Principles and Experiments of a Multi-Agent Approach for Large Co-Simulation Networks Initialization

Jérémy Boes, Tom Jorquera, and Guy Camilleri
University of Toulouse, IRIT/Team SMAC, 118 route de Narbonne, Toulouse, France
{*name.surname*}*@irit.fr*

Keywords: Multi-Agent Systems : Self-Adaptive Systems : Co-Simulation

Abstract: Simulating large systems, such as smart grids, often requires to build a network of specific simulators. Making heterogeneous simulators work together is a challenge in itself, but recent advances in the field of co-simulation are providing answers. However, one key problem arises, and has not been sufficiently addressed: the initialization of such networks. Many simulators need to have proper input values to start. But in the network, each input is another simulator's output. One have to find the initial input values of all simulators so their computed output is equal to the initial input value of the other linked simulators. Given that simulators often contain differential equations, this is hard to solve even with a small number of simulators, and nearly impossible with a large number of them. In this paper, we present a mutli-agent system designed to solve the co-simulation initialization problem, and show preliminary results on large networks.

1 INTRODUCTION

Alternatives to nuclear power, renewable energy like solar panels or wind power, often involves producing electricity near the consumers. This radically changes the topology of the electric grid and makes it way more difficult to manage and control. Smart grids are a very active research field. Simulating smart grids enables the validation of network control methods and various network properties, that are as important as the (simulated) physical properties of the power system (Liberatore and Al-Hammouri, 2011). Nevertheless, designing a simulator and building a model for large networks is a difficult task due to the complexity of the considered systems. Co-simulation overcomes this difficulty by running several models and simulators together, one for each subsystem. In this approach, simulators exchange data with each other, in a "black box" way. Models are then easier to obtain since they only focus on a smaller part of the grid. However, new problems arise, such as time synchronization (Bouchhima et al., 2006). Most of them are handled by existing co-simulation frameworks, but the mutual and interdependent initialization of each simulator is not.

Indeed, simulators in a co-simulation network require coherent initial values for their inputs. These inputs are connected to the outputs of other simulators. To preserve the coherence in the network and prevents simulators from crashing, the initial values that are set have to be equal (or within a given accuracy limit) to the values of the connected outputs of the other simulators. These output values are computed from the other initial input values, that have been set under the same constraints. Since each input is connected to an output in the network, there is no "first input" that is free to go back to set an arbitrary value and solve the whole problem.

Due to the large scale of co-simulations networks, the heterogeneity and the intrication of the models involved, co-simulation initialization (or co-initialization) is a difficult challenge. It requires a lot of expertise and domain specific knowledge about the models. Our approach aims at developping a co-initialization method that does not rely on such expertise, by considering each simulator as a black-box and exploring the solutions. In this paper, co-initialization is viewed as a fixed-point problem. We use a multi-agent approach to find X such as $F(X) = X$, where F is the function that transforms the inputs of all simulators into their outputs. The distributed and decentralised decision process of multi-agent systems should allow our system to scale up to an increasing number of simulators in the co-simulation network.

Section 2 presents the problem of co-initilization, while our multi-agent system is described in section 3. Section 4 shows results before we conclude in the last section.

2 CO-INITIALIZATION

Co-initialization is the problem of finding adequate initial values simultaneously for all the inputs of all the simulators in a co-simulation network. This section explains how we see this problem as a fixed-point problem.

2.1 Co-Simulation

A co-simulation network is a network of possibly heterogeneous simulators running together to simulate several parts or several scales of a large system. This method enables to model huge systems that would be otherwise too complex to handle with a single model, and to reuse well-establihed and specialized model with only minor modifications. For instance, to simulate an emergency brake maneuver for wind turbines, Sicklinger et al. combine several models (blades, gearbox, control system, etc) and achieve a more realistic result than monolithic approaches (Sicklinger et al., 2015).

Co-simulation has been of growing interest these past years, but despite its efficiency, it does raise a number of challenges. The development of a standard for model exchange, called Functional Mock-up Interface (FMI), brings answers to the problem of models interoperability (Blochwitz et al., 2011). Other tools deal with time coherence in co-simulation networks, such as MECSYCO which relies on the multi-agent paradigm (Vaubourg et al., 2015).

2.2 Co-Initialization as a Fixed-Point Problem

This paper addresses the issue of co-initialization, that arises when simulators and models have constraints on their input. For instance, some models can be unstable if their input changes too widely between two timesteps, or crash if the inputs goes outside a given range. In a co-simulation network, all simulators inputs are usually connected to an output from another simulator. For instance, in figure 1, this means that the initial value of the input of simulator 1 should be equal or close to the output of simulator 3, which depdends on the initial value of the input of simulator 3, and so on with simulator 2.

If we consider simulators as functions, we can express the co-initialization problem as a fixed-point problem. For instance, with simulator 1 as function f, simulator 2 function g, and simulator 3 function h, the initialization problem is to find x_1, x_{2a}, x_{2b}, and x_3

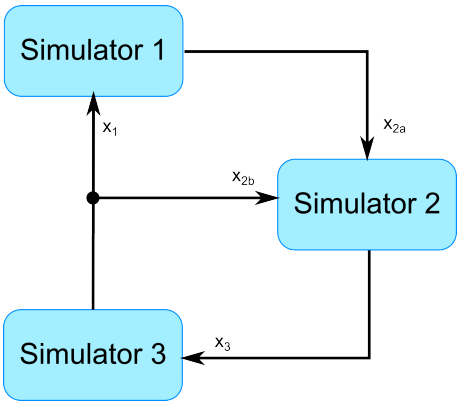

Figure 1: A Co-Simulation Network with 3 Simulators

such as:

$$\begin{cases} f(x_1) = x_{2a} \\ g(x_{2a}, x_{2b}) = x_3 \\ h(x_3) = x_1 = x_{2b} \end{cases}$$

It is a fixed-point problem where we want to solve $X = F(X)$ with $X = (x_1, x_{2a}, x_{2b}, x_3)$ and $F(X) = (h(x_3), f(x_1), h(x_3), g(x_{2a}, x_{2b}))$. In our case, we do not know the models used by the simulators (f, g, and h are unknown). But we have access to their jacobian, so we can know what variation would be caused on an output by a modification of an input for a given simulator.

2.3 Fixed-Point Solvers

The fixed-point problem is a challenging issue that have many applications in fields where the notion of stability can be expressed as a fixed-point, such as economics and game theory (Border, 1989), program analysis and code optimization (Costan et al., 2005), or phase transition in physics (Schaefer and Wambach, 2005).

The most common and proven approaches to solve fixed-point problems are iterative methods (Alber et al., 2003), (Yao et al., 2011). For instance, the Newton-Raphson method (Bonnans et al., 2013) uses the first terms of a Taylor series to approximate a function for which we want to solve an equation.

While their formal definition is sound, this type of approaches tends to suffer from combinatory explosion when faced with a large number of dimensions. This is why we work on an alternative solution with a bottom-up multi-agent approach.

3 A SELF-ADAPTIVE MULTI-AGENT SYSTEM FOR CO-INITIALIZATION

This section details our multi-agent system and explains how agents help their neighborhood so the whole system converge towards a solution.

3.1 Adapative Multi-Agent Systems

Our approach is inspired by the AMAS (Adaptive Multi-Agent Systems) theory, a vision of multi-agent systems where cooperation is the engine of self-organization (Georgé et al., 2011). The AMAS theory is a basis for the design of multi-agent systems, where cooperation drives local agents in a self-organizing process which provides adaptiveness and learning skills to the whole system. As cooperative entities, AMAS agents seek to reach their own local goals as well as they seek to help other agents achieving their own local goals. Thus, an agent modifies its behavior if it thinks that its actions are useless or detrimental to its environment (including other agents).

The AMAS theory has proved useful and efficient to tackle various problems where scaling up is a key challenge, such as big data analytics (Belghache et al., 2016), multi-satellite mission planning (Bonnet et al., 2015), or air traffic simulation (Rantrua et al., 2015). This is why we considered applying this approach to the problem of large co-simulation initialization.

3.2 Topology

The key problem in co-simulation initialization lies in the link between simulators. Thus, there is one agent per link in the network, i.e. one agent for each output of each simulator. In other words, there is only one type of agent in our system, instanciated at each output of each simulator. The neighborhood of an agent A is defined as the agents that are directly affected by the actions of agent A, including itself. Hence, the neighborhood of agent A is composed of agent A and all the agents at the outputs of the simulators connected to the output of agent A. For instance, in figure 2, the neighborhood of agent A is composed of agent A and agent B.

3.3 Agents Description

Each agent has an input and an output. The value of its input is perceived in the environment, it is the output of a simulator. The value of the agent's output is decided by the agent itself, on its own. The global goal of the system is to reach a state in which each agent perceives on its input the same value it had written on its output at the previous timestep.

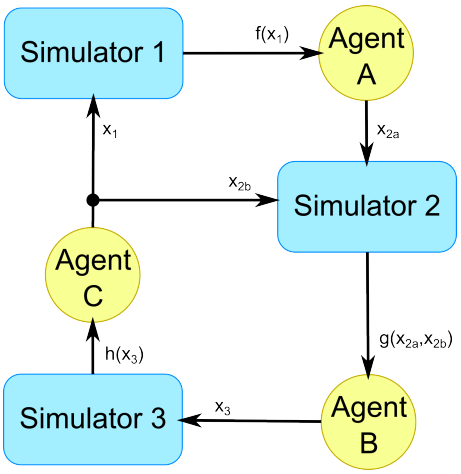

Figure 2: A Co-Simulation Network with 3 Simulators and our Agents

3.3.1 Criticality of an Agent

Each agent seeks to meet the condition $|x_t - y_{t-1}| = 0$, where x is the input of the agent, y its output, and t the timestep. The value $c = |x_t - y_{t-1}|$ is called the *criticality* of the agent. The higher its criticality, the less satisfied an agent is. We define C_{sys} the global criticality of the system as the highest criticality among the agents. By definition, the problem is solved when $C_{sys} = 0$. It is a mere reformulation of the fixed-point problem expressed in 2.2 that is more suitable for our approach.

At each timestep, each agent computes its criticality, watches the criticalities in its neighborhood, and try to correlate their variation to the variation of its own output. The agents adjust their output value in order to lower their neighbors' criticality. The next section explains how they decide their output value.

3.4 Agents Behavior

Each agent has to decide its next action: whether to increase or decrease its value, and by what amount, in order to lower the criticality of its neighborhood as quickly as possible. Its output y at timestep t is:

$$y_t = y_{t-1} + a_t$$

Where a_t is the decided action. The decision is based on two dynamically adjusted internal variables:

- $\delta \in \mathbb{R}^+$, with $\delta_{min} \leq \delta \leq \delta_{max}$ is the amplitude of the action, δ_{min} and δ_{max} are user-defined parameters, which will be discussed later ;

- $\sigma \in \mathbb{N}$ with $-n \leq \sigma \leq n$ indicates the sign of the action, n is a user-defined parameter of the agent, which will be discussed later.

These two variables directly define a:

$$a := \begin{cases} \delta \times \frac{\sigma}{|\sigma|} \text{ if } \sigma \neq 0 \\ random(-\delta, \delta) \text{ otherwise} \end{cases}$$

The sign of σ is the sign of the action, and the absolute value of σ reflects how much the agent is convinced that the sign is correct. If $\sigma = 0$, the decision is a coin flip between $-\delta$ and δ. We introduced randomness here because all agents perceive and decide simultaneously, thus falling into the El Farol bar problem (Whitehead, 2008).

3.4.1 Adaptation

Each agent dynamically manages its own internal δ and σ variables, and decide how to adapt them. This section explains how an agent adjusts its δ and σ.

Thanks to the jacobians provided by the simulators connected on its output, the agent computes a criticality jacobian, giving him an estimation of the variations of the criticalities of its neighborhood. The agent splits its neighbors into two groups, regarding their criticality jacobian:

- the *most critical* group contains the most critical neighbor (i.e. the neighbor with the highest criticality) and all the neighbors whose criticality jacobian has the same sign as the criticality jacobian of the most critical neighbor. This means that if the agent helps its most critical neighbor, its action will also help all the other neighbors of this group.
- the *opposite* group contains all the neighbors whose criticality jacobian are of opposite sign with the criticality jacobian of the most critical neighbor. This means that if the agent helps its most critical neighbor, its action will aslo hurt all the neighbors of this group.

We call *worst neighbor* the most critical neighbor, and *opposite worst neighbor* the most critical neighbor from the opposite group. The goal for the agent is then to help its worst neighbor while preventing its opposite worst neighbor to become the worst neighbor.

The agent observes four indicators in its environment to decide how to adjust its δ and σ:

- the direction of variation of the criticality of its worst neighbor,
- the direction of variation of the criticality of its opposite worst neighbor,
- the direction of variation of the difference between the worst neighbor's criticality and the opposite worst neighbor's criticality,
- the correlation between the sign of σ and the sign of the criticality jacobian of the worst neighbor, which indicates if the action of the agent is currently helping the worst neighbor or the opposite worst neighbor.

Before computing its action, the agent adapts its δ and σ according to a small set of rules conditioned by these four indicators. These rules are presented in table 1.

"Strengthen σ" means its absolute value is increased without changing its sign: if σ is positive then σ is incremented by one (decremented by one otherwise). Likewise, "weaken σ" means its absolute value of σ is decreased without changing its sign: if σ is positive then σ is decremented by one (incremented by one otherwise). Finally, "increase δ" means δ is multiplied by $\alpha > 1$ and "decrease δ" means it is divided by $\beta > 1$. α and β are user-defined parameters.

The idea underlying this set of rules is to decrease as fast as possible the criticality of all the neighbors. The agent seeks to maximize its δ while reducing the difference between the criticality of the worst neighbor and the criticality of the opposite worst neighbor.

The worst case for an agent is when the conditions of line 1 occur: the criticality of both the worst neighbor and the opposite worst neighbor are increasing, the highest criticality faster than the other. In this case, the agent adjusts σ and δ in order to give more help to its worst neighbor and less to the opposite worst neighbor.

The best case is in line 8, when the criticality of both the worst neighbor and the opposite worst neighbor are decreasing, and the highest criticality is decreasing faster. This is an ideal situation if $\sigma = n$ or $\sigma = -n$, meaning that the agent is certain about its current action.

3.4.2 Parameters

There are six parameters for the agents:

- n - defines the boundaries of σ, that varies between $-n$ and n.
- δ_{init} - since δ is adjusted in real time, the system is able to converge for any initial value of δ but a proper δ_{init} may help for a quicker convergence.
- δ_{min} - the minimal value for δ has to be strictly positive, because $\delta_{min} = 0$ often leads to deadlocks.
- δ_{max} - the maximal value for δ is merely a safety guard to avoid too big variations. If $\delta_{min} = \delta_{max}$

#	Variation of the criticality of the worst neighbor	Variation of the criticality of the opposite worst neighbor	Variation of the difference between these two neighbors	Decision of an agent currently helping its worst neighbor	Decision of an agent currently helping its opposite worst neighbor
1	↑	↑	↑	strengthen σ, increase δ	weaken σ, decrease δ
2	↑	↑	↓	decrease δ	weaken σ, decrease δ
3	↑	↓	↑	strengthen σ, increase δ	weaken σ, decrease δ
4	↑	↓	↓	impossible	impossible
5	↓	↑	↑	impossible	impossible
6	↓	↑	↓	strengthen σ	strengthen σ, increase δ
7	↓	↓	↑	strengthen σ, increase δ	strengthen σ, decrease δ
8	↓	↓	↓	strengthen σ	strengthen σ

Table 1: Rules of Adaptation

the system works with a constant δ.

- α - the "increase factor" by which δ is multiplied when it needs to increase,
- β - the "decrease factor" by which δ is divided when it needs to decrease.

α and β both have to be strictly greater than one. They are the only really sensitive parameters as the stabilization of the system depends on them. α should be low comparatively to β. Although a thorough study of their impact is needed, in our experiments we never had good results if β was not at least equals to the double of α. For some experiments, the best results were obtained with β worths 100 times α.

Generally, the lower δ_{min} and δ_{max} the more precise the solution, but at the cost of more timesteps to reach it. A greater β makes δ more frequently close to δ_{min} , hence gaining precision and loosing speed of convergence, and vice versa for α. A big α tends to harm the stability of the system, provoking big oscillations around the solution instead of smoothly converging towards it. Finally, n impacts the inertia of the system: if it is high, an agent will be slower to change direction after having strengthened σ several times consecutively.

Except for δ_{init} all the parameters can be interactively modified at runtime, the system will self-adapt according to the new values.

4 RESULTS

This section shows results of various experiments on generated networks with polynomial nodes. We

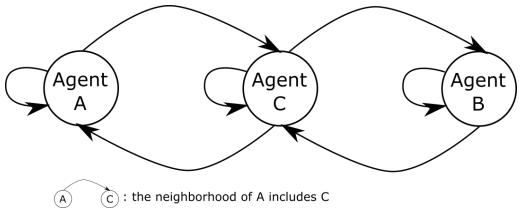

Figure 3: Neighborhood Graph of a 3 Nodes Test Case

used a generator to create random topologies of networks of polynomial nodes. Nodes are of the form of $ax^2 + bx + c$ with random a, b and c. For these experiments, our system is implemented in Java 1.7 and runs on an HP Elitebook laptop with an Intel Core i7 processor and 16GB of RAM. The parameters were set as follows:

- $n = 3$
- $\delta_{init} = 1.0$
- $\delta_{min} = 0.001$
- $\delta_{max} = 1.0$
- $\alpha = 1.2$
- $\beta = 100$

These values were obtained empirically and proved to be efficient for all our experimentations with networks of second degree polynomial nodes, regardless of the topology.

4.1 3 Polynomial Nodes

Our first experiment involves a simple 3 nodes network. Figure 3 shows the neighborhood graph of this

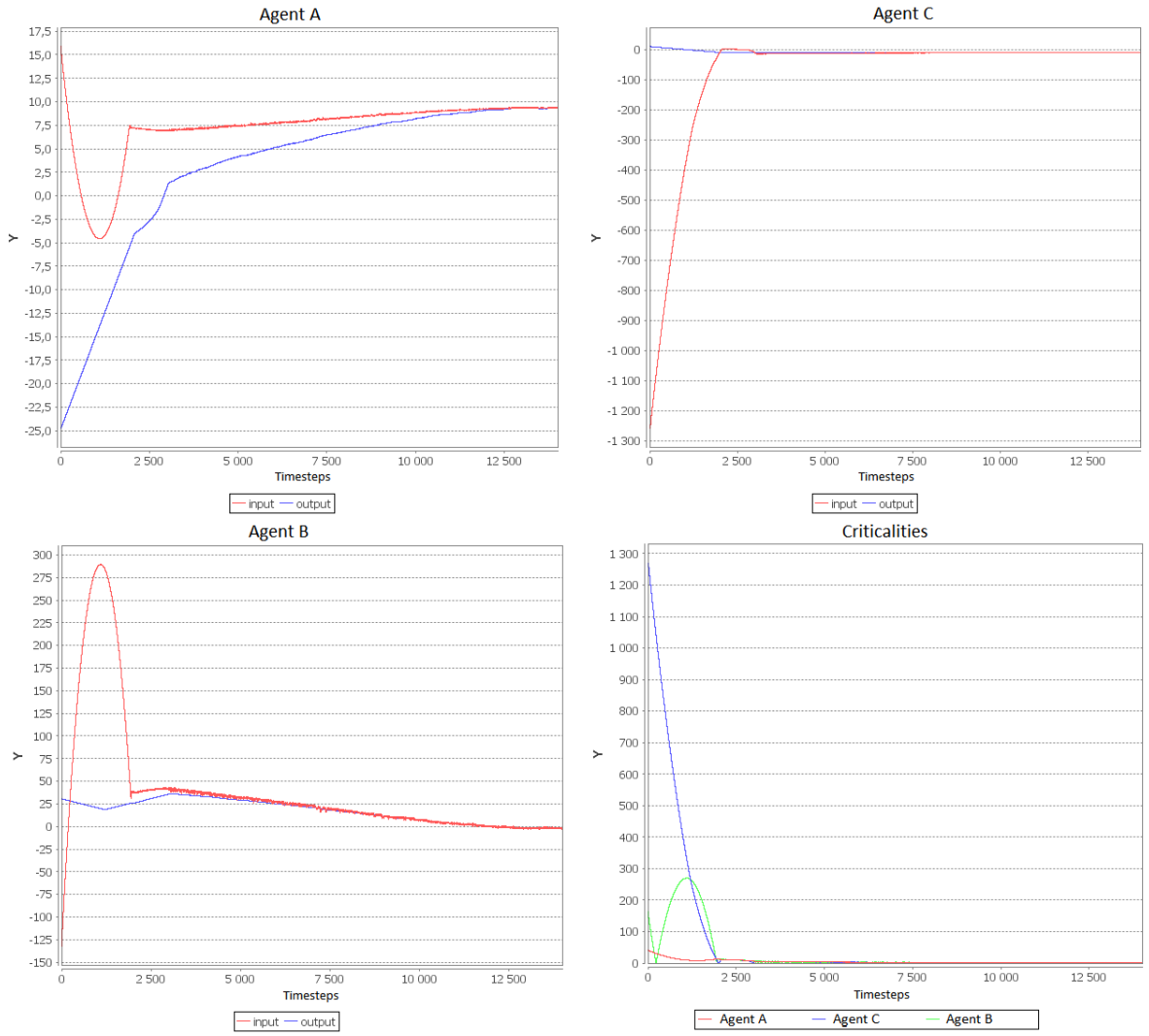

Figure 4: Convergence with a 3 Nodes Network

simple case. It is a representation of the network of influence of the agents. An edge from A to B means that the neighborhood of A includes B, in other words A directly impacts B via a simulator. Here agent A has two neighbors including itself, agent B has two neighbors including itself, and agent C has three neighbors including itself.

Figure 4 shows the input and output of each agent during the resolution, along with their criticalities (the difference between their input and their output). We see that at the beginning, the most critical agent is C. Since it is in the neighborhood of both A and B, these agents tend to help C more, even if it increases their own criticality. Around timestep 1500, agent B is on the verge to become the most critical agent. The agents (including B itself) react and find a way to decrease both C and B's criticalities, at the expense of A. In the end, they all have converged toward a stable state where their criticality is less than δ_{min} (presently 10^{-3}). It took approximatively 10000 timesteps (3 seconds).

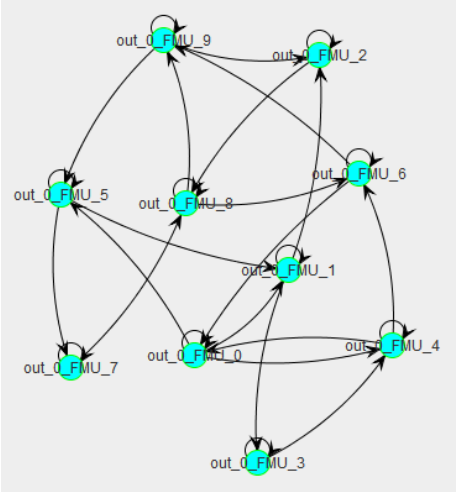

Figure 5: Neighborhood Graph of a 10 Nodes Test Case

4.2 10 Polynomial Nodes

In this experiment, we test our system with a bigger network. Figure 5 shows the neighborhood graph of our MAS for this case. For the sake of concision and readability, we do not show here the input and output curves of each agents, but only their criticalities (6). We can see that the agents immediately find a way to decrease the worst criticalities, but then struggle to make the residual criticalities disappear. However, they end up finding a way to reach a state where each criticality is less than δ_{min}. It took approximatively 14000 timesteps (30 seconds).

With the same parameters, for a network more than three times the size of the previous one, the number of timesteps to converge is only 40% greater than the previous case. Unfortunately, the duration is about 7 times longer.

4.3 100 Polynomial Nodes

This experiments show how the MAS scales up to a network of 100 polynomial nodes. We do not present the neighborhood graph because it is unreadable. Figure 7 shows the criticalities of all agents. We see that the systems converge in about 18500 timesteps. This is only 1.32 times more timesteps, while the network is ten time bigger. However, this time the computation took 90 minutes to complete (180 times longer than the 10 nodes network).

4.4 Discussion

The main result of these experiments is that the number of timesteps to find a solution does not increase with the size of the network. In other words, the system scales up to the size of the network in term of the the total number of actions needed for each agent. However, the duration of the computation increases with the size of the network. This is mainly due to the fact that our prototype is single-threaded: the computation of each node has to be done sequentially. Co-simulations can be distributed, sometimes even physically distributed, enabling to compute each node of the network in parallel. Moreover, our agents take completely local decisions and do not need sophisticated synchronization. This is why our future work will focus on the parallelization of our implementation. This should significantly improve time of computation, making the system also scalable in terms of duration of computation.

5 CONCLUSION AND FUTURE WORKS

Co-initialization, the problem of properly initializing interdependent simulators in a co-simulation is challenging, especially when simulators are numerous. This problem can be reinterpreted as a fixed-point problem, for which current methods suffer from a lack of scalability. This paper presented a multi-agent approach where each output of each simulator is an agent trying to find an initial value for the simulators that are connected to it. Agents seek to help their neighborhood as well as themselves, and together converge towards a stable solution.

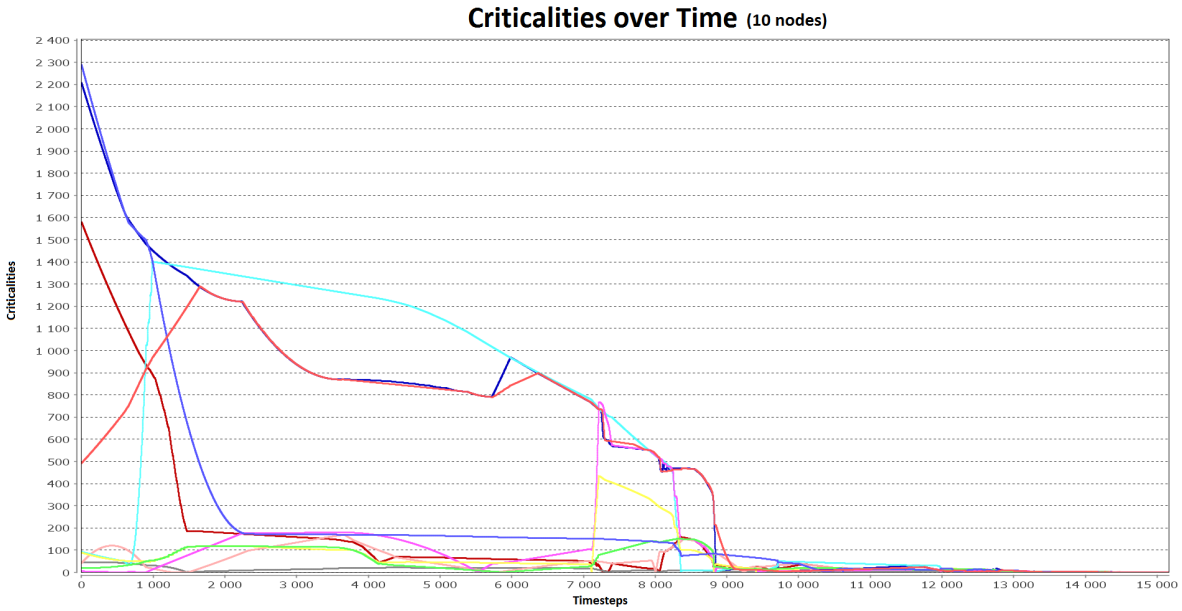

Figure 6: Convergence with a 10 Nodes Network

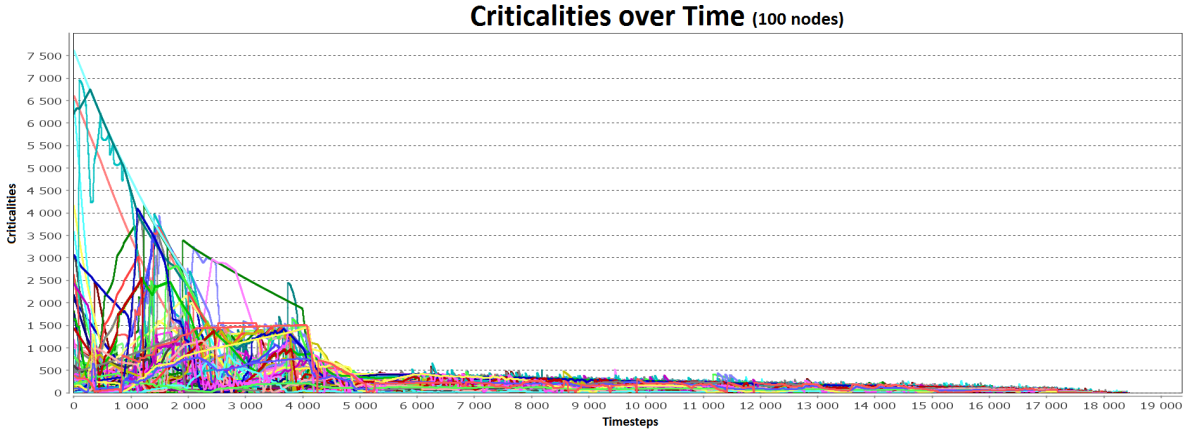

Figure 7: Convergence with a 100 Nodes Network.

Results show that this approach is efficient and scales up to the number of simulators regarding the number of timesteps required to reach a solution. It takes only 1.85 times more timesteps to initialize a network of 100 simulators than a network of 3 simulators. However, our current single-threaded Java implementation does not scale: because their execution is sequential, the higher the number of simulators, the longer a timestep is. Future work will focus on a multi-threaded parallelized implementation to make the system also scalable in terms duration of computation. Our intention is also to get rid of the jacobians. The less the agents need to know about the models, the more generic our approach will be. This could be achieved by enhancing the learning abilities of the agents, so they improve their understanding of the simulators with each action they make.

REFERENCES

Alber, Y., Reich, S., and Yao, J.-C. (2003). Iterative methods for solving fixed-point problems with nonself-mappings in banach spaces. *Abstract and Applied Analysis*, 2003(4):193–216.

Belghache, E., Georgé, J.-P., and Gleizes, M.-P. (2016). Towards an Adaptive Multi-Agent System for Dynamic Big Data Analytics. In *The IEEE International Conference on Cloud and Big Data Computing (CBDCom), Toulouse, France*. IEEE Computer Society - Conference Publishing Services.

Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., et al. (2011). The functional mockup interface for tool independent exchange of simulation models. In *8th International Modelica Conference*, pages 105–114, Dresden.

Bonnans, J.-F., Gilbert, J. C., Lemaréchal, C., and Sagastizábal, C. A. (2013). *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media.

Bonnet, J., Gleizes, M.-P., Kaddoum, E., Rainjonneau, S., and Flandin, G. (2015). Multi-satellite mission planning using a self-adaptive multi-agent system. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Cambridge, MA, USA*, pages 11–20. IEEE Computer Society - Conference Publishing Services.

Border, K. C. (1989). *Fixed point theorems with applications to economics and game theory*. Cambridge University Press.

Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., and Aboulhamid, E. M. (2006). A systemc/simulink cosimulation framework for continuous/discrete-events simulation. In *Behavioral Modeling and Simulation Workshop, Proceedings of the 2006 IEEE International*, pages 1–6.

Costan, A., Gaubert, S., Goubault, E., Martel, M., and Putot, S. (2005). A policy iteration algorithm for computing fixed points in static analysis of programs. In Etessami, K. and Rajamani, S. K., editors, *Computer Aided Verification: 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005. Proceedings*, pages 462–475. Springer Berlin Heidelberg, Berlin, Heidelberg.

Georgé, J.-P., Gleizes, M.-P., and Camps, V. (2011). Cooperation. In Di Marzo Serugendo, G., Gleizes, M.-P., and Karageogos, A., editors, *Self-organising Software*, Natural Computing Series, pages 7–32. Springer Berlin Heidelberg.

Liberatore, V. and Al-Hammouri, A. (2011). Smart grid communication and co-simulation. In *Energytech, 2011 IEEE*, pages 1–5.

Rantrua, A., Maesen, E., Chabrier, S., and Gleizes, M.-P. (2015). Learning Aircraft Behavior from Real Air Traffic. *The Journal of Air Traffic Control*, 57(4):10–14.

Schaefer, B.-J. and Wambach, J. (2005). The phase diagram of the quark–meson model. *Nuclear Physics A*, 757(3):479–492.

Sicklinger, S., Lerch, C., Wüchner, R., and Bletzinger, K.-U. (2015). Fully coupled co-simulation of a wind turbine emergency brake maneuver. *Journal of Wind Engineering and Industrial Aerodynamics*, 144:134–145.

Vaubourg, J., Presse, Y., Camus, B., Bourjot, C., Ciarletta, L., Chevrier, V., Tavella, J.-P., and Morais, H. (2015). Multi-agent multi-model simulation of smart grids in the ms4sg project. In *International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2015)*, pages 240–251, Salamanca, Spain. Springer.

Whitehead, D. (2008). The el farol bar problem revisited: Reinforcement learning in a potential game. *ESE Discussion Papers*, 186.

Yao, Y., Cho, Y. J., and Liou, Y.-C. (2011). Algorithms of common solutions for variational inclusions, mixed equilibrium problems and fixed point problems. *European Journal of Operational Research*, 212(2):242 – 250.