent to be sent.

Note that since the agent sent to the destination is extracted from its flowchart diagram, it will no longer be able to return to its normal behavior. In order to facilitate the return of the agent to the point in the original flowchart diagram before the movement, a variable named `origin` is included by default in the state of the agent. This variable stores a reference to the original buffer, and can be used with another Send component to return the agent back to its behavior.

4.4. Environment

The environment represents the space where the agents move and behave. It can be described in multiple ways, but in order to simplify this proposal it is described as a uniform two-dimensional grid of square cells, as a cellular automaton.

Cells are described using a set of state variables and a transition function, which is periodically evaluated to update the state of the cells. The authors have developed the CellularAutomataLib2 library, which facilitates the description and efficient simulation of cellular automata in Modelica [11].

The interaction between agents and their environment can be described using the interface models included in CellularAutomataLib2. The *ExtInputRegion* model can be used to modify the state of the cells of the environment. The *OutputRegion* model can be used to observe the state of the cells of the environment, and use it to configure the behavior of the agents.

CellularAutomataLib2 models automatically generate a graphical animation of the simulation. This animation can also serve to analyze the simulation results of the ABMs.

5. Interface with other Modelica models

Multiple free and commercial Modelica libraries that support modeling in multiple domains and with multiple formalisms are already available. The combination of these models with ABMLib models offers an extended functionality to describe more complex models. This combination is performed by including Modelica connectors in the ABMLib components. The values of the variables of these connectors can be observed and used to define the behavior of the agents, or vice-versa, the behavior of the agents can define the values of those variables in the connectors.

The following ABMLib components include connectors to interface with other Modelica models:

- Create, includes a Boolean input connector, named EXTIN, that can be used to control the creation of new agents using an external input. Every time the EXTIN port switches its value a new batch of agents is created.

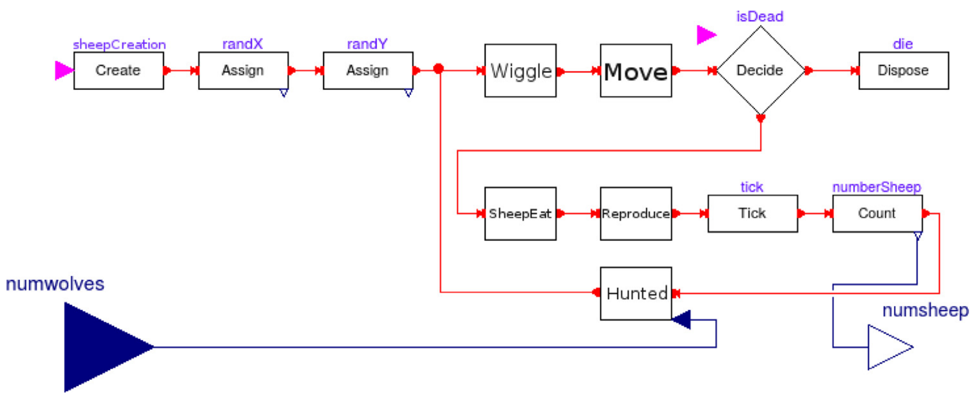


Fig. 3. Flowchart diagram of the Sheep model.

- Assign and Count, both components include an EXTOUT connector of Real type that is assigned with the value assigned to the variable in the Assign component or with the new value of the counter in the Count component. Thus, the EXTOUT connector resembles a discrete-time piecewise constant signal.
- Decide, includes a Boolean connector, named COND, that can be used to define the condition used to divide the flow of agents.

6. Application example: Hybrid Sheep-Wolves model

In order to demonstrate the modeling functionality of ABMLib, an application example is presented. It corresponds to a Lotka–Volterra model where sheep and wolves coexist in the same environment (cf. [15], where it is implemented using Netlogo). The comparison between the Netlogo and ABMLib models helps to validate the simulation results and evaluate the modeling functionality of the latter.

Since one of the main advantages of describing ABMs in Modelica is the possibility to combine equation-based models with ABMs, the presented model is described using a combined approach. Sheep are described as agents, while the wolves are described using the Lotka–Volterra equations for predator-prey models. Both parts of the model are described next.

6.1. Wolves

The Lotka–Volterra equations for predator-prey models are:

$$\dot{x} = \alpha x - \beta xy \tag{1}$$

$$\dot{y} = \delta xy - \gamma y \tag{2}$$

where, x represents the number of preys, y represents the number of predators, and α, β, δ and γ are the parameters that describe the interactions between both species.

The behavior of wolves, Eq. (2), can be directly coded in Modelica as:

```
der(wolves) = C * sheep * wolves - D * wolves;
```

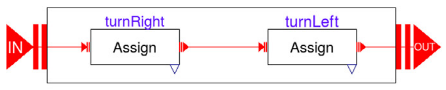
Note that the number of sheep, `sheep`, is an input to this model and needs to be computed by the agent-based Sheep model.

6.2. Sheep

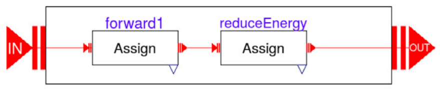
Sheep agents are described using a Modelica record, named `SheepAgent`, which is composed of three variables to represent the current energy of the sheep, the energy gained from eating grass and the cost of moving.

The behavior of the sheep is as follows (the corresponding flowchart diagram is shown in Fig. 3). Sheep are created and a random position for each sheep is assigned. After that, sheep behave in cycles defined using a tick component. During each tick, each living sheep performs the following actions: they wiggle and move around the environment searching for grass, while consuming energy. Sheep die and are removed from the model when they spent all their energy. Otherwise, they eat grass to gain energy and reproduce, which also consumes energy. A counter is used to count the number of living sheep in the model, which is the input required by the wolves model.

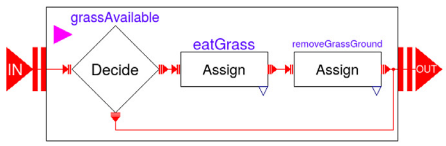
Before starting a new cycle, sheep can be hunt by wolves. Since wolves are not described as agents, sheep are hunted using a probabilistic approach based on the number of wolves computed by the equations (i.e., `wolves` variable in the Wolves model).



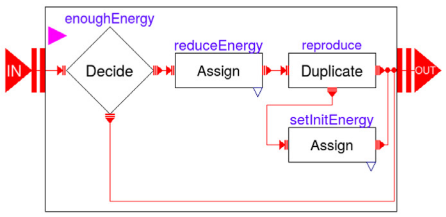
(a) Wiggle



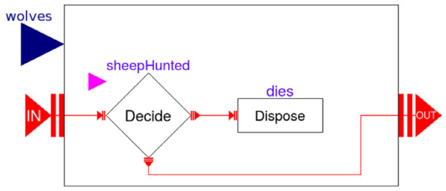
(b) Move



(c) SheepEat



(d) Reproduce



(e) Hunted

Fig. 4. Sheep model coupled components.

Note that the Wiggle, Move, SheepEat, Reproduce and Hunted are coupled components of the flowchart diagram. Their internal contents are graphically shown in Fig. 4.

6.3. Simulation

An implementation of the message passing communication mechanism has been included in the ModelicaCC compiler in order to simulate the model. Some additional features are required to flatten the ABMLib model and generate standard Modelica code, that is simulated using OpenModelica.

The procedure to flatten and simulate the Hybrid Sheep-Wolves model is as follows:

- A new class is included in the model to represent the buffers (i.e., `class buffer`). Input and output buffers inherit this new class. While the model is analyzed, the different types of buffers that appear in the model are registered.

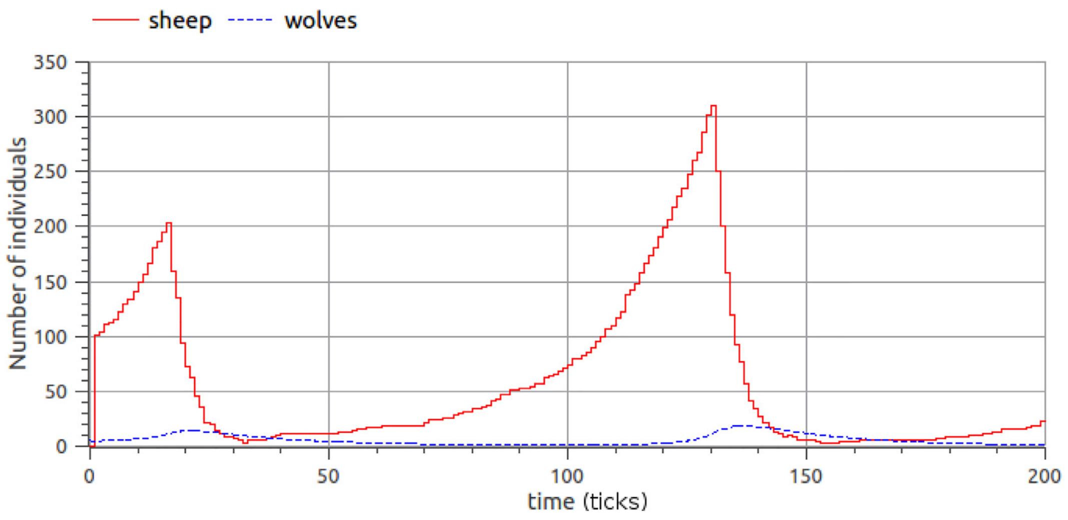


Fig. 5. Simulation of the Hybrid Sheep-Wolves ABMLib model.

- Expressions using the `size` variable of a buffer are replaced with a call to an external C function (e.g., `IN.size`, where `IN` is a buffer of type `B` is replaced by a call to the C function `B_size(IN)`).
- A read access to the first element of a buffer is replaced by a call to the external C function `peek` (e.g., `IN[1]` is replaced with `B_peek(IN)`).
- A read access to a variable in the first element of a buffer is similarly replaced by a function (e.g., `IN[1].a` is replaced with `B_peek_a(IN)`).
- Two functions, named `B_put` and `B_pop`, are defined to substitute the `put` and `pop` functions for the type of buffer `B`.
- The `couple` sentences are also replaced by external C function calls (e.g., `B_couple(OUT,IN)`).
- Buffers defined as interfaces of coupled models are removed and their two couple sentences (one outside the model and one inside) are replaced by only one. This operation has to be performed manually, since the tool does not currently support this simplification.
- All buffers are replaced by external C objects, with calls to their constructor and destructor functions.

After the model is flattened, it can be simulated using OpenModelica. An example of simulation run is shown in Fig. 5. Note the alternating oscillatory behavior between the number of sheep and wolves.

7. Conclusions

A new free Modelica library, named ABMLib, has been designed and developed to facilitate the description of agent-based models (ABMs) and their combination with other Modelica models. The design of the library is based in the analyses of the requirements to describe ABMs in Modelica and the current functionality of the language. Agents are described as messages moving through a flowchart diagram, whose components represent the actions that define the behavior of the agents. This allows to have a variable number of agents in the model during the simulation run. The communication between components of the flowchart diagram is performed using the message passing mechanism proposed by the authors. Additional extensions to this mechanism are also proposed to describe the interactions among agents. The environment where the agents live is represented using a cellular automaton, and described using the CellularAutomataLib2 library. This allows an efficient simulation of large spatially dependent models, using two-dimensional lattice structures.

The main limitation of the library is that it is based in language extensions, and thus it is not supported by current Modelica tools. However, a prototype implementation using the ModelicaCC compiler is presented to demonstrate the functionality of the library. The message passing mechanism has been implemented in ModelicaCC as calls to external functions in C. ABMLib models can be flattened and the resulting code can be simulated using OpenModelica. Some manual manipulations of the model have to be performed during the flattening process. The simulation algorithm for ABMs has to be studied and improved.

In the future, a full implementation of the message passing mechanism will be developed. The support of ABMLib in other Modelica tools has to be analyzed in order to facilitate the use of the library to different users. The functionality to describe agents will be revised taking into account already used concepts such as the BDI (Belief-Desire-Intention) and the process calculus. A better integration between CellularAutomataLib2 and ABMLib has to be performed, to facilitate the description of models with different types of environments. Also, the integration between ABMLib and other agent-based tools will be considered, specially those that support the FIPA standards. Additional flowchart components will be included in ABMLib, specially to automatically generate statistical indicators to analyze the simulation results. Also, a better graphical animation will be developed to include more information about the model.

Acknowledgments

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

References

[1] D.J. Barnes, D. Chu, Guide to Simulation and Modeling for Biosciences, (second ed), Springer, London, 2015.
[2] K.J. Åström, H. Elmqvist, S.E. Mattsson, Evolution of continuous-time modeling and simulation, in: Proceedings of the 12th European Simulation Multiconference (ESM'98), Manchester, UK, 1998, pp. 9–18.
[3] Modelica Association, Modelica - An unified object-oriented language for physical systems modeling. Language spec. v. 3.4, 2017, [online; accessed 14-Dec-2017].
[4] F.E. Cellier, E. Kofman, Continuous System Simulation, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[5] T. Bosse, A. Sharpanskykh, J. Treur, Integrating agent models and dynamical systems, in: Proceedings of the 5th International Conference on Declarative Agent Languages and Technologies V, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 50–68.
[6] T. Bosse, C.M. Jonker, L.V.D. Meij, J. Treur, Leadsto: A language and environment for analysis of dynamics by simulation, in: Proc. of the Third German Conference on Multi-Agent System Technologies, MATES'05. Lecture Notes in Artificial Intelligence, Springer Verlag, 2005, pp. 165–178.
[7] A. Djanatliev, R. German, P. Kolominsky-Rabas, B.M. Hofmann, Hybrid simulation with loosely coupled system dynamics and agent-based models for prospective health technology assessments, in: Proceedings of the 2012 Winter Simulation Conference (WSC), 2012, pp. 1–12.
[8] L. Caudill, B. Lawson, A hybrid agent-based and differential equations model for simulating antibiotic resistance in a hospital ward, in: Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World, in: WSC '13, IEEE Press, Piscataway, NJ, USA, 2013, pp. 1419–1430.
[9] A. Constantin, A. Löwen, F. Ponci, K. Huchtemann, D. Müller, Dymola-JADE co-simulation for agent-based control in office spaces, in: Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, 2017, pp. 345–351.
[10] B.P. Zeigler, T.G. Kim, H. Prähofer, Theory of Modeling and Simulation, Academic Press, Inc., Orlando, FL, USA, 2000.

[11] V. Sanz, A. Urquia, A. Leva, CellularAutomataLib2: Improving the support for cellular automata modeling in Modelica, Math. Comput. Model. Dyn. Syst. 22 (3) (2016) 244–264.
[12] V. Sanz, A. Urquia, Modelica extensions for supporting message passing communication, in: Proceedings of the 7th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Milan, Italy, 2016, pp. 21–28.
[13] B.C. Pierce, Programming in the pi-calculus: A tutorial introduction to Pict, 1997. Available electronically.
[14] E.A. Lee, The problem with threads, Computer 39 (5) (2006) 33–42.
[15] U. Wilensky, W. Rand, An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with Netlogo, MIT Press, Cambridge, MA, USA, 2015.
[16] M.P.I. Forum, MPI: A message-passing interface standard, 2015, [online; accessed 14-Dec-2017].
[17] V. Sanz, A. Urquia, S. Dormido, Parallel DEVS and process-oriented modeling in Modelica, in: Proc. of the 7th Intl. Modelica Conf., Como, Italy, 2009, pp. 96–107.
[18] F. Bergero, M. Botta, E. Campostrini, E. Kofman, Efficient compilation of large scale dynamical systems, in: Proceedings of the 11th International Modelica Conference, Versailles, France, 2015, pp. 449–458.