Standardize data synchronization policies for distributed agent-based simulations using proxies

Lucas Grosjean[1,2], Paul Breugnot[1,2], Alexis Drogoul[1], Bénédicte Herrmann[2]

Nghi Quang Huynh[1,3], Christophe Lang[2], Nicolas Marilleau[1,2] and Laurent Philippe[2*†‡]

January 30, 2025

Abstract

Agent-based modeling, or agent-based model (ABM), is a powerful tool that helps researchers understand complex and high-stakes problems, such as the impact of climate change, digital twins, or epidemiology issues. However, as the scale and/or the precision of these models increase, performance problems emerge when running large simulations on standard computers. Agent-based simulation platforms, can no longer run these large-scale models in a reasonable amount of time, or even cannot run them at all. To address these problems, ABM can be distributed using High-Performance Computing (HPC) techniques to divide the workload across multiple processors, speeding up the simulation execution. Distribution bring the need for defining data synchronization protocols between processors, and ways to access agents across processors to properly execute the distributed simulation. To tackle the issues introduced by the distribution of a model, we propose the concept of *proxy*: entities managing interactions between agents across distributed instances of a simulation and controlling access to agent data using *data synchronization policies*. The effectiveness is shown through a case study using the GAMA platform.

Keywords: Data synchronization, Distribution, Simulation, Agent-based modeling

1 Introduction

Agent-based approaches are well suited to represent and understand complex systems. Agent Based Models (ABM) are indeed used in numerous contexts[?] [?] [?] [?] [?], to simulate the behavior of various systems and showcase the emergence of global trends from individuals, by modeling the behavior of individual agents and their interactions.

However, as the scale and/or the precision of these models increase, performance problems emerge when running large simulations on standard computers. Since most agent-based simulation platforms run on these type of computer, the execution of large-scale models is sometimes no longer possible, or not possible within a reasonable time. In this case, the use of high-performance computing (HPC) techniques, by distributing the same simulation across multiple processors that does not shared memory, opens up interesting prospects for overcoming resource constraints and hardware limitations of large-scale simulation. Some HPC Agent-based modeling and simulation platforms (HPC ABMS) emerged [?] [?] [?] [?] [?] [?] [?] [?] [?], proposing frameworks that help users designing distributed ABM and deal with some of the distribution issues. Several challenges however remains open. Thus, despite the intrinsically "distributed" nature of multi-agent systems, these systems describe a single world with synchronized interactions and shared data. Distributed agent-based simulations, for instance, require frequent synchronization operations, such as updating the state of the environment and the agents perceptions, that are not taken into account into the HPC Agent-based simulation platforms, making their distribution still challenging. The distribution of an ABM is therefore neither a trivial task, nor easily generalizable.

*[1]UMI 209, UMMISCO, IRD, Sorbonne Université, Bondy, France

†[2]Université de Franche-Comté, CNRS, institut FEMTO-ST, F-25000 Besançon, France

‡[3]Can Tho University, Can Tho, Vietnam

Some HPC Agent-based modeling and simulation platforms (HPC ABMS) emerged [?] [?] [?] [?] [?] [?] [?] [?] [?], proposing frameworks that help users designing distributed ABM and deal with distribution issues.

A critical problematic in distributed ABM simulation is effectively managing data synchronization among the set of processors. As the agent set is split across the processors, it is often required for an agent to access other agents, located on different processors, to interact with them. While agents can be easily copied to make interactions possible, many issues arise from having multiple copies in the distributed simulation. Data synchronization is needed to ensure that interactions between agents follow rules or policies, keeping the distributed simulation coherent. The main focus of this article is thus on improving data synchronization in distributed simulations.

While data synchronization remains not supported in many HPC ABMS platforms, FPMAS [?] was the first platform to define data synchronization rules for distributed simulation. While their proposition represents a promising start, it lacks a formal frame to define these interactions and the interaction scheme is limited. This paper proposes solutions to address two critical challenges to take better account of data synchronization in distributed agent based simulations: (i) **Standardizing the way to describe ABM distribution**, to describe data synchronization in a standard, and (ii) **Taking user input in the process**, to adapt data synchronization policies to each model with input from the user. The contribution of this article is to introduce two new concepts:

- **Proxies**: entities acting as an agent's unique contact point for other agent interacting with this agent
- **Data synchronization Policy**: a formal language/notation for specifying data synchronization across various levels (simulation, species, and individual) in distributed agent-based simulations

We introduce these two concepts as a way to provide a flexible, customized and standardized management of distributed interactions and to improve data synchronization following the targeted challenges. The GAMA Platform is our testing ground to validate the effectiveness and practicality of our data synchronization approaches and concepts.

To have a better understanding of the possible issues in distributing ABM, existing platforms are reviewed in **??**: fully centralized ABM, distribution-oriented ABM, and ABM that were initially centralized but have been adapted for distribution purposes. Identifying implemented data synchronization solutions on these platforms is the objective of the review. In **??**, we present *proxies* and how they can be used to improve the data synchronization policies in any ABM and how they can be used by users to adapt data synchronization policies to their needs. In **??**, we show the use of proxies with proof-of-concept ABM developed with the GAMA Platform. This section analyses synchronization policies implemented with proxies and their impact on coherence and performance in GAMA simulations. Lastly, we conclude in **??**.

2 Data synchronization in distributed ABM

While the autonomous nature of agents might suggest that distributing ABM simulations is straightforward, this is however not the case. In the now relatively vast field of Agent-Based Modeling and Simulation (ABMS) [?], [?], [?], the development of distributed ABM simulations has not reached the same level of sophistication as centralized modeling. Some platforms are still specialize in distributing ABM like FPMAS [?], PDES-MAS [?], Pandora [?], MASS [?], FAME-core [?], Care HPS [?], FLAME and FLAME GPU [?]. While others, initially focused on centralized environments, have since been able to distribute simulation, e.g. Repast [?] with RepastHPC [?] and Mason [?] with D-MASON [?]. In the following, we analyze the issues arising when synchronizing data in a distributed ABM environments. This analysis builds upon existing work in the field through the previously cited platforms.

2.1 Coherence in distributed simulation

In distributed ABM simulations, keeping data consistent across processors during run time is a challenge for efficient large-scale simulations. Data synchronization ensures that agents access consistent data, from other processors and have access to coherent copy of data. Addressing this issue is crucial to maintain the coherence of large scale simulations.

To illustrate the state of data synchronization in the literature, we use the example of a Prey-Predator model [?] defined as:
"*Preys and Predators are moving around the space. If they have enough energy, Preys and Predators reproduce with other members of their population following a stochastic law (normal distribution). When a Predator is in range of an alive Prey with less than 25 energy, it kills and eats the Prey to consume its energy. A Predator cannot eat a prey with more than 25 energy, as it runs too fast. When a Prey or a Predator runs out of energy, it dies* ". A sequential simulation of this model ensures that each predator always has an exclusive access to preys, so there is no need for explicit concurrency management in this case.

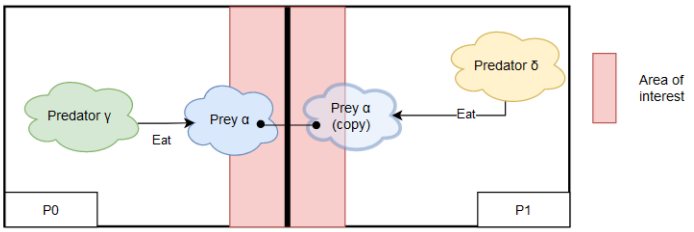


Figure 1: Incoherence in a distributed Prey-Predator model: Prey α can be eated by two Predators at the same time on different processors.

In distributed simulation, preys and predators are executed on different processors, as shown in **??**. In order to allow Predator δ, running on P1, to interact with the Prey α, on P0, a copy of Prey α can be created on P1. Note that the concept of "copy" is known by different names in ABM literature, e.g, "proxy agent", "distant agent", or "copy agent". The introduction of a copy of Prey α can lead to this prey being consumed by two predators at the same time: Predator γ can consume Prey α and Predator δ can consume the copy agent. The incoherence in the distributed model generates biases that could violate rules of the ABM and potentially lead to wrong results compared to the centralized execution of the model. The distributed model architecture and data synchronization mechanisms can introduce inconsistencies in some models, if they do not fit correctly with the needs of the ABM.

2.2 Current proposition in HPC ABMS

HPC ABMS platforms, like RepastHPC [?], Pandora [?], and D-Mason [?] use local and distant agents to access data across processors. They update the copy agent with the most recent data of the local agent between each step. This can however lead to inconsistencies as interactions with copy or local agent are treated identically. FLAME [?] uses a strong conservative messaging board to synchronize the data. It prevents incoherence at the cost of relying on the message board for all interactions. From an user point of view, it might be difficult to understand how data synchronization is performed on these HPC ABMS platforms.

In FPMAS [?], distant agents are created to ensure that every processor has access to agents needed to execute their simulation, updating the set of distant agents at each step. They define "synchronization modes" (see **??**) to ensure data synchronization in interactions between agents and all agents follow the same mode in the simulation. Each mode has a purpose regarding data synchronization. Authors give these definitions [?]:

- *GhostMode:* no restriction on local agent, distant agent cannot be written on, data are retrieved from the previous step only for read on distant agents.
- *GlobalGhostMode:* read always returns the state of the agent from the previous state; no write is allowed on any agent except on the agent itself.
- *HardSyncMode:* allows for all types of interactions between agents, but the consistency of read and write accesses is ensured using concurrent access algorithms. Any read or write operation on a remote agent requires communication with the processor that holds the local copy of that agent.
- *PushGhostMode:* same as *GhostMode*, but writes on distant agents are allowed. At the step end, all agents must retrieve the data from their copies and solve conflicts following an user provided conflict resolution mechanism.
- *GlobalPushGhostMode:* writes on agents are handled as distant writes in *PushGhostMode*, reads on agents return data from the previous step.

Table 1: Synchronization modes defined in FPMAS

Synchronization Mode	*self*	*local*		*distant*	
	write	read	write	read	write
GhostMode	T	T	T	T-1	×
GlobalGhostMode	T+1	T-1	×	T-1	×
HardSyncMode	T	T	T	T	T
PushGhostMode	T	T	T	T-1	T+1
PushGlobalGhostMode	T+1	T-1	T+1	T-1	T+1

Depending on the ABM needs, the appropriate synchronization mode can be selected to cover different range of situations. In **??** a strong synchronization mode ("HardSyncMode") is appropriate. For ABM like the Flocking model a less restricting synchronization mode ("GhostMode") that requires less synchronization might be better. This work shows that the global behavior of a distributed ABM depends a lot on the choice of the synchronization mode.

2.3 Data Synchronization Policies

Synchronization modes are a first step to accurate and coherent distributed simulations. On the other hand, defining a global synchronization mode for all the agent is not flexible, as different agent behaviors may imply different types of interactions, and may lead to performance issues. For this reason, we introduce Data Synchronization Policy (or DSP) and we state that every agent need to have a different DSP, depending on its needs. Let us look at the previously defined Prey-Predator ABM example. The critical scenario requiring a strong DSP is when a predator attempt to access a remotely copied prey from another processor, i.e. when a *write* action is performed on a copy prey. The predator does not need strong synchronization since its behavior and interactions on its data happen without conflict. Thus, a solution with customised DSP for each agent or species of agents would be more suitable for distributed ABM. Since only the modelers possess the in-depth knowledge of their ABM critical interactions, their input is crucial to select the right DSP for each agent type. Moving forward, we need to break free from the limitation of a single global data synchronization mode to control synchronization at different levels.

Overcoming the DSP customization challenge requires the implementation of standardized and entirely manageable agent interaction mechanisms. We hence have to manage the way agents are reached to introduce a data synchronization layer between interacting agents.

2.4 Standardized synchronized interactions between agents

Distributing an ABM is a complex and time-consuming process, which can deter people from using distributed ABM architectures. Most of the current architectures to distribute ABM force the modelers to modify the base code of their ABM to incorporate all the distribution related functions: communication between processors, partitioning of the workload, and data synchronization. Having a clear distinction between the ABM and its distribution follows the principle of separation of concerns, giving the modelers more ease in maintaining the project with two clearly identified sides: the ABM definition on one side and the distribution functions on the other. To this end, we identify key points to improve the data synchronization aspect of HPC ABMS:

- **Standardize interaction between agents**: agent interactions imply synchronizations, they must follow a well-defined standard that can describe most agent interactions in a simple way. It should allow any kind of interaction happening in the ABM to be translated and executed properly in a distributed context.
- **DSP customization**: HPC ABMS needs to be able to cover a large range of different DSP to match what modelers need in data synchronization. Data synchronization should be tailored to the specific needs of each ABM. As each species of agent has different needs, we must be able to define a DSP for each of them to improve the efficiency of the distributed simulation.

Those improvements are linked to the challenges previously stated in the introduction: (i) **Defining a standard way to describe the distribution of an ABM** and (ii) **Taking user's input in the distribution process**.

Overcoming the DSP customization challenge requires the implementation of a standardized and entirely manageable agent interaction mechanisms. We hence have to manage the way agents are reached to introduce a data synchronization layer between interacting agents.

3 Proxy: standardize interaction between agents

To introduce a data synchronization layer for mediating agent interactions, we use **proxies**, a versatile concept enabling customization of DSP. This standardized and transparent approach leverages modelers' knowledge of simulated agent behavior to tailor synchronization within distributed simulations.

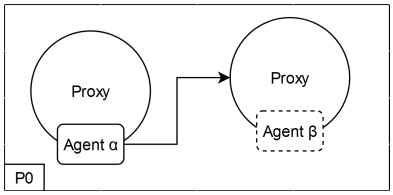


Figure 2: When Agent α interacts with Agent β, it must go through Agent β's proxy first

In ABM context, a proxy can be defined as an *"agent unique handle point for other agent interacting with this agent"*. In the interaction shown in **??**, Agent α accesses unknowingly Agent β through its proxy. Every agent in the distributed simulation is assigned a proxy when created. Proxies ensure transparency in the interactions by hiding the process of reaching an agent and may be used to introduce control over the remote agent data. This approach was already introduced in [**?**] but is generalized here.

Algorithm 1 *Proxy* transparently indirecting to agent attributes

```
function getAttribute(attributeName)
    return agent.getAttribute(attributeName)
function setAttribute(attributeName, value)
    return agent.setAttribute(attributeName, value)
```

?? shows the basic implementation of a proxy. It simply implements the accessors of the agent attributes to act as a layer between interacting agents. It may be completed with any rules before and after the access to the agent attributes. For instance, rules to manage concurrency or data synchronization can be added in this layer.

3.1 Data synchronization policies with proxies

DSP specify how the attributes of an agent can be accessed by other agents depending on the location of the agent (local or distant) and the type of access (read or write). The synchronization modes outlined by [**?**] defines how agents access the data of other agents in a distributed simulation. We adapt the idea of synchronization mode and improve it by combining it with proxies. A DSP is introduced in the proxy of an agent. It controls who can read (access) and modify (write) the attributes of this agent. The agent itself does not perform any action related to this control. An agent reaching another agent has to follow the DSP of the targeted agent. **??** shows how the DSP is set on the proxy of Agent β. In this example, the used DSP is "HardSyncMode", as introduced in **??**. Access to Agent β attributes is now determined by its DSP, meaning that when Agent α tries to access Agent β, the proxy of Agent β, the interaction follows its rule.

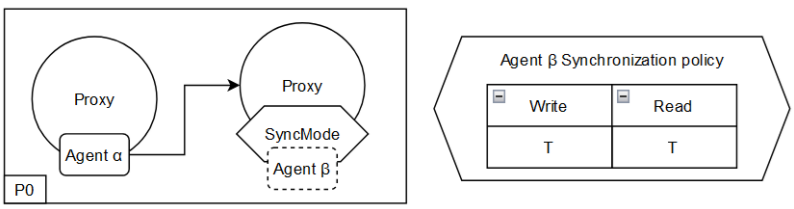


Figure 3: When Agent α reach Agent β, it must respect the DSP of Agent β to ensure the coherence of Agent β attributes.

?? shows a possible implementation of a proxy, including a DSP. Whenever an agent tries to reach

another agent data, startSync() and endSync() are called. These functions control how data access happens and are defined according to the current DSP of the proxy, startSync() and endSync() specific behavior depends on the DSP used. These functions can be customized to create a specific data access policy according to the agent need.

Algorithm 2 *Proxy* implementation with DSP inclusion

```
function getAttribute(attributeName)
    startSync()
    value = agent.getAttribute(attributeName)
    endSync()
    return value
function setAttribute(attributeName, value)
    startSync()
    agent.setAttribute(attributeName, value)
    endSync()
```

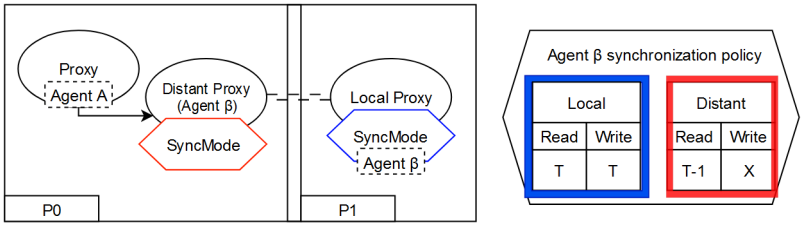


Figure 4: Agent α interacts with a distant proxy of Agent β on P0. The interaction is following the DSP of the distant proxy of Agent β instead of the DSP of Agent β's local proxy on P1.

?? shows how, depending on the state of an agent (local or distant), the data synchronization is different. In this figure, we use a DSP that allows only write and read operations on the local proxy agent and read operations on the distant proxy. No write operation is possible on a distant proxy with this DSP ("GhostMode" in FPMAS).

Our approach of defining DSP at the proxy level instead of simulation level allows the full customization of synchronization for any given agent in the distributed simulation. The approach allows to define a custom DSP for each agent according to its behavior and the opportunity to define three distinct customization levels:

- **Simulation level mode:** DSP can be applied to any agent of the simulation (approach originally defined in FPMAS platform).
- **Species level mode:** DSP can be applied for each type of agent depending on the behaviors and the needs of the species.
- **Individuals level mode:** DSP can be applied for individual agent depending on its behavior.

Note that the data synchronization requirement of the most restrictive agent determines the DSP at both the Simulation and Species levels. Specification of DSP allows to accurately define and describe the synchronization algorithms, chosen accordingly to the case study. Current data synchronization in HPC ABMS does not permit customization from the simulation level to the individual level. As far as our knowledge goes, there is no Domain-Specific Language (DSL) [**?**] to specify data synchronization in HPC ABMS platforms. Our approach promotes a DSL dedicated to the specification of a DSP according to the three previously defined levels using this notation:

$$\begin{aligned}\textbf{DSP} = \textbf{LevelMode[AP,}\\ \textbf{Local(Read: ST;Write: ST),}\\ \textbf{Distant(Read: ST;Write: ST)]}\end{aligned}$$

Where LevelMode $\in$ [Simulation, Species, Individuals], AP (Activation Predicate) can be any predicate, ST (Synch Temporality) $\in$ [..., T-1, T, T+1, ...] or X when the operation is not permitted.

We illustrate this notation on the **??**, Agent β needs two distinct kinds of proxies: a local proxy for local Agent β on Processor 1, with a policy qualified by $Local(Read : T; Write : T)$, a distant proxy for the distant Agent β on Processor 0 with a policy qualified by $Distant(Read : T - 1; Write : X)$. The DSL for Agent β can now be fully defined as follows:

Individuals[Agent β, Local(Read: T;Write: T,

Distant(Read: T-1;Write: X)]

This notation is the root of a DSL to describe the distribution of an ABM. It is a first step toward the promotion of a DSL for distributing ABM, opening a new perspective to distinguish *the thematic ABM* (representing the studied case) from *the distribution model* (specifying the distribution of the thematic model).

3.2 Illustration of DSP with the Prey-Predator model

To illustrate the practical use of DSP in a real case scenario, we describe the possible DSP defintions that can be used for the defined Prey-Predator model.

The model defines two species with differents behaviors. (i) Preys are shared and indivisible resources for predators; a strong DSP should be associated to preys to ensure competition constraints when the simulation is distributed. (ii) Predators have independent life cycle; they die only if they run out of energy, and they do not require specific synchronization. **??** shows a distributed example of the model. The environment is divided into two subspaces with an overlapping zone (OLZ) [**?**], and any agent located in this OLZ is mirrored on the other processor, thus requiring the definition of a DSP.

Table 2: Data synchronization for Prey-Predator model.

		Data Synchronization Policy			
Level mode	Concerned agent	Local		Distant	
		Read	Write	Read	Write
Simulation level	Any agents	T	T	T	T
Species level	Prey	T	T	T	T
	Predator	T	T	T-1	X
Individual level	Prey with energy < 25	T	T	T	T
	Prey with energy > 25	T	T	T-1	X
	Predator	T	T	T-1	X
	Possible other DSP	?	?	?	?

?? shows all the DSP that can be used for the Prey-Predator model. Any agent can choose to use the Simulation level DSP, any agent of a given species can choose the Species level DSP and any agent can choose a fitting Individual level DSP. An agent can only use one DSP at a time but can change DSP over time if needed. For a better understanding, let us examine the data presented in the table.

At *Simulation level*, the most constraining agents species in the model are preys. More specifically, preys that can be eaten by predators as specified by the rules of the model. This DSP can be assigned to any agent. It is defined as:

Simulation[Local(Read: T;Write: T),

Distant(Read: T;Write: T)]

At *Species level*, preys and predators have to follow respectively the DSP of their species, for Prey:

Species[Prey,Local(Read: T;Write: T),

Distant(Read: T;Write: T)]

and for Predator:

Species[Predator,Local(Read: T;Write: T),

Distant(Read: T-1;Write: X)]

At *Individual level*, preys could be divided into two sets depending on their state: (i) preys with less than 25 energy (ii) preys with more than 25 energy; predators have a unique state.

The preys with less than 25 energy require concurrent write operations to be managed, hence the following DSP:

Individuals[Prey, Energy $<$ 25,
Local(Read: T;Write: T),
Distant(Read: T;Write: T)]

Since their data are replicated across multiple processors and that they can be consumed by predators, it raises the risk of concurrent write conflicts (see **??**). We adopt the name "DSP_HardSyncMode" for this DSP, aligning with the FPMAS definition in **??**.

Preys with more than 25 energy do not require the same DSP as they cannot be eaten by predators, we can then set a less restricting DSP:

Individuals[Prey, Energy $>$ 25,
Local(Read: T;Write: T),
Distant(Read: T-1;Write: X)]

We adopt the name "DSP_GhostMode" for this DSP, aligning with the FPMAS definition in **??**. Predators do not require any change from the DSP defined in Species level as there is no difference between predators at individual level. The three DSP (two preys and one predator) are assigned automatically to agents depending on their state.

We have presented the functioning of proxies and DSP. In the following, we depict how proxies and DSP make communication transparent to all agents.

4 Case study with the GAMA Platform

We use the GAMA Platform [**?**] to demonstrate the use of proxies and DSP in a distributed context. This platform is an easy-to-use, open-source modeling and simulation environment for spatially explicit ABM. Despite being useful in many domains, GAMA does not provide any tool to distribute ABM across several processors. Being an active open-source platform makes GAMA a solid candidate to implement our approach using proxies and DSP. We use this platform to demonstrate the use of proxies and DSP using a distributed prey-predator model. Building upon the proven performance of the FPMAS platform [**?**], our research will mainly focus on the coherence of data in a distributed context. We developed solutions [1] to distribute instances of a model on multiple computers or cores. Note that these solutions are simple show-cases. We present concrete scenarios in which proxies help us solving synchronization issues in an ABM distribution.

4.1 Proof of concept for DSP_HardSyncMode

We present here the result of our implementation of DSP_HardSyncMode in GAMA through a modified Prey-Predator model shown in **??**. It illustrates how the DSP_HardSyncMode is working in this situation. Now, when a predator attacks a prey, the energy level of the later decreases by 5. We use this attack mechanism only in this example. Prey0 is executed on P0 with 200 energy and is associated with DSP_HardSyncMode, and copied on P1 as Prey0_COPY. With DSP_HardSyncMode, every time Prey0_COPY is attacked, the change to the energy is reported to the original Prey0 agent on P0.

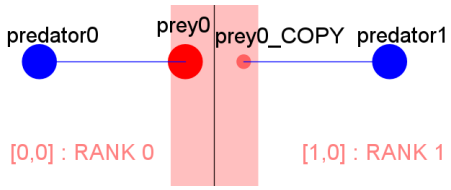
?? is the graph of Prey0 energy over time. From it, we see that Prey0 is attacked by two predators, starting from Step 11 (the arbitrary step in which we make Prey0 available on P1). At Step 21, Prey0 has been killed by either Predator1 or Predator0. It implies that DSP_HardSyncMode is working as intended, as attacks from Predator1 are well reported on the value of Prey0 energy.

4.2 Tailoring DSP: a case-by-case approach

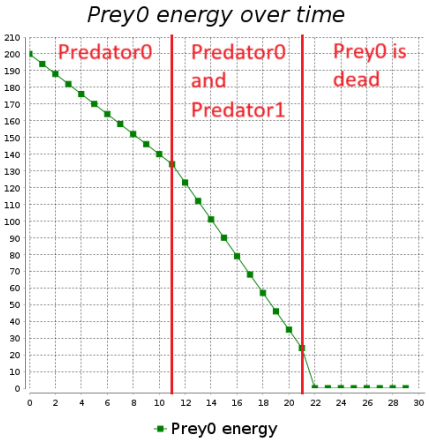
In this second case, we use the Prey-Predator model shown in **??**, using the same situation as in **??**. In this model, Prey0 and Prey1 are shared between P0 and P1, bringing concurrent access. We show that with proxies and DSP, we can tailor the DSP of all agents depending on their needs while keeping the model coherent. **??** shows 20 steps of the model execution.

Prey0 is assigned to DSP_HardSyncMode and Prey1 to DSP_GhostMode. Although Prey0 has an energy level above 25, we chose to set its DSP to DSP_HardSyncMode. This let the Prey0 energy decreases over time to reach the 25 energy threshold five steps later. With this setup, we ensure that

[1] `https://github.com/LucasGrjs/Proxies_DSP`

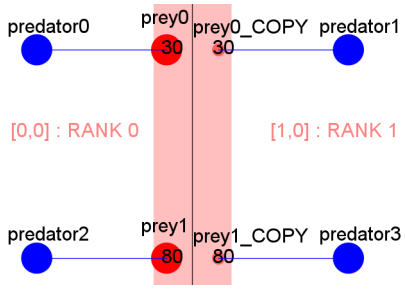


(a) Prey-Predator GAMA model. Prey0 is executed on P0 and copied on P1.

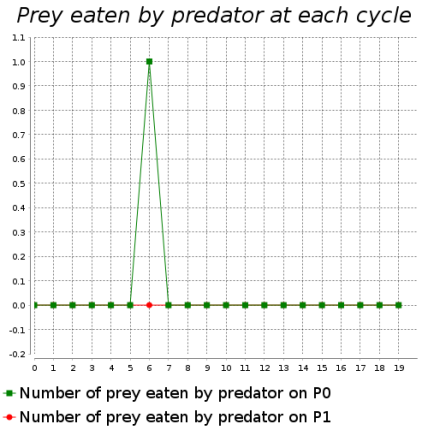


(b) Prey0 energy over time

Figure 5: GAMA model proving the effectiveness of DSP_HardSyncMode in forwarding change made on prey0_COPY to the prey0



(a) GAMA model representing the issue raised in **??**. Prey0 and Prey1 are executed on P0 and copied on P1. Prey0 have 30 energy, Prey1 have 80 energy. We assigned DSP_GhostMode to Prey0 and DSP_HardSyncMode to Prey1.



(b) Result of **??**. Number of prey eaten on P0 and P1 at each step

Figure 6: GAMA model showing how we are able to lower the restrictive level of Prey1 DSP by using different DSP depending on individuals properties

Predator0 and Predator1 can compete to eat Prey0 fairly. Prey1 has 80 energy and then cannot be eaten by predators before the end of the model execution.

?? shows the number of preys eaten at each step on each processor over the execution duration of our model. It shows that Predator0 and Predator1 were competing for Prey0 in Step 6 and that Predator0 eats Prey0. Predator0 has however a significant advantage over Predator1: it is located on the same processor as Prey0, while Predator1 is on a different one. Communications between P1 and P0 introduce delays, which can affect the order in which requests for Prey0 are processed. This delay can give Predator0 a crucial advantage in eating Prey0 over Predator1. We experimented on the same model, this time by having the original Prey0 agent located on P1 and then sent on P0. Predator1 won Prey0 over Predator0 in that case.

This example does, in fact, demonstrate that once Predator0 consumes Prey0 on P0, it cannot be accessed or eaten again on any other processor. This implies that the issue of concurrent accesses between processors for this agent has likely been solved using proxies and DSP. While the whole situation has maintained coherence, we were able to lower the restrictive level of Prey1 DSP using a tailored DSP matching its needs. Using proxies and DSP correctly can improve the way data are synchronized in a distributed ABM.

4.3 DSP and coherence: concurrency access and change forwarding

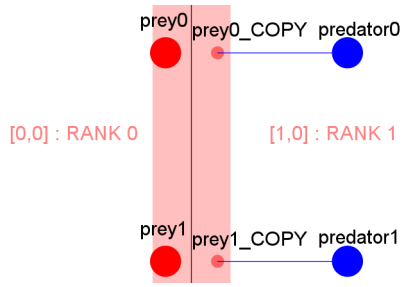
A second use of DSP in the distribution process of an ABM is to protect the coherence of the distributed ABM. For the sake of clarity in **??**, we have simplified the Prey-Predator model by removing the energy threshold. This induces a clear advantage to the predator which is able to kill the prey regardless of the prey energy reserves.

?? shows a simple situation where Prey0 and Prey1 are located on P0 and Predator0 and Predator1 are located on P1. We assigned DSP_GhostMode to Prey0 and DSP_HardSyncMode to Prey1. The difference between the two DSP is that DSP_HardSyncMode report changes made on Prey1_COPY to Prey1 located on P0. This means that, when Predator1 eats Prey1_COPY on P1, Prey1 is effectively killed on P0. Changes to the copy would not be reported to the original agent if DSP_GhostMode was used. Here is a breakdown of what happened to Prey0 and Prey1 over time in this model:

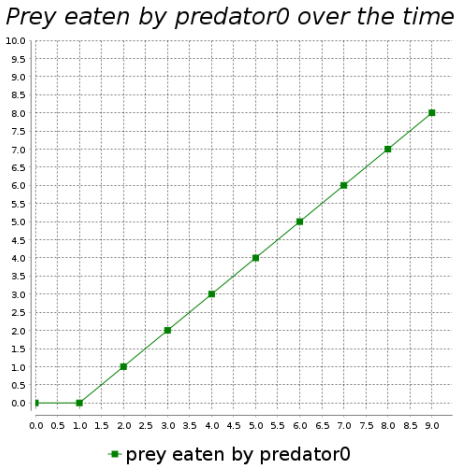
- *Before Step 0:* Prey0 and Prey1 are located on the OLZ between P0 and P1, we copy the preys on P1 as they do not have a copy on this processor.
- *Step 0:* Prey0_COPY and Prey1_COPY are killed by the predators, only Prey1_COPY forward the change to its original agent, Prey1 on P0 is killed. From this point Prey1 is dead in the simulation. Prey0 is still alive on P0.
- *Between Step 0 and Step 1:* Prey0 is still located in the OLZ, we copy Prey0 on P1 as Prey0 does not have a copy on P1, overriding the information that Prey0 has been killed at step 0.
- *Step 1:* Prey0_COPY is killed again.
- *Between Step 1 and Step 2:* Prey0 is copied again on P1.
- *Step 2:* Same as Step 1.

Our experiment, showed in **??**, demonstrates that concurrent access to agent data is not the only data synchronization issue. Managing how changes are reflected back to the original agent is required to ensure the coherence of the simulation. **??** illustrates the incoherence introduced when changes are not forwarded from a copy to the original agent on a critical aspect of the agent (here the living state of the agent). It shows that Predator0 will be able to kill Prey0_COPY at every step of the simulation. DSP definition has then to integrate the notion of forwarding changes from copies to the original agent to fully cover the data synchronization issue in distributed ABM. Some early concept have been theoretically introduced by FPMAS [**?**] with "PushGhostMode" to solve the forwarding of information from copies to the original agent: all the copies of an agent would have to forward all their changes to the original agent between step, the original agent would have to merge all the states of its possible copies into one coherent state. This approach places a significant burden on the modeler, responsible for defining the merging state function.

The main point of this example is to show that DSP are not only meant to solve concurrent access over a resource in a distributed simulation, but also ensure the coherence of the simulation by synchronizing changes made to any copy agent with the original agent.



(a) GAMA model of Prey-Predator showcasing the use of DSP to protect the coherence of the distributed simulation. We assigned DSP_GhostMode to Prey0 and DSP_HardSyncMode to Prey1



(b) Result of **??** highlighting the coherence problem resulting from the non-forwarding changes made on a copy to the original agent on a critical aspect

Figure 7: GAMA model showcasing the impact of not forwarding changes from copies to original agent on the coherence of the distributed simulation

5 Conclusion

In this article, we presented new concepts aimed at distributing ABM: *proxy* and *DSP*. Proxy is a flexible and transparent concept for managing interactions between agents across distributed instances of a simulation. Our use of proxies is to allow the usage of DSP more effectively, but we can think of other uses for proxies in centralized ABM. In fact, proxies can be useful for data processing, ensuring the correctness of interactions. Being close to the agent, they can efficiently process relevant data and validate interactions before they are passed to the agent. This might reduce the risk of errors or inconsistencies in the overall simulation by giving more insight into the execution of a centralized or distributed model. We introduced DSP using proxies as a mean to define data synchronization for distributed simulation. Our approach of assigning agent-level DSP control through proxies offers control over three level of data synchronization (simulation, species and level) in distributed ABM. Modelers can tailor data flow for each species and individual within their simulation, leveraging their expertise to achieve the desired level of coherence. Our successful implementation and testing within GAMA validates the effectiveness of this approach, paving the way for more nuanced controls over data synchronization in future distributed ABM development.

In the introduction of the article, we defined two critical challenges for improving the way ABM are distributed:

- **Standardizing way to describe the distribution of ABM:** we have introduced a novel notation to define DSP in distributed simulations, species within those simulations, and individual entities. This DSP notation effectively addresses the challenge of data synchronization at various levels.
- **Taking user's input in the process:** with proxies and DSP, we have achieved customization of data synchronization for any individual in the simulation. User can then create and use new DSP adapted to the ABM needs enabling the definition of DSP based on individual agent states.

Future works will also focus on defining new DSL to describe different parts of a distributed ABM. Ultimately, aiming to create a complete DSL that can describe the entire distribution of an ABM. This DSL will be the basis to address other critical challenges:

- **Distributing an ABM out-of-the-box:** the goal is to make it easy to distribute an ABM without making changes to the base code of the model. Meaning that we should be able to outsource distribution functions outside of the model definition.
- **Adapting the architecture to the user's skill level:** adjust the distribution architecture to modelers with diverse technical expertise. Our goal is to develop customizable, ready-to-use solutions for distributing ABM, allowing for ongoing improvement based on user needs.