Designing co-simulation with multi-agent tools: a case study with NetLogo

Thomas Paris, Laurent Ciarletta, Vincent Chevrier

Designing co-simulation with multi-agent tools: a case study with NetLogo

Thomas Paris*, Laurent Ciarletta*, and Vincent Chevrier*

* Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France
firstname.name@loria.fr

December, 2017

Abstract

Multi-agent approach has demonstrated its benefits for complex system modeling and simulation. This article focuses on how to represent and simulate a system as a set of several interacting simulators, with a focus on the case of multi-agent simulators. This raises a major challenge: multi-agent simulators are not conceived (in general) to be used with other simulators.

This article presents a preliminary study about the rigorous integration of multi-agent simulators into a co-simulation platform. The work is grounded on the NetLogo simulator and the co-simulation platform MECSYCO.

Keywords: Complex system, Multi-agent system, Co-simulation, MECSYCO, NetLogo

1 Introduction

The modeling and simulation (M&S) of complex systems is one of the key challenges in research. One of the difficulties is to combine several perspectives of the same system [Seck and Honig, 2012] into a coherent one (multi-modeling). It needs to manage the system with several levels (micro, macro), different scales (time, space, ...), etc. Handling such heterogeneities calls for the development of new approaches and tools.

One of the most promising approaches to face these challenges is co-simulation [Gomes et al., 2017]. It consists in making different simulators interact into a simulation by ensuring the synchronization and the data exchanges between them. It enables the reuse of existing simulators used in specific domains. However, this implies to be able to manage the heterogeneities of the simulators both at software level (how to control simulators execution to make them interact?) and at formal one (how to make compatible the different dynamics?).

In parallel, multi-agent approach is convenient to represent and simulate systems composed of numerous interacting entities (which is a definition of complex systems [Ramat, 2007]). It makes possible to represent both the individual and collective levels [Michel et al., 2009]. Then multi-agent approach is a relevant choice to model and simulate complex systems.

The general question we address in this paper is *How can we represent and simulate a complex system with different multi-agent systems*, each representing a complementary perspective of the whole. We adopt a co-simulation approach to answer it. The issue we are now facing becomes how to make the multi-agent simulators interact to exchange information and synchronize their execution. We limit our scope in this article to spatial coupling of multi-agent systems: one agent is present in one simulator at a time, agents in different simulators can not interact. Interactions are restricted to events that pass from one simulator to another. Our goal is to rigorously integrate multi-agent simulators in an hybrid co-simulation which uses both continuous and discrete simulators. We do not consider ad-hoc solutions (potentially source of errors), nor the rewriting of models into one single simulator (source of errors, waste of time,...).

The problem we focus on can be solved by answering two questions: i) how to manage the time and the synchronization of the multi-agent simulator with the rest of the simulation ?; and ii) how to manage information exchanges between the simulator and the other simulators of the co-simulation ?

We demonstrated that these questions can be answered in the MECSYCO middleware [Camus, 2015] by using the DEVS formalism as a formal basis for integration. This article details how we answer these questions and build a DEVS wrapper in the case of the NetLogo multi-agent simulator.

The remaining of the article is structured as follows: The section 2 presents related works; next section 3 introduces concepts used to build our proposal. The section 4 presents the principles of our proposal which is detailed in 5. The section 6 presents different use-cases that illustrate the possibilities of the proposal. Section 7 discusses the approach and section 8 concludes the article.

2 Related works

Several works dealt with the simulation of a multi-agent system as the integration of different subsystems.

A first question is to position that integration with respect to the modeling and simulation process. We use the structuring of [Galán et al., 2009] which distinguishes four steps as represented in figure 1. From this point of view, the integration of subsystems is possible at the interface of these 4 levels.

In the first case, a conceptual formulation describes the integration by proposing means of exchanges of information between components and of components synchronization, as for example, patterns [Gangat et al., 2012], or models [Morvan et al., 2013, Maudet et al., 2013]. These works limit the integration of the different components of the multi-agent system into the same conceptual framework and the same tool.

The second case considers a formalism as a pivot for a rigorous integration. It is the case of the VLE (Virtual Laboratory Environment)[Quesnel et al., 2009], from which we retain its ideas under the wrapping perspective.

The third case envisages the integration as a software interoperability problem, the conceptual and formal issues are solved from an ad-hoc manner.

Alternatively, different multi-agent simulators are able to simulate a system as a set of different (logical or physical) environments (such as Madkit[1], or Repast[North et al., 2013]), but are not open to include environments coming from other tools. An effort for

[1] http://www.madkit.net/madkit/

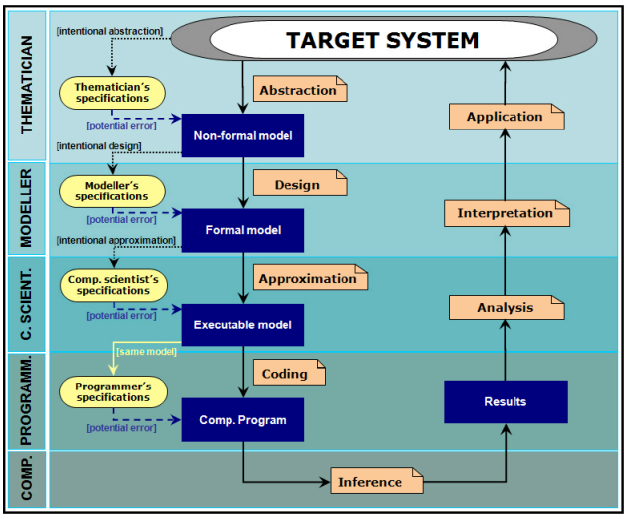

Figure 1: The different steps of the modeling process. Source [Galán et al., 2009]

a more standard integration has been done in [Behrens et al., 2011] by considering an interface for agent/environment interaction. As far as we know, the issue of integrating a multi-agent simulator into a co-simulation is not tackled, nor the rigorous management of synchronization. Existing works we know (except [Quesnel et al., 2009]) propose different multi-agent concepts to define the integration of components into a simulation but are not open to a wider scope.

3 Prerequisites

3.1 DEVS

DEVS (Discrete EVent System specification) is an event-based formalism proposed by Bernard P. Zeigler in 1970 [Zeigler et al., 2000]. One interesting properties of DEVS is its ability to integrate different formalisms. Thanks to its universality property, DEVS has a pivot place for the integration of formalisms [Vangheluwe, 2000].

As summarized by [Quesnel et al., 2005], the integration of a formalism in DEVS can be performed either by a mapping or a wrapping strategy. While the former consists in establishing the equivalence between the formalisms, the latter implies bridging the gap between the abstract simulators.

The advantage of the wrapping strategy is to enable the reuse of preexisting models already implemented in some simulation software. This is the choice we make in this article.

3.1.1 Description of models

DEVS distinguishes atomic from coupled models. A DEVS atomic model describes the behavior of the system and corresponds to this structure: $M = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$ where:

$X = \{(p, v) | p \in InPorts, v \in X_p\}$ is the set of input ports and values. These ports can receive external input events,

$Y = \{(p, v) | p \in OutPorts, v \in Y_p\}$ is the set of output ports and values. These ports can send external output events,

S is the set of the model states,

$\delta_{ext} : Q \times X \to S$ is the external transition function (describing how the model reacts to input events) where

$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the total state of the model,

e is the elapsed time since the last transition,

$\delta_{int} : S \to S$ is the internal transition function describing the internal dynamic of the model -i.e. the function processes an internal event which changes the model state,

$\lambda : S \to Y$ is the output function describing the output events of the model according to its current state,

$ta : S \to \mathbb{R}^+_{0,\infty}$ is the time advance function describing the time during which the model will stay in the same current state (in the absence of input event). The function is used to get the date of the next internal event.

A coupled model describes the structure of the system, that is how atomic models are connected together to describe a system. As we do not refer directly to this concept for our proposal we shall not detail it here.

3.1.2 Wrapping and simulation

DEVS formalism proposes different abstract simulation algorithms. It imposes five functions (detailed below) in order to perform a simulation with an atomic model. Defining a wrapper for a multi-agent system corresponds to define an interface with the simulator that implements these five functions. Once implemented, the simulator can be used as an atomic model and be rigorously integrated in a DEVS co-simulation.

3.2 NetLogo

NetLogo [Wilensky, 1999] is an environment for modeling and simulating multi-agent systems. NetLogo can be controlled through an API that eases its integration in a wider project. A NetLogo model is composed of a graphical interface which makes possible for a user to interact with the simulation and view its evolution; a script (written in the NetLogo language) that describes the behavior of the agents (called *turtles*), the dynamics of the environment and, more generally, which actions to perform during simulation. NetLogo language permits the definition of methods; the *command* ones that act on the world and the *report* ones that collect data; and a model documentation that explains the model, its functioning and how to experience it.

Conventionally, the simulation parameters are initialized by the *setup* method and the simulation is ran with successive calls to the *go* command that makes the simulation progress step by step. The graphical interface proposes buttons associated with these commands, and sliders to select parameters values.

3.3 MECSYCO

MECSYCO [Camus et al., 2015] is a DEVS wrapping platform[2] that takes advantage of the DEVS universality for enabling multi-paradigm co-simulation of complex systems. It is currently used for the M&S of smart electrical grids in the context of a partnership between LORIA/Inria[3] and EDF R&D (leading French electric utility company) [Vaubourg et al., 2015].

MECSYCO is based on the AA4MM (Agents & Artifacts for Multi-Modeling) paradigm [Siebert et al., 2010] (from an original idea of Bonneaud [Bonneaud, 2008]) that sees an heterogeneous co-simulation as a multi-agent system. Within this scope, each couple model/simulator corresponds to an agent, and the data exchanges between the simulators correspond to the interactions between the agents[4]. Originality with regard to other multi-agent multi-model approaches is to consider the interactions in an indirect way thanks to the concept of passive computational entities called artifacts [Ricci et al., 2007]. By following this multi-agent paradigm from the concepts to their implementation, MECSYCO ensures a modular, extensible (i.e. features such as an observation system can be easily added), decentralized and distributable parallel co-simulation. MECSYCO implements the AA4MM concepts according to DEVS simulation protocol for coordinating the executions of the simulators and managing interactions between models.

So far, we successfully define DEVS wrappers for discrete and continuous modeling tools like the telecommunication network simulators NS-3 and OMNeT++ [Vaubourg et al., 2016], the FMI standard [Blochwitz et al., 2012], or application-specific wrapper for the NetLogo simulator [Camus et al., 2015].

4 Proposal

Til now, the integration of NetLogo in MECSYCO obliges to specify a new wrapper each time a new NetLogo model is used. This drawback comes from the absence of declarative representation of DEVS concepts; i.e., the declaration of inputs, outputs and parameters (model specific elements) were made directly in the code of the wrapper, making it model specific.

Specifying a DEVS wrapper for MECSYCO implies to create an interface between the simulator and the five functions of the DEVS simulation protocol:

- `init` sets the parameters and the initial state of the model,
- `processExternalEvent` makes the simulator process its external input event(s) coming from other simulators, it is dependent of the input ports of the model and the kinds of information associated with,
- `processInternalEvent` makes the simulator process its internal event(s) at a given time (makes the simulator progress according time),
- `getOutputEvent` returns external output event(s) to be sent to other simulators, it is dependent of the output ports of the model

[2] MECSYCO is available on www.mecsyco.com under AGPL license.

[3] French IT research institute.

[4] Please note that multi-agent systems appear at two levels in this article: as a middleware architecture for co-simulation, and as simulation models to be integrated in a co-simulation.

- `getNextInternalEventTime` returns the time of the earliest scheduled internal event. The simulator scheduling policy must have temporal meaning in order to determine that value.

These five methods handle the time management, the synchronization and the date exchanges between a simulator and the remaining of the co-simulation. To design a generic wrapper, these functions must be independent of the simulated models. In particular, this implies for each model m to specify the sets of input and output ports (X and Y respectively). These information are rarely present in multi-agent system (mostly because there are not conceived to be connected to other models and not thought as a port-based architecture). We propose to define explicitly these sets and the corresponding events in a documentation associated with the model (in the same sense as the XML description of the FMI standard). Similarly, we have to express what has to be done when input events are received, how to get the data corresponding to output events and how to set initial parameters.

As the multi-agent simulators adopt various strategies of implementation and meta-model, we don't target a unified way to design wrapper, but rather try to propose a generic wrapper for NetLogo for which we take advantage of the possibility to send commands to the simulator thanks to an interpreter provided by the API.

It must be underlined that the *processInternalEvent* function makes evolve the simulation state and cannot be broken into different subfunctions. As a consequence, we have no mean to make agents from different simulator interact together in the same simulation step (or we stop respecting DEVS simulation protocol).

5 The DEVS wrapper for NetLogo

This part details the wrapping of the NetLogo tool through the definition of the wrapper documentation. Figure 2 summarizes the principles behind it.

5.1 Documentation information

Until now, the NetLogo wrappers of MECSYCO embedded in their Java code the information related to the input and output ports, as well as the statements to process when executing each of the five DEVS methods. We propose to provide them separately in the wrapper documentation. That means providing the name of input and output ports, and what to process for each method of the DEVS protocol.

init(): We suppose the *setup* method of NetLogo can be used as the *init* method[5]. We add the concept of parameters in the documentation in order to provide values to set instead of the default ones used by setup (generally defined by sliders in the interface).

getNextInternalEventTime(): We suppose a constant time step simulation strategy in which each tick has no special temporal meaning. It is the responsibility of the modeler to propose a meaning of a tick in term of co-simulation time (e.g. one tick represent 0.1 unit of time simulation).

processInternalEvent(): We suppose[6] the *go* method corresponds to the statements to process at each tick and can correspond to the processInternalEvent one.

[5]In other cases, either the documentation provides the command to call, either it provides the code to execute.

[6]In other cases, it has to be specified the same as for setup.

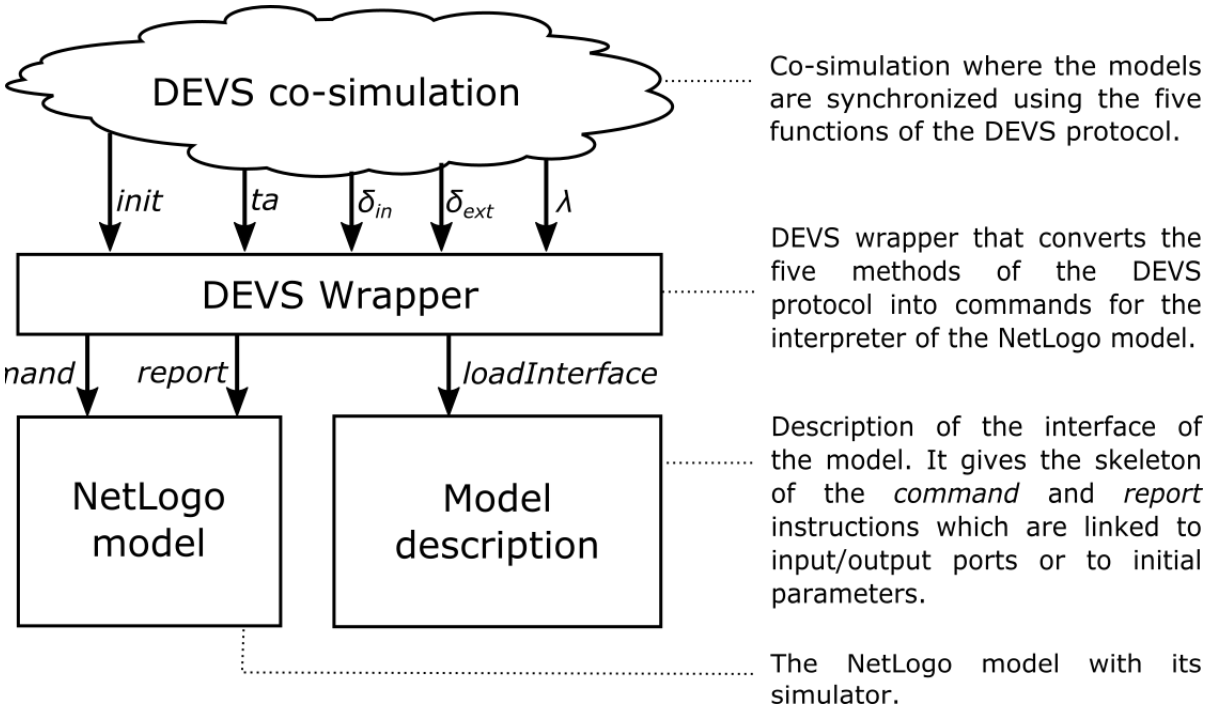

Figure 2: Principle of the NetLogo wrapper.

processExternalEvent(): The concept of input ports doesn't exist in NetLogo and must be defined for each model. Additionally, it has to be precised how to process each incoming event on each port.

getOutputEvent(): Again, the concept of output ports doesn't exist in NetLogo and must be defined for each model. Additionally, it has to be precised how to process each external event on each port.

5.2 Management of the simulation time

As in [Quesnel et al., 2005], we choose to let the modeler define the meaning of a tick as a constant duration t (as a consequence the ta function will constantly return $currentTime + t$. This is a simple solution. More complex ones are possible since NetLogo authorizes the modeler to define its own time progression. Wrapping can easily be extended to make a call to the ticks function and return the appropriate value.

5.3 Management of the inputs, outputs and parameters

As said previously, NetLogo architecture does not have the concept of ports associated with events but proposes an API through which the interaction with the model is possible via an interpreter. It consists in providing a string that corresponds to a NetLogo *command* to execute in order to modify the model or to a NetLogo *report* to fetch data from the model.

We propose to define in the wrapper documentation the port names of the inputs and outputs in association with the NetLogo methods to execute.

In the case of input events, we distinguish three cases:

1) the port accepts events which do not depend of data from other simulators: the model has to run some commands (with no parameters) defined by the modeler ;

2) the port accepts events which contains only single data from other simulators: this data should be integrated in the simulation through a NetLogo command that modifies environment variables or turtles attributes ;

3) the port accepts events which contains a list of data from other simulators: the command has to be adapted to process these data.

In case of output events, two cases can be envisaged:

1) the event corresponds to one or several data (the value of one attribute, ...). In this case, one or several reports will be used to access these values. This kind of event has no impact on the NetLogo model

2) In the second case, the event has an impact on the model (for example, turtles are exiting the model). One or more commands should be used to define this impact (e.g., suppress the turtles from the model).

The δ_{ext} of the DEVS interface processes input events with NetLogo *command*, whilst the λ one makes use of *report* to collect data and to convert them into events (some commands can be used to maintain a coherent state of the model, e.g., suppress turtles).

The parameters specified in the documentation will be set by a single *command* to modify a value before the call to *setup*.

5.4 The wrapper

The basic principle of definition of the wrapper is to associate each function of the DEVS protocol to the corresponding NetLogo code. This is done by a java code using the NetLogo API facilities.

As the code to be executed is specified in the wrapper documentation, the wrapper becomes generic and can be used for any NetLogo models.

To summarize, as NetLogo proposes two functions associated to initialization (*init*) and to the simulation of one step (*go*), we reuse them. When one step of simulation is executed, the time progress of a constant value. As the simulation of one step is atomic (it can not be broken and we don't have something equivalent to the elapsed time function), processing of incoming events is undertaken at the next time step of the simulator by modifying the model state, the next invocation of "go" will perform the reaction of the model.

6 Proofs of concepts

6.1 Experiment goal

The goal of the following experiments is to illustrate what can be done with such an approach, notability by showing how a NetLogo component can be used in a co-simulation with MECSYCO.

Two experiments are detailed. The first shows how we can modify the state of the simulation by modifying variables that impact the agents behavior (as the GUI could have done). The second illustrates the spatial coupling between several NetLogo models by a transfer of agents between them.

Before describing these proofs of concepts we make two remarks. First, we do not recall what a MECSYCO co-simulation is but just provide an intuitive definition through the example. We orient the reader who wishes more details to the MECSYCO website where several tutorials explain the main concepts used and illustrate co-simulation. Second, we

```
model WolfSheepPredationExample version "1.0"
path "My Models/NetLogo/Wolf Sheep Predation.nlogo"
interface
    parameters // model parameters
        grass : true command "set grass? %s" // always true
        grass_regrowth_time : 10 command "set grass-regrowth-time %s" //small value in order to see effect of e
        // the following just put the same values as the GUI. Only for illlustration purposes.
        initial_number_sheep : 100 command "set initial-number-sheep %s"
        initial_number_wolves : 50 command "set initial-number-wolves %s"

    inputs
        //Inputs with the commands to process accordingly
        grass_regrowth : Integer initOption parameter command "set grass-regrowth-time %s"
        wolf_hunt : Integer initOption no command "ask n-of %s wolves [die]"
        sheep_coming : Integer initOption no command "create-sheep %s [set color white set size 1.5  set label-
    outputs
        //outputs (NB we use double instead of int because of some trouble with netlogo representation ... :-(
        nb_sheep : Double initOption no report Double "count turtles with [breed = sheep]"
        nb_wolves : Double initOption no report Double "count turtles with [breed = wolves]"
        nb_grass : Double initOption no report Double "grass"

information
    keywords
        "Predation"
        "NetLogo"
    description
        "This model represents wolf-sheep predation system example of NetLogo, it is an example of the integrat
simulator
    simulation variables
        //simulation variable
        stopTime: 1000. variability parameter // Time of the end of simulation
        discretization: 1. variability parameter //correspondence between Netlogo tick and the simulation time
```

Figure 3: DSL definition for example 1.

use a DSL (see figure 3) to describe the wrapper documentation. It is out of the article scope to detail the possibility of the DSL. We just provide the key elements necessary to the understanding.

6.2 Variation on prey-predator model

The NetLogo Wolf-Sheep-Predation model[7] describes how the populations of wolves and sheep interact and evolve according to time as in a prey-predator ecosystem. Several parameters can be changed to observe their impact and the populations dynamics. We do not modify the original model but extend it in order to illustrate the possibility to modify model parameters, to define input events that modify at runtime some features of the model. Namely we want to:

- set initial values of some parameters,
- provide some input events that modify environment features,
- collect periodically information about the population to draw graphics (externally to NetLogo).

These elements are detailed below. They are summarized in Table 1 and followed by their definition in the NetLogo DSL we defined in MECSYCO. Note that parameters of NetLogo commands are denoted by %s.

[7]Provided in the models library.

Table 1: Summary of wrapper documentation.

Parameters		
Name	**Value**	**Command**
grass	true	"set grass? %s"
grass_regrowth_time	10	"set grass-regrowth-time %s"
initial_number_sheep	100	"set initial-number-sheep %s"
initial_number_wolves	50	"set initial-number-wolves %s"
Input ports		
Name	**Command**	
grass_regrowth	"set grass-regrowth-time %s"	
sheep_coming	"create-sheep %s [set color white set size 1.5 set label-color blue - 2 set energy random (2 * sheep-gain-from-food) setxy min-pxcor max-pycor]"	
wolf_hunt	"ask n-of %s wolves [die]"	
Output ports		
Name	**Report statement**	
nb_sheep	"count turtles with [breed = sheep]"	
nb_wolves	"count turtles with [breed = wolves]"	
nb_grass	"grass"	

We use the following parameters: `grass`, the grass dynamics is active (contrarily to the default value of the GUI); `grass_regrowth_time`, the time needed for grass to regrow in *tick*); `initial_number_sheep`, the initial number of sheep; and, `initial_number_wolves`, the initial number of wolves.

We create the following input ports (each event contains the value to be applied for the modification): `grass_regrowth` implies a modification of the time of grass to regrow (We have defined a port name that is different of the name of the environment variable); `sheep_coming`, results in an increase of the sheep number; and `wolf_hunt` that results in a decrease of the wolves number. Each port corresponds to one event coming from separate simple models (each sends a single event at a specified time).

We define the following output ports that are connected to a graphic drawer in order to display graphics: `nb_sheep`,the current number of sheep; `nb_wolves`, the current number of wolves; and `nb_grass`, the grass quantity.

Figure 4 shows the results obtained with that configuration. We can observe the impact of the different events (arrivals of 100 sheep at t=100; death of 25 wolves at t=175, increase of the time needed for grass to regrow at t =400).

This proof of concept shows that we are able to provide initial values (e.g. the grass is active), to modify some characteristics of the model (remove wolves, add sheep or modify environment features) by input events (coming from other models) and to collect information from the simulation (here the numbers of wolves and sheep, and the grass quantity) through output events that will be used in other model (here a graphical drawer).

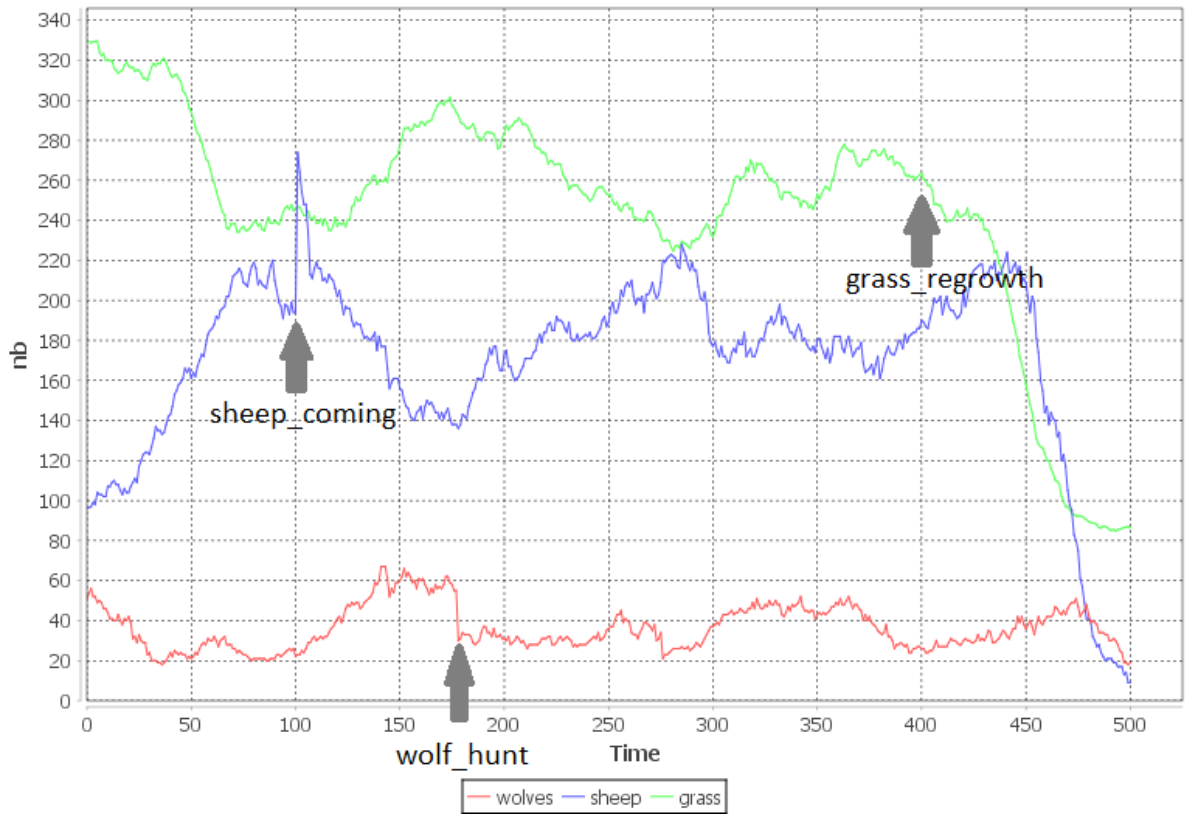

Figure 4: Impact of different events on the prey-predator model.

6.3 Spatial coupling

In this experiment, we show the possibility to transfer turtles from one NetLogo model to another. We connect three models together as follows: a prey-predator model "sends" sheep to a pedestrian model (a model in which turtles travel from left to right as pedestrians in a corridor do) which, when turtles arrive at the right extremity, sends them to another prey-predator model.

The first model is the same as the previous experiment, we add a new output port `sheep_escaping` that is associated to the sheep present at the right side of the model. Data are collected as a list of sheep features (ordinate and energy).

The second model is initially empty and sheep, coming from the `sheep_escaping` port, arrive through the input port (called `left_in`). Sheep move from left to right. An output port (`right_out`) is associated to the sheep present at the right side.

The third model is again a prey-predator model on which we add an input port `sheep_loop_arrival` that accepts a list of sheep to be created on the left side of the environment with attributes whose values are specified in the event coming from he `sheep_escaping` port). This model is initialized with empty populations of wolves and sheep. Figure 5 illustrates the connections and shows a snapshot of the three NetLogo windows.

These connections are possible because the types of events are compatible between the ports we use here and the pedestrian model for coherence purpose does define an "energy" attribute.

One interesting thing to notice is that the pedestrian model definition we used here enables the connection of several instances, one following the other, by the reuse of the same

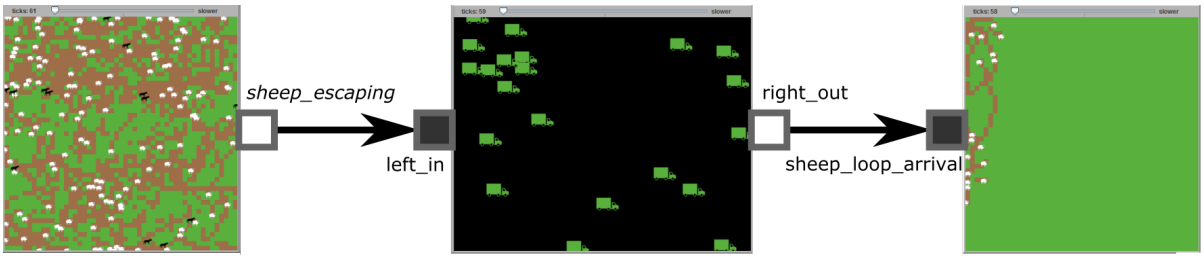

Figure 5: Connections between model for spatial coupling.

wrapper documentation. A model becomes a modular component for the co-simulation platform.

7 Discussion

The possibility of integrating a NetLogo model into a co-simulation as a component that can be added/removed or switched enables to define models of systems as the coupling of different sub-models.

One question not addressed in our approach is in what extend does an existing NetLogo model be adapted to be integrated in DEVS co-simulation. Currently, we do not propose any generic answer because of the variety of models available in NetLogo. However some directions can be proposed. Concerning the modification of the models, NetLogo GUI authorizes the user to modify parameters, to execute some actions at runtime. This is compatible with our approach: an input event can do the same. As we restricted our proposal to spatial coupling, we authorize agents to enter or exit the model. The exit of agents can be handled by providing some properties these agents have to respect and i) getting their features to be "exported" as a list and ii) removing them from the model. Agents entering the models can be represented by an event having a list of properties from which we create turtles.

We simplified the NetLogo integration by considering the use of *setup* and *go* commands instead of user defined ones. These choices can be revoked without putting into question the principles used.

As a system can be composed of many subsystems, the performance and the scalability of simulations can be put into question. Even if we did not focus on efficiency in the design of MECSYCO, a first answer is its architecture that can distribute the simulators execution among several machines. This enables to scale-up in terms of simulators number whilst keeping execution time.

This article focuses on the DEVS wrapping of NetLogo. We claim a declarative approach to bridge the gap between the DEVS simulation protocol and the model/simulator primitives. Readers may wonder on the generalization of this to other multi-agent system. From our experience, having a systematic approach is difficult because of the diversity of multi-agent platform architectures and multi-agent models structures, there is still no multi-agent standard to rely on.

8 Conclusion and perspectives

This article presented a preliminary work on the integration of a multi-agent simulator in a co-simulation. Our proposal (implemented on the NetLogo platform) is grounded on a wrapper documentation which precises i) the initial parameters, the input and output ports; and ii) the NetLogo codes that correspond to the implementation of the DEVS protocol functions in the model. This documentation can be written within a dedicated DSL and is used inside the generic NetLogo wrapper (a Java code in our case) of the MECSYCO platform.

We provide two proofs of concepts that showed the possibility to integrate NetLogo models in MECSYCO, and to make them interact together or with other already integrated simulators without any additional coding (except the one provided in the documentation).

This integration opens the possibility to reuse the wide variety of existing NetLogo models in a co-simulation (with or without other multi-agent simulators).

However, our proposal imposes that the model has been adapted in order to be used inside a co-simulation. A second limitation coming from the wrapping strategy imposes that the information exchanges make the time to progress; this forbids interactions between agents situated in different simulators.

As perspectives, we want to confront our proposal with more NetLogo models in order to gain a better understanding of how a NetLogo model has to be modified to be integrated and to validate conceptually the approach before confronting the principles of our proposal to other multi-agent simulators (with which we will be faced to the same conceptual issues and to new software integration problems).

References

[Behrens et al., 2011] Behrens, T. M., Hindriks, K. V., and Dix, J. (2011). Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295.

[Blochwitz et al., 2012] Blochwitz, T., Otter, M., Åkesson, J., et al. (2012). Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proc. 9th International Modelica Conference*, pages 173–184.

[Bonneaud, 2008] Bonneaud, S. (2008). *Des agents-modèles pour la modélisation et la simulation de systèmes complexes - Application à l'écosystémique des pêches.* PhD thesis.

[Camus, 2015] Camus, B. (2015). *Environnement Multi-agent pour la Multi-modélisation et Simulation des Systèmes Complexes.* PhD thesis, Université de Lorraine.

[Camus et al., 2015] Camus, B., Bourjot, C., and Chevrier, V. (2015). Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP). In *Proc. of TMS/DEVS 15*, pages 85–90. SCS.

[Galán et al., 2009] Galán, J. M., Izquierdo, L. R., Izquierdo, S. S., Santos, J. I., del Olmo, R., López-Paredes, A., and Edmonds, B. (2009). Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 12(1):1.

[Gangat et al., 2012] Gangat, Y., Payet, D., and Courdier, R. (2012). *Methodology for a New Agent Architecture Based on the MVC Pattern*, pages 230–239. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Gomes et al., 2017] Gomes, C., Thule, C., Broman, D., Gorm Larsen, P., and Vangheluwe, H. (2017). Cosimulation: State of the art. *International Mediterranean Modeling Multiconference*.

[Maudet et al., 2013] Maudet, A., Touya, G., Duchêne, C., and Picault, S. (2013). Improving multi-level interactions modelling in a multi-agent generalisation model: first thoughts. In *Proceedings of 16th ICA Workshop on Generalisation and Multiple Representation, Dresden, Germany*.

[Michel et al., 2009] Michel, F., Ferber, J., Drogoul, A., et al. (2009). Multi-agent systems and simulation: a survey from the agents community's perspective. In Uhrmacher, A. and Weyns, D., editors, *Multi-Agent Systems: Simulation and Applications, Computational Analysis, Synthesis, and Design of Dynamic Systems*, pages 3–52. CRC Press - Taylor and Francis.

[Morvan et al., 2013] Morvan, G., Veremme, A., and Dupont, D. (2013). IRM4MLS: the influence reaction model for multi-level simulation. *ArXiv e-prints*.

[North et al., 2013] North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1):3.

[Quesnel et al., 2005] Quesnel, G., Duboz, R., and Ramat, E. (2005). Wrapping into DEVS Simulator: A Study Case. *International Mediterranean Modeling Multiconference*, pages pp. 374–382.

[Quesnel et al., 2009] Quesnel, G., Duboz, R., and Ramat, E. (2009). The virtual laboratory environment - an operational framework for multi-modelling, simulation and analysis of complex systems. *Simulation Modelling Practice and Theory,*, (17):641–653.

[Ramat, 2007] Ramat, E. (2007). Introduction to discrete event modelling and simulation. *Agent-based Modelling and Simulation in the Social and Human Sciences, Lavoisier, Oxford, The Bardwell Press, Phan D., Amblard F. Eds.*

[Ricci et al., 2007] Ricci, A., Viroli, M., and Omicini, A. (2007). Give agents their artifacts: the A&A approach for engineering working environments in MAS. In *AAMAS '07*. ACM.

[Seck and Honig, 2012] Seck, M. D. and Honig, H. J. (2012). Multi-perspective modelling of complex phenomena. *Comput. Math. Organ. Theory*, 18(1):128–144.

[Siebert et al., 2010] Siebert, J., Ciarletta, L., and Chevrier, V. (2010). Agents and artefacts for multiple models co-evolution: building complex system simulation as a set of interacting models. In *Proc. of AAMAS '10*. AAMAS/ACM.

[Vangheluwe, 2000] Vangheluwe, H. L. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*, pages 129–134. IEEE.

[Vaubourg et al., 2016] Vaubourg, J., Chevrier, V., Ciarletta, L., and Camus, B. (2016). Co-simulation of ip network models in the cyber-physical systems context, using a devs-based platform. In SCS/ACM, editor, *Communications and Networking Simulation Symposium (CNS'16)*.

[Vaubourg et al., 2015] Vaubourg, J., Presse, Y., Camus, B., et al. (2015). Multi-agent multi-model simulation of smart grids in the MS4SG project. In *Proc. PAAMS 15*, pages 240–251. Springer.

[Wilensky, 1999] Wilensky, U. (1999). Netlogo (and netlogo user manual). *Center for connected learning and computer-based modeling, Northwestern University. http://ccl.northwestern.edu/netlogo*.

[Zeigler et al., 2000] Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation : integration discrete event and continuous complex dynamic systems*. Academic press.