Agents and Artefacts for Multiple Models coordination. Objective and decentralized coordination of simulators.

Julien Siebert, Laurent Ciarletta, Vincent Chevrier

Agents & Artefacts for Multiple Models coordination

Objective and decentralized coordination of simulators

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

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

Vincent Chevrier
Université Henri Poincaré (Nancy 1)
vincent.chevrier@loria.fr

LORIA - Campus Scientifique - BP 239 - 54506 Vandoeuvre-lès-Nancy Cedex

ABSTRACT

Complex systems simulation implies the interaction of different scientific fields. However, most of the time people involved into the simulation process do not know intricate distributed simulation tools and only care about their own domain modelling. We propose a framework (called AA4MM) to build a simulation as a society of interacting models. The main goal is to reuse existing models and simulators and to make them interact. The coordination challenges remain to the AA4MM framework so that the simulation design and implementation stay as simple as possible. In this paper, we present the coordination model which intends to decentralize the simulators interactions. We propose to use the environment through the notion of artefact in order to deal with the coherence, compatibility and coordination issues that appear in parallel simulations.

Keywords

Multiple interacting models. A&A paradigm. Data-driven coordination model. Decentralization.

1. INTRODUCTION

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 [12]

Complex systems modelling also involves the interaction of different scientific domains or different abstraction levels. In biology for example, in order to understand and to predict the impact of a molecule on a specific organism, both models from chemistry (chemical reaction) and biology (cellular, tissue, organ) are needed [10]. 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 design and the implementation of such a simulation is to build it as a society of interacting models. Models should be seen as components we can weave together (as in component-based software engineering).

We propose a framework (called AA4MM) to build a simulation as a society of interacting models. The main goal is to reuse existing models and simulators and to make them interact. However, the main constraint is that people involved into the simulation process do not know intricate distributed simulation tools and only have to care about their own domain modelling. Coordination challenges remain to the AA4MM framework.

2. CHALLENGES AND RELATED WORKS

Contrary to the work in [5], where all models are integrated into DEVS formalism and run in a single simulator, we assume that each model has been created independently and implemented in its own simulator. Consequently, each model has its own representation of time and data. In the same way, each simulator proceeds its own execution. The following sections list issues that appear when coupling different models and simulators.

2.1 Coherence and compatibility issues

2.1.1 Coherence between models

Scales or dimensions in which a piece of data is represented could be different from a model to another. 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, discretisation, reduction) in order to achieve this coherence.

Moreover, each model could have its own time representation. We need to assure, 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. A possible solution is to express an operation that makes the correspondence between both time values.

2.1.2 Compatibility between simulators

SAC'10 March 22-26, 2010, Sierre, Switzerland.

Each simulator could implement a single piece of data in its own way (integer, float...) or some simulators may not implement all aspects of a given model. These challenges are discussed in [6]. 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 Coordination issues

The goal of time management in distributed simulation is to ensure that simulation events (or steps) are executed in the correct order. 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 2.2.1. *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 $> ct_i$.*

Conservative coordination can be done by using a central and global scheduler that synchronizes all the simulators as in [1, 6, 3]. These solutions prevent to easily reuse the existing models and simulators since they need substantial modifications in order to be controlled by the scheduler. Moreover, a global scheduler imposes a bottleneck that arduously allows the simulation to scale up in terms of either systems size or number of abstraction levels. In section 3, we remove this central scheduler and we propose a decentralized coordination model.

3. PROPOSAL

3.1 Hypothesis and requirements

We do not target on-line nor real time simulations, which directly interact with the reality. Instead we focus on a model that fits our initial requirements. We try to facilitate the design of a society of interacting models by suppressing the global scheduler and by modifying as little as possible the existing models and simulators used.

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 [11, 7] (stigmergy) or in parallel systems [8] (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.

3.2 Validity interval and coordination

3.2.1 Description

Each model M_i holds a current simulation time value ct_i. The simulator S_i knows the simulation time value for the next event to be processed nt_i. 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 M_i is executed (at time nt_i). 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 2.2.1). Then, the issue for the simulator is to know when an event is safe. It can be solved 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 and only if:

$$\forall j \neq i : ct_i \in \Gamma_j$$

This way, the input data $< \delta_j; \Gamma_j >$ are safe for the simulator S_i and the latter can process the simulation event at time ct_i.

3.2.2 Properties

We have developed a formal specification (in event-B) of this coordination model. Describing the whole specification is out of the scope of this article. However, it is available in [9]. This formal specification is used to prove that coordination occurs between models and that the system is alive and deadlock free with k models ($k \in \mathbf{N}$). A sketch of the proof, using *reductio ad absurdum* is presented hereafter.

Assume, within this coordination model, that a simulator S_i is stopped at time ct_i. It is waiting for input data $< \delta_j, \Gamma_j >$ from another simulator S_j ($i \neq j$). This simulator cannot send data because it is also stopped but at a time $ct_j < ct_i$. S_j also wait for input data. Two cases appear. Either the simulator S_j is waiting for input data from S_i. In this case it means that S_i is waiting for itself, that contravenes our assumption. Or the simulator S_j is waiting for input data $< \delta_k, \Gamma_k >$ from another simulator S_k at time $ct_k < ct_j < ct_i$ with $(i \neq j \neq k)$. In this case, we come back to the very first case. Since the number of simulators and the number of simulation events are assumed to be finite, and since $\forall i : ct_i \geq 0$, we can show by recursion that the latter case means that initial conditions are not set correctly and then that the simulation cannot happen. This contravenes our initial assumption. We target the initial conditions on an example in section 5.2.

4. FRAMEWORK OVERVIEW

In this section, we present how the A&A paradigm is used to take up the coupling challenges and how it implements the coordination model.

4.1 Architecture

In agent oriented software engineering, agents are autonomous entities that interact with each other and with their environments in order to solve a given task [12]. In the A&A paradigm [4], the artefacts are used to design and to implement the interactions. They can be seen as tools used by the agents. In the case of building a simulation as a society of interacting models and simulators, the agents are in charge of the models execution and they interact through some specific artefacts. The *coupling-artefact* allows the agents to exchange data. It is in charge of coherence and compatibility issues and it implements the coordination model. The *model-artefact* allows the agents to initialize, to execute the model, to send input data and to get output data. The figure 1 depicts this architecture. An implementation example is given in the section 5 and the corresponding agents and artefacts used are described by figure 4.

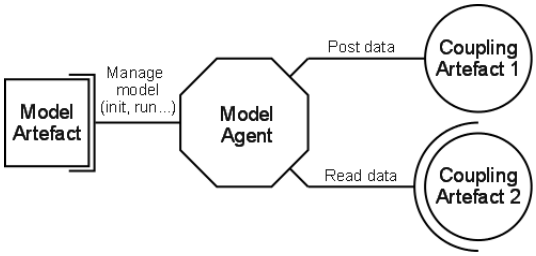

Figure 1: Architecture overview

4.2 Artefacts functions

In A&A paradigm [4], artefacts hold *functions* that agents can use. In this section, we present details about the *model-artefact* and the *coupling-artefacts*.

4.2.1 Model-artefact

The *model-artefact* role is to allow the agents to operate on a given model. We propose 6 functions. *Init()* allows to create a model instance and to initialize it. *Run()* allows to run the simulation only for one step or for one event. Then, in order to exchange some data between models, next functions are *getOutputData()* and *setInputData()*. Finally, as they are needed for the coordination process, last functions are *getCurrentTime()* that returns ct_i and *getNextTime()* that returns nt_i (Cf 3.2).

4.2.2 Coupling-artefact

The *coupling-artefact* allows the agents to *post()* and *read()* data δ_i. However, this artefact also intends to prepare data and to filter them. Thus, the *post()* function adds the validity interval Γ_i to the data δ_i. The *read()* function includes the guard condition $ct_i \in \Gamma_j$ to coordinate models. *read()* only returns valid and last produced data to the agent. Moreover, it is possible to add operations (as in [1]) in order to deal with coherence and compatibility issues (Cf 2).

4.3 Agents behaviour

The role of the agent (a *model-agent* in our case) is to run a specific task. The artefacts are tools it can use. The very first role of a *model-agent* is to execute the model and to read and post data. We distinguish three major phases. First, the agent has to create and initialize the artefacts it is going to use (at least one *model* and one *coupling-artefact*). Then, the agent manages the simulation process as describe on figure 2. It loops over the five steps. Finally, once the simulation is over, this agent can retrieve results and save them for analysis.

5. IMPLEMENTATION EXAMPLE

In this section, we present an example of a simulation made of different interacting simulators. We choose to use Netlogo [14] since it is easy to understand and well known. Note that each model is executed independently in its own Netlogo instance, *i.e.* there are as many simulators running in parallel as interacting models. Implementation details and other use cases are discussed in section 6.

5.1 Coordination of existing Netlogo models

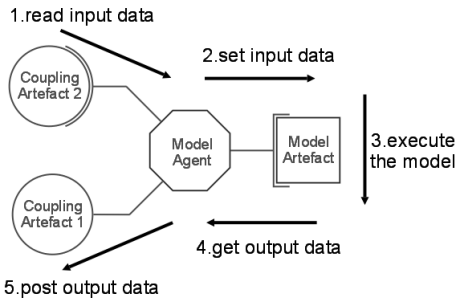

Figure 2: A *model-agent* managing the simulation process

The example is the following: assume we want to build a simulation of a sheepfold. We are interested into the influence between the sheep movements and the dynamic of sheep grazing. Models already exist for each dynamic [13, 15]. The first one M_1, the sheep model, depicts the sheep movements: sheep move randomly, lose some energy and shepherds try to gather them. The second model M_2, the grass model, represents sheep[1] eating grass and gaining energy. We want these two dynamics to influence each other. That is, M_1 must send the sheep positions to M_2 while M_2 gives the sheep energy levels back to M_1 (see figure 3). The architecture with A&A concepts is represented by figure 4.

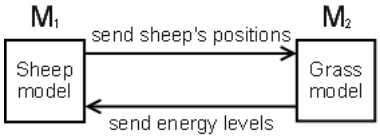

Figure 3: Sheep - grass models dependency network

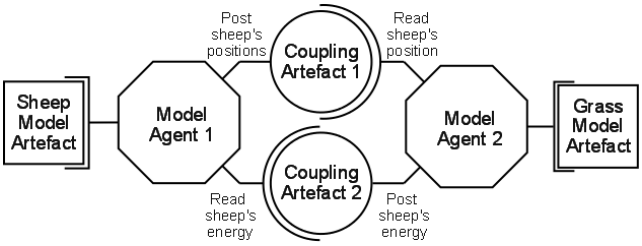

Figure 4: Sheep and grass models implementation

5.1.1 Model artefacts design

In order to make both simulators S_1 and S_2 (and their own models M_1 and M_2) interact, we need to interface them by creating the *model-artefacts* (one for each). In 4.2, we define *model-artefact* functions. We implement them, without modifying the original models, by simply calling procedures through the provided Netlogo API.

For example, the functions of the *model-artefact* in charge of M_1 are designed as follows. The *Init()* function, that sets the number of sheep, shepherds and their initial positions in M_1, is a call for the "setup" Netlogo procedure. The *Run()*

[1]Originally rabbits, we changed species.

function, that runs one simulation step of M_1, calls the "go" procedure. *getCurrentTime()* and *getNextTime()*, that returns ct_1 and nt_1, report the number of "ticks" and "ticks + 1" (as we process simulation step by step). *getOutputData()* function, that returns the sheep positions δ_1, reports all sheep present in the model and gathers their positions. *setInputData()* sets into M_1 the sheep energy δ_2 produced by M_2. This is done by invoking the "set energy"Netlogo command on all the sheep in M_1.

5.1.2 Coupling-artefacts design and architecture overview

Once the dependency network is done (figure 3), we can design the *coupling-artefacts*. We propose that each *coupling-artefact* is in charge of only one set of data $< \delta_i, \Gamma_i >$. In our example, one *coupling-artefact*, cA_1, is in charge of sheep positions $< \delta_1, \Gamma_1 >$. The other one, cA_2, is in charge of the sheep energy levels $< \delta_2, \Gamma_2 >$.

This way, the *model-agent* in charge of M_1 can post sheep positions $< \delta_1, \Gamma_1 >$ to cA_1 and read sheep energy levels $< \delta_2, \Gamma_2 >$ from cA_2 (Cf section 3.2).

Finally, we link each *model-agent* with its dedicated artefacts. Thus, the implementation strictly follows the initial dependency network. The whole architecture is represented in figure 4.

5.2 Model and simulators coordination

We saw on figure 3 that each model is waiting for data from the other one. In order to bootstrap the simulation, each *model-agent* must post initial data to the *coupling-artefacts*. Initial sheep positions $< \delta_1; [0, 1[>$ and initial sheep energy levels $< \delta_2; [0, 1[>$ have to be sent to their respective *coupling-artefacts* before the simulation process begins.

Then, the models execution follows the process described by figure 2. Once the *model-agent* 1 has read the sheep energy levels from the *coupling-artefact* 2, it sends them to the sheep *model-artefact*, executes M_1, gets the sheep positions and posts them to the *coupling-artefact* 1. On the other side, the *model-agent* 2 reads and sends the sheep positions to the grass *model-artefact*, executes M_2, gets sheep energy levels and posts them. In fact *model-agents* wait for each other and synchronize themselves thanks to the exchanged data δ_i present in their environment.

5.3 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 next sections, we present how that framework is useful to make models with different scales interact.

5.3.1 Different space scales

It may not be necessary to represent grass as precise as sheep are. That is, one patch of grass may correspond to a square of 2×2 patches in the sheep model M_1 (Cf figure 5). 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.

Here, we target a challenge due to the exchanged data

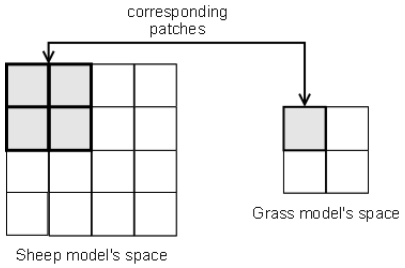

Figure 5: Sheep and grass models spaces correspondence

coherence. We only modify the entity in charge of that issue: the *coupling-artefact*. However, since we have altered the space in the grass model M_2, we may also want to change the model behaviour. For example, how the grass on a patch is eaten could now be a function of the number of sheep on that patch. To do that, we only need to change the model M_2 itself; no matter to change either the *model-artefact* or the *model-agent*.

5.3.2 Different time scales

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 (Cf figure 6). This is related

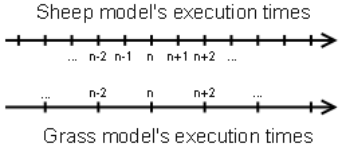

Figure 6: Sheep and grass models execution times correspondence

to the model execution process. So we modify the grass *model-agent Run()* and *getNextTime()* functions. That is, instead of calling the "go" Netlogo procedure only once in the *Run()* function, the *model-agent* calls it twice. Then *getNextTime()* returns now the number of "ticks + 2". These are the only modifications to do since the coordination occurs only thanks to time values given by *getCurrentTime()* and *getNextTime()* functions.

6. DISCUSSION

All the framework has been developed in Java since it makes Java Messaging Service platform[2] and Netlogo integration easier. JMS platform is used for shared memory purposes. In this article, we present an example based upon the Netlogo platform that makes only two simulators (and their models) interact. Due to space constraints, we do not present all the features allowed by that framework. Indeed, coupling and synchronizing the *model-agents* through their environment deeply simplify the addition of a new model. We have made experiments with three Netlogo models (both M_1 and M_2 plus a model of wolves predation).

Changing the dependences between the models is also simplified by the use of one *coupling-artefact* for each kind of

[2]Java Messaging Service. http://java.sun.com/products/jms/

dependence. That is, if M_1 and M_2 both depends on the data δ_3 provided by a third model M_3, we just need to build a *coupling-artefact* in charge of δ_3. M_1 and M_2 will read that data from this new *coupling-artefact*.

In this article, we only talk about step-by-step simulation. We currently work on mobile ad hoc networks (MANET) simulations in which an event-driven simulator interact with a step-by-step multiagent simulator. The coordination model is exactly the same as the one described here. No additional modification is needed in order to integrate an event-driven simulator into the AA4MM framework.

The technical details, the examples and source code are available on the framework webpage[3].

7. CONCLUSION

We propose a framework (called AA4MM) to build a simulation as a society of interacting models. The main goal is to reuse existing models and simulators and to make them interact. However, the main constraint is that people involved into the simulation process do not know intricate distributed simulation tools and only care about their own domain modelling. That is, coordination challenges remain to the AA4MM framework.

In this paper, we present the coordination model in order to decentralize the simulator interactions. We propose to use the environment through the notion of artefact in order to deal with the coherence, compatibility and coordination issues that appear in parallel simulations. We have developed a framework that intends to deeply simplify the simulation of complex systems by easily building a society of interacting models and simulators.

We do not target the implementation performances. Indeed, parallel simulations have the advantage of theoretically scaling up. Large size systems or numerous abstraction levels may be simulated. However, the data exchange between simulators can cause a huge overhead and can slow down the whole simulation. These scalability issues are plan as future work.

We do not mention the open system consideration. That is, we only focus on different interacting models where agents do not enter or leave their model. We think that when an agent leaves, enters or goes from one model to another, exchanging data is not sufficient. We plan to extend our framework in order to deal with that issue. The challenge here is to respect our requirements of coordination via the environment.

This work has been motivated by our initial studies on the interactions between humans behaviour and dynamic networks. Since we have now the tools to reuse existing models and simulators, we plan to focus on the experiments in this domain.

8. ACKNOWLEDGEMENTS

The authors would like to thank the ANR SARAH project and La Region Lorraine for their financial support. Coordination formal specification in event B has been developed in collaboration with Joris Rehm[4]. JMS implementation has been done in collaboration with Virginie Galtier[5].

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

[4]joris.rehm@loria.fr; MOSEL Team, LORIA.

[5]virginie.galtier@supelec.fr; Supelec Metz.

9. 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] 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.

[4] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] U. Wilensky. Netlogo shepherds model, 1998. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

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

[15] U. Wilensky. Netlogo rabbits grass weeds model, 2001. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.