Agents and artefacts for multiple models co-evolution

Building complex system simulation as a set of interacting models

Julien Siebert
INRIA, Centre Nancy Grand Est
julien.siebert@loria.fr

Laurent Ciarletta
Ecole National Supérieure des Mines de Nancy
laurent.ciarletta@loria.fr

Vincent Chevrier
Nancy Universtity (UHP)
vincent.chevrier@loria.fr

LORIA - Campus Scientifique - BP 239 - 54506 Vandoeuvre-lès-Nancy, France

ABSTRACT

Complex systems simulations generally involve the interaction of different scientific fields. Human economies, ecosystems or dynamic computer networks such as P2P are good examples. Since models and simulators already exist in those fields, designing the simulation as a society of interacting and co-evolving models appears attractive. Beyond the technical issues to make different simulators cooperate, the challenges are to make the co-evolution design and implementation easier for the scientist that rarely know intricate modelling and simulation tools, and to facilitate the collaboration of different experts. Agents and artefacts (A&A) paradigm simplifies the design and the implementation of a society of interacting and co-evolving models. That is, the addition, the removal or the interchange of models require less effort. Contrary to classical approaches, we have built a decentralized co-evolution architecture based upon A&A and a data-driven coordination model. In this article, beyond the architecture presentation, we focus on the benefit provided by A&A used for multiple models co-evolution.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modelling**]: Model Development

General Terms

Design

Keywords

Simulation techniques, tools and environments, Software engineering, Complex systems

1. INTRODUCTION

Modelling and Simulation (M&S) techniques and software engineering have always been in touch with each other. In [4] the authors declare that *[...] the real limits on the future adoption of simulation may rest on our ability to represent complex systems and to do it easily, which can be constructed as a matter of modelling style. [...] Thus the distinction in programming style is not what can and can't be done but what can be done easily.*

A complex system is composed of a set of interacting parts that as a whole exhibits properties that cannot be predicted from the simple sum of the individual parts properties. Human economies, social structures, climate or ecosystems are good examples of complex systems. Equation based modelling cannot represent interactions among components and their impact on the global system behaviour. Multiagent approach offers an interesting alternative [17].

Complex systems modelling also involves the interaction of different scientific domains or different abstraction levels. This is the case in biology, for example, where models from chemistry and biology (organ, tissues) may be needed [15] or in dynamic networks with models from sociology and computer networks [12]. This way, different specialists work on the same simulation. Each one brings its own models and simulators.

The challenge is then to allow those scientists to build a complex simulation from their own building blocks. Moreover, we should keep in mind that they are probably not familiar with the intricate modelling and simulation tools and theories. One way to facilitate the interaction of heterogeneous models is to build the simulation as a society of interacting models. Models should be seen as component we can weave together (as in component-based software engineering).

Some work has already been done in this way. The high level architecture (HLA) [5] is a standard that proposes to use a central infrastructure in order to make models interact. VLE [6] and JAMES II [3] are both modelling and simulation frameworks that allow to couple different models (issued from different formalisms). Agents and artefacts (A&A) paradigm [8] has been used in order to build a complex systems simulation as a society of interacting and co-evolving models [1].

In this paper we propose modelling engineering concepts to simplify the design of distributed and decentralized complex systems simulations. To do this, we use the perspective of A&A. However, contrary to approaches such as HLA or A&A used in [1], we do not rely on a global scheduler in order to synchronize the different interacting models. Instead, we propose a data-driven coordination model that allows a decentralized synchronization [13]. On the other hand, contrary to approaches such as VLE [6] and JAMES II [3] that imposes to build models directly in the modelling framework, we concentrate on reusing existing models and simulators and making them interact. Since our work fo-

Cite as: Agents and artefacts for multiple models co-evolution. Building complex system simulation as a set of interacting models., Julien Siebert, Laurent Ciarletta, Vincent Chevrier, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 509-516

cuses on model coupling level, innovations from the parallel and distributed computation domain may of course be used to implement our approach.

The outline is the following: in section 2, we present the challenges related to the coupling and the coordination of different simulators. In section 3, we propose a coordination model and an architecture based upon A&A. In section 4, we developed a proof of concepts. We use our concepts to make some existing Netlogo models interact. Finally, discuss the advantages and the limits of such an approach.

2. CHALLENGES AND RELATED WORKS

2.1 A modular approach

When we talk about complex systems, we usually think of decentralized, open and large scale systems. It is generally accepted that there is no central point of control. Actually, global properties emerge from the interaction of subsystems (or entities). Moreover, entities can enter and leave the system and their number is admitted to be large. A good example is P2P networks, in which there is no central point of control, people can connect and disconnect themselves *on the fly* and the size of these networks have reached several million users.

However, complexity occurs through the interaction of different abstraction levels. For example, prediction of P2P networks performances are difficult because it involves the interaction of socio-economic parameters (users willingness to share and to download data) and network parameters (bandwidth, download time). Models exist in both humanities and computer network sciences. Moreover, modelling a system as a set of interacting subsystems is simpler and requires a collaborating team of experts. Thus, it appears that, in order to predict the behaviour of a complex system such as P2P networks, it is necessary to couple heterogeneous models [12]. It may also be necessary to interchange a model by another one in order to compare simulation results. In the same way, a new instance of a model may be added in order to scale up the simulation.

We think that, in order to build a complex system simulation from heterogeneous models, specialists from the different concerned fields should be able to, firstly, reuse the existing models and simulators they develop with as little modifications as possible and, secondly, easily add, remove or interchange the models.

Existing modelling and simulation frameworks (such as JAMES II [3] or VLE [6]) imposes that every specialist involved in the simulation should know the framework and modify its own models in order to include them. Conversely, we think that models should be seen as component we can weave together (as in component-based software engineering). The challenge is then to modify as little as possible the existing models, to make them interact in such a way that it is transparent for the people involved in the simulation design.

In order to build concepts generic enough, we make several assumptions. Models are black boxes with input and output ports (as in [19], see figure 1) and there exist simulators functions that give us the current and the next simulation time (see section 2.3).

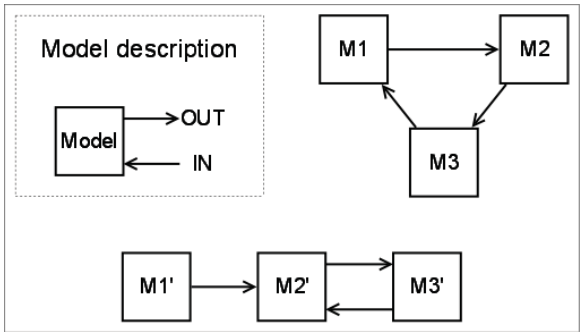

Figure 1: Examples of different kinds of models coupling

2.2 Exchanging data between simulators

When we couple different models the first thing to do is to represent the data flow between each model (as in figure 1). Since these models represent several dynamics at different scales and have been implemented independently, each model has its own representation of time and data. In the same way, each simulator proceeds its own execution. Coherence and compatibility issues appear when coupling different models and simulators. The next sections present the related challenges and issues.

2.2.1 Coherence

Scales or dimensions in which a piece of data is represented could be different from a model to another. Moreover the representation can be discrete or continuous. For example, a position $pos_1 = < x, y, z >$ (with x, y and z expressed in $meters$) in a first model can be represented only in two dimensions in a second one: $pos_2 = < x, y >$ (with x and y expressed in $kilometers$). A solution proposed in [1] is to define operations (projection, discretization, reduction) in order to achieve this coherence.

The same way, each model could have its own time representation. For synchronization purpose, we need to be sure, for example, that a time value $t_1 \in \mathbb{R}^+$ in a first model correspond to a time value $t_2 \in \mathbb{N}$ in a second one. We propose to express an operation that makes the correspondence between both time values.

2.2.2 Compatibility

Each simulator could implement a single piece of data in its own way (integer, float, *etc.*) or some simulators may not implement all aspects of a given model. These challenges are discussed in [9]. A solution is to add an entity (a program) between the simulators. Its role is to translate the data in order to respect the compatibility between simulation tools.

2.2.3 Synthesis

Coupling different models and simulators implies to deal with coherence and compatibilities issues that are neither models, nor simulators purpose. Since we try to modify as least as possible the existing models, a solution that appears is to design of a new entity in charge of these issues (see figure 2). This is what is proposed in [1], where an *artefact* [8] is define in order to deal with these issues.

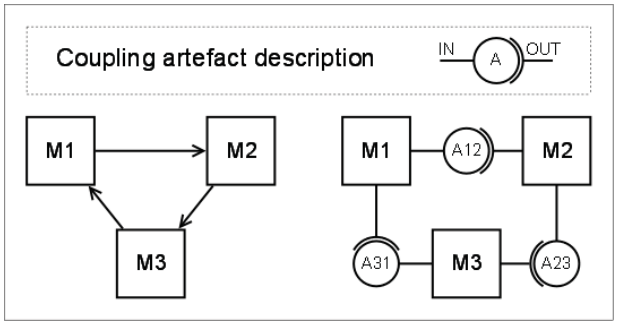

Figure 2: Using an artefact to deal with coupling issues. A *coupling-artefact* is needed for each link between two models

2.3 Coordination in co-evolution

The goal of time management in co-evolution is to ensure that simulation events (or steps) are executed in the correct order (local causality constraint [2]). Two main approaches have been proposed to coordinate interacting simulators: optimistic and conservative [2]. We assume that the existing simulators we reuse have been developed independently and where not thought for distributed simulations. So they do not have a roll-back capability (see optimistic approach): they cannot go back into the simulation process in order to take new input events into consideration. As a consequence, we focus on the conservative mode. In the latter, the coordination model has to determine when a simulation event (or step) is *safe* to process.

Definition 2.3.1. *For a model M_i, a simulation event (or step) associated with the current simulation time ct_i is said to be safe to process if all the input events received after this event execution are timestamped with a time value greater than ct_i.*

For example, in the figure 2, suppose the model M_1 is at simulation time $ct_1 = 8$. Its next event to process ne_1 is at time $nt_1 = 12$. This event ne_1 is said to be safe if no other event will be send to M_1 with a timestamp lesser than 12. In our example, M_1 has to know if M_3 will send events with timestamps $\in [8, 12[$. If not, ne_1 is safe and M_1 can process it. This way, simulators stay synchronized and the causality constraint is satisfied.

Conservative coordination can be done by using a central and global scheduler that synchronizes all the simulators as in [1, 9, 5]. In these solutions, models and simulators need substantial modifications in order to be controlled by the scheduler. This prevents to easily reuse the existing M&S tools. In section 3, we remove this central scheduler and we propose a novel decentralized coordination.

3. PROPOSITION

We present the concepts used in order to build complex simulation as a society of interacting models. We do not target on-line nor real time simulations, which directly interact with the reality.

3.1 Goals

We use the A&A paradigm [8] to facilitate the design and the implementation of a society of co-evolving models. Our approach is intended to make the design and the implementation of such simulations transparent. This way the specialists involved in the simulation do not care about coordination issues and only focus on the modelling aspects. The architecture we propose is modular and decentralized. It requires as little modifications as possible to integrate existing models and simulators. More important, these modifications are done once for all. Thus, to design a simulation of heterogeneous models becomes a building block game.

We propose to use objective coordination. That is, coordination does not rely on a single entity but is provided by the surrounding environment. This method is well known in the field of situated multiagent systems [16, 10] (stigmergy) or in parallel systems [11] (shared memory). This provides a way to loosely couple and to coordinate the interacting processes. In our case, the simulators interact through the set of data they exchange. Thus, their *surrounding environment* is composed of the *coupling-artefacts* between them and the set of exchanged data.

3.2 Validity interval and coordination

Our coordination model principle is the following, each model M_i holds a current simulation time value ct_i and knows the simulation time value for the next event (or next step) to be processed nt_i. That is, when a model M_i is executed, it produces data δ_i at time ct_i. These data δ_i will not change until the next time nt_i when the model M_i will be executed. As a result, we can say that δ_i are *valid* for the simulation time interval $\Gamma_i = [ct_i, nt_i[$.

A simulator can execute a model if the simulation event to process is safe (see definition 2.3.1). Then, the issue for the simulator is to know when an event is safe. It can be solve if the simulators exchange both the data and the corresponding validity interval: $< \delta_j; \Gamma_j >$. Indeed, a simulation event is safe to process if $ct_i \in \Gamma_j$ for all the input data ($i \neq j$).

Coordination is neither the simulator, nor the model purpose. It is the environment role. In our case, the *coupling-artefacts* besides being in charge of coupling issues (see section 2), are also in charge of coordination. That is, when data are transmitted through the *coupling-artefact*, the latter saves them (as a shared memory), then, when a model M_i needs input data for the time ct_i, it asks the *coupling-artefact* for *safe data* (i.e. data $< \delta_j; \Gamma_j >$ that fulfills the condition $ct_i \in \Gamma_j$ (with $i \neq j$)). This way, models can interact and coordinate themselves in a loosely and decentralized way.

We have developed a formal version [14] of this coordination model and we have proved that coordination occurs between models and that the system is alive and deadlock free with k models ($k \in \mathbf{N}$).

3.3 Architecture overview

Our architecture is intended to make the design and the implementation of a simulation of heterogeneous models transparent, modular and decentralized. In this section, we presents the concepts from A&A we use.

The *coupling-artefacts* are in charge of the coordination and the coupling aspects. The main contribution of our work is that data exchanges and coordination happen through the simulators environment (the *coupling-artefacts*) so the models execution can be decentralized. We previously define one *coupling-artefact* for each link between models (see figure 2). Data exchanged between two models flow through a *coupling*

Artefact	Functions	Role
Model-Artefact	*init()*	Initialize M_i
	run()	Run one simulation event, step or time interval
	getOutputData()	Return output δ_i from M_i
	setInputData()	Send input δ_j data to M_i
	getCurrentTime()	Return ct_i
	getNextTime()	Return nt_i
	stop()	Manage the end of the simulation (storing data, *etc.*)
Coupling-Artefact	*post()*	Post $< \delta_i, \Gamma_i >$
	read()	Return a *safe* data $< \delta_i, \Gamma_i >$

Table 1: Artefacts functions summary

artefact (see figure 3).

We use two other kinds of entities to deal with separation of concerns. To format data (i.e to associate δ_i and Γ_i), to send data through the *coupling-artefact* and to ask for safe data is neither the simulator, nor the model, nor the *coupling-artefact* purpose. So we define an entity which behaviour will be to manage the data flows for a given model: the *model-agent*. Its role is to get the data δ_i, the ct_i and nt_i values from the model M_i, to send the $< \delta_i; \Gamma_i >$ tuples to the *output coupling-artefacts*, to ask the safe data δ_j from the *input coupling-artefacts* and to send them to M_i. The *model-agent* behaviour is describe in figure 4.

In order for the *model-agent* to interact with the models, and in order to modify as little as possible the existing models M_i, we define an interface that allows the *model-agent* to achieve its purposes: the *model-artefact*. It allows the *model-agent* to initialize, execute the model M_i and to manage its input and output ports. The table 1 lists both the *coupling* and *model-artefacts* functions.

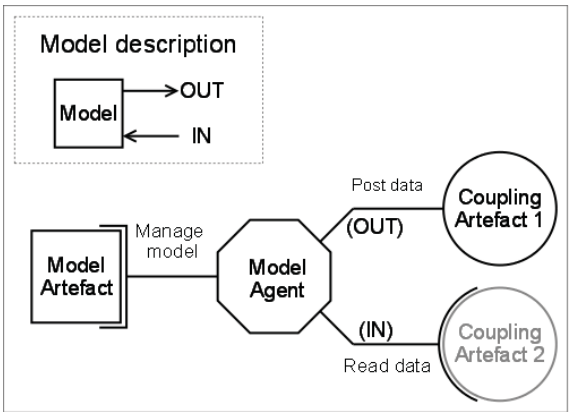

Figure 3: Architecture overview: from concepts to implementation

The whole architecture is described in the figure 3. It is modular, transparent and decentralized so the simulation can be seen as a set of distributed and reusable components. Models are in charge of modelling. The *model-agents* are in charge of executing the models. Finally, the *coupling-artefacts* are in charge of the coordination process. This way, we can easily add, remove or interchange models without being concerned with coupling and coordination issues (see section 4).

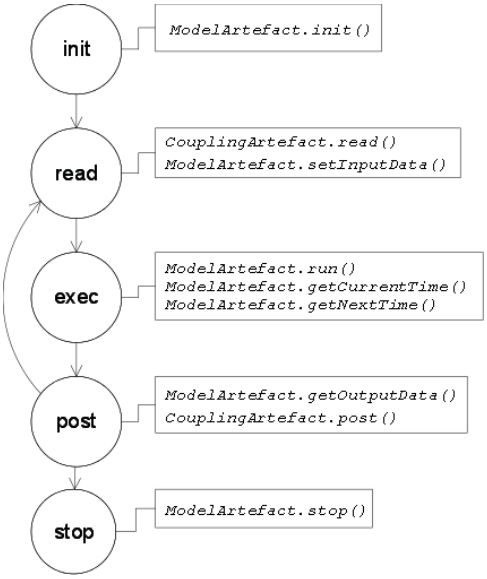

Figure 4: The *model-agent* behaviour: to manage the simulation process.

4. PROOF OF CONCEPT

The objective of this section is to present how to build a complex simulation as a society of interacting models. The simulation results validation is out of the scope of this article. We present a proof of concept illustrated by NetLogo [18] models co-evolution. We choose those simple models for the sake of illustration: they are straightforward to understand and we think they highlight some of the coupling and synchronization problems we raised in section 2. We discussed more evolved examples in section 5.

We implement our concepts in a framework called AA4MM[1]. It is coded in Java and rely upon the Java Messaging Service in order to deal with shared memory purposes. Since this framework, at first, has been developed in order to build a proof of concepts and since the discussion about implementation choices and performances is out of the scope of this article, we do not give a lot of details about this framework. Interested readers can find more details on the AA4MM web page.

4.1 A sheepfold study made of sheep, grass and wolves models co-evolution

Imagine we want to predict the impact of a sheepfold. For example: what are the best herding strategies ? What is the risk for a sheep to be eaten by a wolf ? *etc.* We want to model an ecosystem made of wolves, sheep, shepherds and grasslands. The first model M_1 represents sheep moving and shepherds trying to herd them. The second one M_2 represents sheep[2] eating grass and gaining energy. The last one M_3, is a prey predator model that represents the wolf predation. All models are available within the Netlogo [18] platform.

Our goal is to design the sheepfold simulation as a set of interacting and co-evolving models. We want to couple the three above-mentioned models using the concepts we present

[1]http://www.loria.fr/∼siebertj/aa4mm/

[2]Originally rabbits. We changed species.

in section 3.

4.2 Integration basic steps

At first we only consider M_1 and M_2. Assume we are more preoccupied by the impact of sheep on the grasslands. We target the addition of M_3 later (section 4.4). Both specialists of sheep model and grassland model should be able to make their own models and simulators interact.

The first thing to do to couple different models is to define the input and output ports of each model and the connections between them. The sheep movements are described by the sheep model M_1. That is M_1 provides the set of sheep positions SP. In order to be executed, M_2 needs to update its own local sheep positions from M_1. On its part, the sheep movement is influenced by the sheep energy SE. Energy is provided by M_2. As a consequence, M_1 needs to update its own local sheep energy values from M_2. Figure 5 depicts both conceptual links between M_1 and M_2 and how it is implemented.

4.2.1 Existing models integration

The second thing to do to couple different models is to build the *model-artefacts* for each involved model. Each specialist have to create a *model-artefact* for its model. Consider M_1, its behaviour is given by the algorithm 1. The *model-artefact* in charge of M_1 must implement the functions listed in table 1.

The *init()* function sets the number of sheep, shepherds and their initial positions in M_1. Since Netlogo processes simulation step by step, the *run()* function executes one simulation step of M_1, the *getCurrentTime()* and *getNextTime()* returns ct_1 (the number of clock ticks) and $nt_1 = ct_1 + 1$. *getOutputData()* function returns all sheep positions SP. *setInputData()* sets into M_1 the sheep energy SE.

The same work is done for M_2. All those functions are implemented by simply calling procedures through the provided API. As a consequence, we do not modify the original model. However, the simulators has to provide the mentioned functions (*run()*, *init()etc.*). Thus, we modify the execution process.

Algorithm 1: Model M_1 (one step)

Input: Sheep energy levels SE
Output: Sheep Positions SP
Set the sheep energy levels from SE.
foreach Shepherd B:
  B moves randomly.
  if B is not carrying a sheep.
  then Search for a sheep.
  else Find a herd to drop the sheep.
foreach Sheep S not carried:
  S moves randomly.
tick $\leftarrow$ tick + 1.
return SP

4.2.2 Coupling models

In our example, one *coupling-artefact* cA_1 is in charge of the sheep positions. The other one cA_2 deals with sheep energy (figure 5).

Consider cA_1, its *post()* function allows M_1 for sending the sheep positions with their validity interval: $< SP, \Gamma_1 >$. The *read*() function provides safe sheep positions to M_2. We assume that both models M_1 and M_2 have the same representation of the sheep positions. We will see in section 4.3 how to deal with different scales and representation. It is worth noticing that the specialists involved in the simulation do not have to deal with coordination issues since the coordination model is already implemented into the *coupling-artefact*.

4.2.3 Model-agent behaviour

Once the artefacts are built, the last thing to do is to build the *model-agents* that will use them. One *model agent* is created for each model. The generic *model-agent* behaviour is presented in figure 4. In our example, agents have to create and to initialize their own model via the proper *model-artefact* functions *init()*. Then, they post the initial output data (SE and SP at time 0).

The agents manage the simulation process as follows.

1. Read the input data SE (resp. SP) from the *coupling-artefact* cA_2 (resp. $cA1$). Set them to the *model* one.
2. Execute the model M_1 (resp. M_2) (i.e call the *model-artefact run()* function).
3. Get the current and next time values ct_1 and nt_1 (resp. ct_2 and nt_2).
4. Get output data SP (resp. SE) from the *model-artefact*. Post them to the *coupling* one cA_1 (resp. $cA2$).

The agents do this loop until the simulation ends.

4.2.4 Synthesis

In this section, we presented a case study in which a simulation is built from two models. Notice that those models come from different domains, have been designed independently and can be ran separately. We assume that the specialists of each domain wish to couple those models in order to build a more complex simulation. In order to make those models interact they need to deal with issues such as coupling, coordination and model execution. However, they are not familiar with distributed simulations and coordination process.

The aim of our framework is to facilitate the creation of this simulation by making the coupling and coordination issues transparent. When the specialists need to couple models, they need to define the data flows between them and to implement a *coupling-artefact* for each link. They do not have to care about the synchronization process since the coordination model is already implemented into the *coupling-artefacts* and the exchanged data. A *model-artefact* has to be created for each model to interface. There are six functions to implement (see table 1). We rely upon the work in [19] in order to define and make these functions as generic as possible, anyway existing models may be adapted in order to match them. In the next sections, we show that, having done these simple modifications, it is straightforward to add or interchange models and also to deal with the interaction of models designed in different scales.

4.3 Dealing with scales differences

Until now, we have assumed that time and space scales in both models M_1 and M_2 were the same. In the reality, specialists design their models independently. Most of the time,

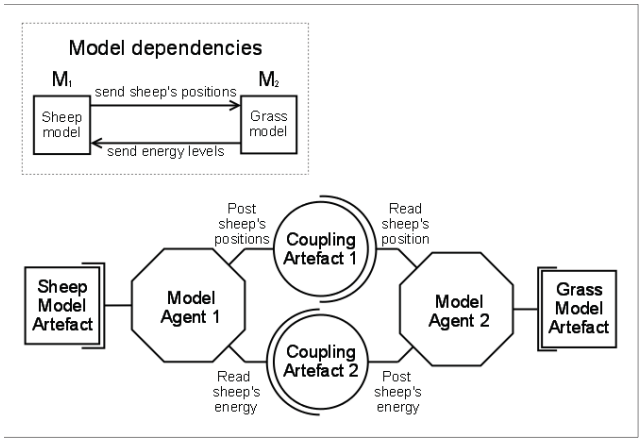

Figure 5: Coupling sheep and grass models from the conceptual view to the implementation

the models scales are different. For example, the grassland specialists are interested into how fields evolve each season although sheep specialists are more concerned with how the sheep population evolves each week.

As a consequence, the time and space scales in both models M_1 and M_2 are not similar. This firstly causes coherence and compatibility issues (see section 2). For example data such as positions in model M_1 do not correspond to positions in model M_2. Secondly, models executions are different. E.g., when the model M_2 computes the fields evolution from winter to spring, the model M_1 needs to compute the sheep population evolution for the 12 corresponding weeks. Although M_2 processes one single simulation event, M_1 processes 12 of them. So the models need to be coordinate.

In the next sections, we present how to make models with different scales interact through our framework.

4.3.1 Different space scales

Consider that one patch of grass correspond to a square of 2×2 patches in the sheep model M_1 (see figure 6). Since it is not possible to change the patches size in the grass model M_2, sheep positions produced by M_1 does not fit anymore with space in M_2. As a consequence, we need to add an operation in the *coupling-artefact* cA_1 when the grass *model-agent* reads the sheep positions. This operation consists in dividing each sheep position coordinates by a factor 2.

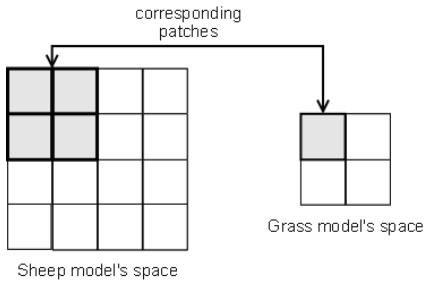

Figure 6: Sheep and grass models spaces correspondence

Note that we target a challenge due to the exchanged data coherence. We only modify the entity in charge of that issue: the *coupling-artefact*.

4.3.2 Different time scales

The same way, consider that the grass model M_2 may no longer be executed step by step but 2 steps by 2 steps; while the sheep model execution remains step by step (see figure 7). This is related to the model M_2 execution process. So we modify the related *model-artefacts* functions *run()* and *getNextTime()*. That is, instead of executing only one simulation step, the *run()* function executes two simulation steps. Then the *getNextTime()* function returns $nt_2 = ct_2 + 2$. Since the coordination occurs only thanks to time values given by *getCurrentTime()* and *getNextTime()* functions, these are the only modifications to make.

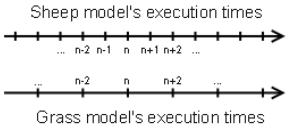

Figure 7: Sheep and grass models execution time correspondence

4.4 Adding, or exchanging models

In this section, we present the advantages to consider model as component and the simulation as a distributed set of interacting models.

4.4.1 Simple addition

The sheep and grass specialists involved in the sheepfold simulation wish to predict the impact of wolves living in the neighbourhood. As a consequence, a new model M_3 should be added in the simulation. The challenge is to couple a new model to the previous ones without rebuilding the whole simulation (see figure 8).

The first thing to do is to define the new links between three the models. The specialists have to know which data are exchanged between M_1, M_2 and M_3. Two cases are possible. Either the exchanged data are the same as those existing or they bring into play new ones. On the one hand, we can reuse the existing *coupling-artefacts*. On the other hand, we need to create new *coupling-artefacts* (as described in section 4.2).

Consider M_3, this models defines the wolf predation dynamics: wolves move, catch a sheep and took energy from the sheep (as they eat them). This way, M_3 needs to update its own local sheep position values from M_1 and M_3 provides negative sheep energy values SE_3 when they eat the sheep. The links between the models are described in figure 8. Since they do not require some new data to exchange, we can reuse the existing *coupling-artefacts* cA_1 and cA_2.

Since it is already present, M_3 can ask the *coupling-artefact* cA_1 to give it the sheep positions SP from M_1. The only thing to do is to add an operation to cA_1 in order for M_3 to read that value. However, when M_1 ask cA_2 to give it the sheep energy values, it has to take into account both the positive values SE_2 given by the grass model M_2 and the negative values SE_3 given by the wolf model M_3. That is, we need to define a new operation in cA_2 in order to combine both energy levels. In our case, for a given sheep energy level, cA_2 needs to add the energy gain SE_2 and the energy loss SE_3. This way M_1 updates a single sheep energy value for each sheep.

4.4.2 A (little bit) more complicated addition

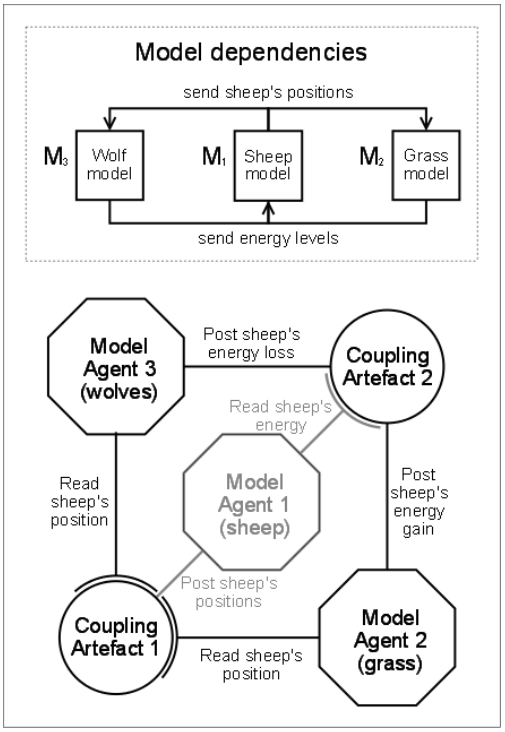

Figure 8: Addition of a third model

As we saw in section 4.3, M_1, M_2 and M_3 may have been designed independently and, as a consequence, scales in each model may be different. For example, the grassland model represents fields evolution for each season, the sheep model represents the sheep population evolution for each week and the wolves model represents the wolf predation techniques for each day. Assume that we couple the three model as in figure 8.

There is no need to care about coordination since the coordination model is already implemented into the *coupling-artefacts* and the exchanged data. The only modifications to make concern the *run()* and *getNextTime()* functions as in section 4.3.2.

In our example, the *coupling-artefact* cA_1 needs to provide sheep positions from M_1 to both models M_2 and M_3. Since the sheep positions in these models are different, cA_1 need an operation to translate sheep positions from M_1 to M_2 (as in section 4.3.1) and, in the same way, another one that translate sheep positions from M_1 to M_3.

Note that, since the *coupling-artefacts* store the data sent by the *model-agents* it is possible to define as many operations as needed in order to deal with the coupling issues (see section 2). As a conclusion, if works from sections 4.3 and 4.4.1 have already been done, no more modifications are needed in order to build the new coupling scheme.

4.4.3 Interchange

After some measurements, one may find that a model is not accurate enough and that another one should be better. In order to interchange models, the simulation designers simply need to check that they will have the same input and output ports. For example, in M_1, instead of moving randomly, the sheep can move accordingly to a flocking model (as [7]). Instead of rewriting a new model, we can simply replace M_1 by an existing flocking model M_1' that have the same input and output ports. This way, we do not change the *model-agents* nor the *coupling-artefacts*. We only need to build a *model-artefact* for the new model M_1' (see figure 9).

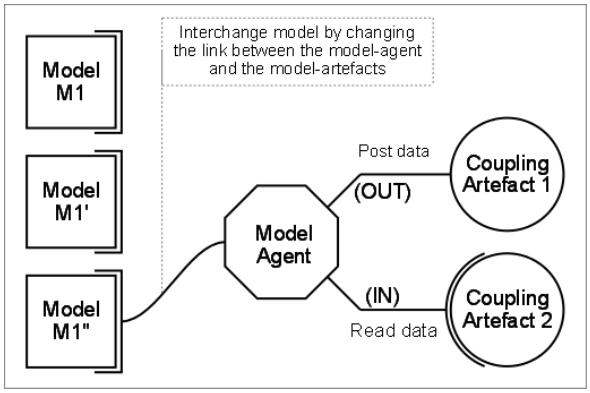

Figure 9: Interchanging models

4.4.4 Building new simulations

Imagine now that we are more interested in reintroducing wolves into the wild than in the sheep industry. However, we want to reuse the existing models and results we previously obtained. That is, we want to build new simulations from the same building blocks. In addition, new models may be needed. In order to build this simulation, we do not have to rewrite neither all the models, nor the coordination process. We only have to take the building blocks and to link them properly. This is one of the advantages to see simulation as a set of interacting and co-evolving models, and to have the concepts that allows to easily implement it.

Indeed, it is worth noticing that none of the *model-agent* needs to know each other, neither they know which reads the data they post. This simplifies the addition, the removal or the interchange of the models. The important thing is to check the dependencies between the models. That is, the interacting models should have the right input and output ports. Then, it is straightforward to implement new *coupling-artefacts*, to reuse the existing ones and to link them with the proper *model-agents*.

5. CONCLUSION

5.1 Summary

In this article, we present an architecture to build simulations as a society of interacting and co-evolving models. It is based upon the A&A paradigm. We also propose an objective coordination model with decentralized models execution. Our concepts facilitate the building of such simulations since it makes the coupling and coordination issues transparent for the users. This way, people involved into the simulation design only have to care about modelling and about the data flows between models.

The architecture we propose is modular and decentralized so that any model could be added, removed or interchanged with as little modifications as possible. We have developed a formal coordination model and proved that synchronization happens without deadlocks for a number of simulators $k \in$

$\mathbb{N}$. We implemented our concepts in order to build a proof of concept illustrated by several examples.

5.2 Discussion and future directions

In this article, we have made the assumption that the complex system studied is divided into several abstraction levels. One may want a geographical division where the subsystems may represent different geographical subparts. As a consequence, entities can enter, leave the models or go from a model to another. This point of view raises new modelling and coupling challenges. We think the concepts presented in this article can help to solve these new issues.

We use this framework on an ongoing work on co-evolution of mobile ad hoc networks and social models. We then couple an event-based simulator (network model) with a step-by-step simulator (agent based model). It is part of a french ANR research project. We plan to use some of the experiments to improve and study the implementation performances.

Although the examples presented are quite simple and broadly compatible (both discrete event), we think that with the concept presented here we can be used to couple equation based models (EBM) with agent based ones. Technically, we need to resolve (or *run()*) EBM for time intervals instead of events or steps. It is possible to go from the micro scale to the macro one by using operations (as defined in section 2) such as sums, means or distributions, *etc.* ; the other way round by, for example, introducing randomness. Discrete event formalism and the work in [19] should be a good line of thinking in order to look further.

Another open issue would be in the validation process of co-evolution. One question that arise is the following: do several independent and valid models give valid results when coupled together ? We think that this question is domain related and that complex simulations have to be done and compared to real measurement in order to answer this issue.

Acknowledgements

The authors would like to thanks ANR SARAH project and La Région Lorraine for their financial support, Joris Rehm[3] and Virginie Galtier-Ciarletta[4] for their collaboration.

6. REFERENCES

[1] S. Bonneaud, P. Redou, and P. Chevaillier. Pattern oriented agent-based multi-modeling of exploited ecosystems. In *6th EUROSIM congress on modelling and simulation*, september 9-13 2007.

[2] R. M. Fujimoto. Parallel simulation: parallel and distributed simulation systems. In *WSC '01: Proceedings of the 33nd conference on Winter simulation*, pages 147–157, Washington, DC, USA, 2001. IEEE Computer Society.

[3] J. Himmelspach and A. M. Uhrmacher. Plug'n simulate. In *ANSS '07: Proceedings of the 40th Annual Simulation Symposium*, pages 137–143, Washington, DC, USA, 2007. IEEE Computer Society.

[4] J. A. Joines and S. D. Roberts. Fundamentals of object-oriented simulation. In *WSC '98: Proceedings of the 30th conference on Winter simulation*, pages 141–150, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[5] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating computer simulation systems: an introduction to the high level architecture*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[6] G. Quesnel, R. Duboz, D. Versmisse, and E. Ramat. The virtual laboratory environment: A multimodelling and simulation framework. In *Transactions on Modeling and Computer Simulation*. ACM, 2009.

[7] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics*, pages 25–34, 1987.

[8] A. Ricci, M. Viroli, and A. Omicini. Give agents their artifacts: the a&a approach for engineering working environments in mas. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA, 2007. ACM.

[9] G. F. Riley, M. H. Ammar, R. M. Fujimoto, A. Park, K. Perumalla, and D. Xu. A federated approach to distributed network simulation. *ACM Trans. Model. Comput. Simul.*, 14(2):116–148, 2004.

[10] M. Rupert, A. Rattrout, and S. Hassas. The web from a complex adaptive systems perspective. *J. Comput. Syst. Sci.*, 74(2):133–145, 2008.

[11] M. Schumacher. *Objective coordination in multi-agent system engineering: design and implementation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[12] J. Siebert, L. Ciarletta, and V. Chevrier. De l'intérêt du couplage de modèles pour appréhender les interactions utilisateurs-réseaux dynamiques. *Revue d'Intelligence Artificielle*, 2009.

[13] J. Siebert, L. Ciarletta, and V. Chevrier. Agents & artefacts for multiple models coordination. In *25th Symposium On Applied Computing*, 2010.

[14] J. Siebert, J. Rehm, V. Chevrier, L. Ciarletta, and D. Mery. Aa4mm coordination model: event-b specification. Technical report, INRIA, 2009.

[15] J. Southern, J. Pitt-Francis, J. Whiteley, D. Stokeley, H. Kobashi, R. Nobes, Y. Kadooka, and D. Gavaghan. Multi-scale computational modelling in biology and physiology. *Progress in Biophysics and Molecular Biology*, (96), 2008.

[16] H. Van Dyke Parunak, S. Brueckner, and J. Sauter. Digital pheromone mechanisms for coordination of unmanned vehicles. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 449–450, New York, NY, USA, 2002. ACM.

[17] H. Van Dyke Parunak, R. Savit, and R. L. Riolo. Agent-based modeling vs. equation-based modeling: A case study and users' guide. In *MABS*, pages 10–25, 1998.

[18] U. Wilensky. Netlogo, 1999. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[19] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation*. Academic Press, January 2000.

[3]joris.rehm@loria.fr; MOSEL Team, LORIA.
[4]virginie.galtier@supelec.fr; Supelec Metz.