Accepted Manuscript

Care HPS: A high performance simulation tool for parallel and distributed agent-based modeling

Francisco Borges, Albert Gutierrez-Milla, Emilio Luque, Remo Suppi

PII: S0167-739X(16)30275-8
DOI: http://dx.doi.org/10.1016/j.future.2016.08.015
Reference: FUTURE 3138

To appear in: *Future Generation Computer Systems*

Received date: 22 January 2016
Revised date: 1 August 2016
Accepted date: 16 August 2016

Please cite this article as: F. Borges, A. Gutierrez-Milla, E. Luque, R. Suppi, Care HPS: A high performance simulation tool for parallel and distributed agent-based modeling, *Future Generation Computer Systems* (2016), http://dx.doi.org/10.1016/j.future.2016.08.015

Care HPS: A High Performance Simulation Tool for Parallel and Distributed Agent-Based Modeling

Francisco Borges[a,*], Albert Gutierrez-Milla[a], Emilio Luque[a], Remo Suppi[a]

[a] *Universitat Autònoma de Barcelona*
Department Computer Architecture & Operating Systems. School of Engineering. Campus Bellaterra. Cerdanyola del Vallès. Barcelona. Spain. 08193

Abstract

Parallel and distributed simulation is a powerful tool for developing complex agent-based simulation. Complex simulations require parallel and distributed high performance computing solutions. It is necessary because their sequential solutions are not able to give answers in a feasible total execution time. Therefore, for the advance of computing science, it is important that High Performance Computing (HPC) techniques and solutions be proposed and studied. In literature, we can find some agent-based modeling and simulation tools that use HPC. However, none of these tools are designed to enable the HPC expert to be able to propose new techniques and solutions without great effort. In this paper, we introduce Care High Performance Simulation (HPS), which is a scientific instrument that enables researchers to: 1) develop techniques and solutions of high performance distributed simulations for agent-based models; and, 2) study, design and implement complex agent-based models that require HPC solutions. Care HPS was designed to easily and quickly develop new agent-based models. It was also designed to extend and implement new solutions for the main issues of parallel and distributed solutions such as: synchronization, communication, load and computing balancing, and partitioning algorithms. We conducted some experiments with the aim of showing the completeness and functionality of Care HPS. As a result, we show that Care HPS can be used as a scientific instrument for the advance of the agent-based parallel and distributed simulations field.

Keywords: Agent-Based Model; Agent-Based Modeling and Simulation; High Performance Simulation; High Performance Computing; Parallel and Distributed Simulation;

1. Introduction

Agent-Based Modeling and Simulation (ABMS) is an approach for modeling and simulating complex systems. ABMS can model a complex system if it

*Corresponding author
Email address: francisco.borges@caos.uab.cat (Francisco Borges)

can be represented as a collection of agents. These agents are programmed to follow some rules of behavior [1]. In addition, the interactions of the agents might generate a collective behavior. Thus, it allows scientists to gain new knowledge and hypotheses in their research field. For that, a dynamic process is required that simulates the agent interaction over time. ABMS relates to several disciplines: military, biology, social science, economics, etc. As advantages of ABMS, we can cite:

1. Determining the implications of different hypotheses starting from a verbal argumentation [2]. Thus, what-if experimentation can be conducted with only parameter and configuration adjustments.
2. ABMS can execute in a parallelized way [2], therefore this enables the execution of complex models.
3. ABMS allows the researcher to produce emergent phenomena.
4. ABMS is a suitable tool to study complex systems.
5. ABMS enables us to simulate problems where analytical solutions are not possible.

Therefore, scientists can reach conclusions and gain knowledge about the system under using ABMS. However this is only possible if these simulations offer realistic results. It means simulations whose results are valid in reality, and which can also be used for prediction or to explain some phenomenon. Therefore, these simulations require reliable results through statistical approaches. Moreover, they have a high computational complexity because thousands of agents model them, their higher level of parameters and their rules complexity. Thus, this kind of simulation requires a long execution time. Consequently, parallel and distributed simulations can be a solution to solve these simulations, since parallel and distributed simulations can take advantage of the powerful architecture available nowadays. Thus, it is possible to create complex models. This enables us to achieve simulation results that are closer to reality. In this context, we find two different user profiles. However, they are complementary and necessary for developing these solutions:

1. application area researchers have a lot of knowledge of their specific research field, yet they are not able to deal with the complex computing solutions, which are needed to execute large and complex simulations.
2. HPC experts are able to provide scalable and efficient complex simulations, but they do not have a deep knowledge of the application areas.

In this paper, we introduce a tool called Care HPS, whose main objective is to be a scientific instrument tool for ABMS in a parallel and distributed architecture. Its sub-objectives are: 1) to enable applications area researchers to study, design and implement complex agent-based models that require an HPC solution; and 2) to enable HPC experts to develop techniques and solutions of high performance distributed simulation for agent-based models. The remainder of the paper is organized as follows. We dedicate Section 2 to presenting the history, motivation and justification of Care HPS. In Section 3, we present related works where we show the results of an extensive search in the literature

for tools in the scope of Care HPS. Care HPS is presented in Section 4, and here we focus on the theoretical support and present its architecture. In Section 5, we compare the main tools that we have found in the literature review technically with Care HPS. Some examples and explanations of implementation are given in Section 6. This section takes into account the points of view of the application area researchers and HPC experts. Then, we check the completeness and functionality of our tool through the experiments presented in Section 7; and, lastly, our conclusions and final considerations are shown in Section 8.

2. Why Care HPS? History, motivation and justification

Over the last few years, our group has been researching HPC in parallel and distributed simulation. From the beginning, our main aim has been to get more complex simulations. The motivation is because researchers in application areas require more realistic results coming from complex simulations. The problem is that complex simulations demand a lot of computing resources. Thus, we have to study how to provide these simulations in a more efficient and scalable way. Therefore, we have developed an agent-based modeling and simulation tool where we have applied our techniques and solutions proposed for HPC.

The initial ideas about Care HPS emerged in 2013. However, we cannot disregard all of the know-how developed and acquired by research group before the conception of Care HPS. Almost all of this knowledge has given support to and has been implemented in Care HPS. Therefore, the history of Care HPS has two periods. The first between 2004 and 2013, when many contributions in HPC for agent-based models were proposed. And the second after 2013, when the idea of the current conception of Care HPS emerged and its implementation started.

The contributions of our research group in HPC to ABMS from 2004 until 2013 can be summarized in the following way. The first version of the simulator dates from 2004. Firstly, we implemented the first fish schooling agent-based model version with MPI. The distribution of data throughout the architecture was implemented with a partial partitioning. The synchronization of the distributed processes was controlled using conservative protocol. After that, the fish schooling model was improved and other HPC features were included. Next, in 2005/2006 [3, 4, 5], the simulator started to support a higher number of agents after improvements. Also, a halo exchange routine for communication between agents near partition borders was implemented. In 2008, some optimizations were implemented to improve scalability and communication [6]. Next, in 2009, we represented the behavior of fish agents using a Fuzzy Logic[7]. In 2010, we added obstacles in order to improve the environment representation. Also, we simulated different species in the same simulation space [8]. In 2011, the research group developed a clustering algorithm partitioning [9]. Then, in 2012, the research group implemented a load balancing routine for agents [10]. And lastly, in 2013, we developed two new MPI communication approaches (BSP and asynchronous)[11].

From this point on, all new implementations in the new simulator (named Care HPS) have considered two basic object-oriented programming concepts: extension of functionality and reusability. The biggest change in the original simulator was uncoupling the agent layer. After that, we redesigned all important HPC features. Then, all previous HPC features implemented were also decoupled and included in Care HPS. This process was always cyclic and progressive. Also, we added new HPC features in Care HPS, described as follows. In 2014, we improved the clustering algorithm[9] to support the memory shared paradigm [12]. In the same year, we adapted a statistical method defined by[13]. This method identifies the best run length of the simulation[14]. And finally, in 2015, we implemented a new agent-based model and partitioning algorithm[15]. For that, we used the existing agent layer and features to extend partitioning algorithms. Also, we added other features: serializable agents, an environment layer and a model layer.

As we said, scalability and efficiency are the most important issues for providing a complex simulation. Therefore, from the point of view of scalability and efficiency in the ABMS context, we have noted that the HPC approaches used to reach a good scalability and efficiency depend on the characteristics of the model. To illustrate this, we will give two forms of agent-based model interactions among agents and environment. There are models in which agents do not modify the environment, for example, fish schooling. In this model, the fish swim to avoid obstacles, but no environmental resources are consumed. In this context, no resources consumed means that there are no changes in environmental properties. Therefore, the synchronization among the distributed processes is not needed to update the environment. However, the dynamic of the environment in an Ant Colony is completely different. In this model, the stacks of food are distributed among the processes. Therefore, the quantity of food available must be synchronized in order for all processes to know when the simulation has finished. Thus, the ABMS cannot deal with synchronization using the same synchronization approach. Otherwise, the fish schooling model can spend time carrying out an unnecessary procedure. Consequently, this will increase the total execution time of the simulation. We understand that the characteristics of the agent-based model and HPC techniques must fit together in order to extract the best solution performance. The synchronization is just one example. Another example would be whether the simulation type is communication or computing demand. An agent-based model that has a high communication volume requires a different distribution of data compared with agent-based models that are computing demand. This reasoning is applicable for other important HPC and ABMS issues.

Care HPS basically comes from the answers to two questions. The first question is: how can we make a generalization of our HPC techniques for ABMS? The second one is: how can we develop other agent-based models rapidly and test our HPC techniques with other agent-based model rules? Why are these questions important? In computer science, as well as in general science, the generalization of results is a fundamental scientific aspect. The research must give results that can be applied in other contexts. In this way, the research can

have a bigger impact on and importance to the studied field. We have observed that our previous contributions had an impact in a specific context. In this case, a fish schooling model or similar agent-based models.

Applying these solutions to other contexts would require a huge effort in programming, since other models with different computing characteristics must be implemented. Therefore, we hope that the next step in our research will provide solutions for a greater range of models. To reach this aim, the ABMS tools need to enable the user to easily add new models in order to test/evaluate a new proposed HPC feature, as well as enabling the user to easily extend/develop features for a specific model.

Figure 1 shows two hypothetical scenarios with the objective of presenting the applicability of Care HPS. In Scenario 1, we have implemented a load balancing (LB) algorithm. Care HPS enables us to use different agent-based models (ABM_A, ABM_B and ABM_C), which have different computing requirements in order to analyze the load balancing algorithm proposed. These scenarios enable the HPC expert to evaluate the efficiency of the load balancing with different agent-based models. In the other scenario, we want to analyze which load balancing algorithm (LB_A, LB_B and LB_C) will fit best with the agent-based model's requirements. This scenario enables the application area researcher to identify which LB algorithm offers the best execution times without a great effort in implementation.

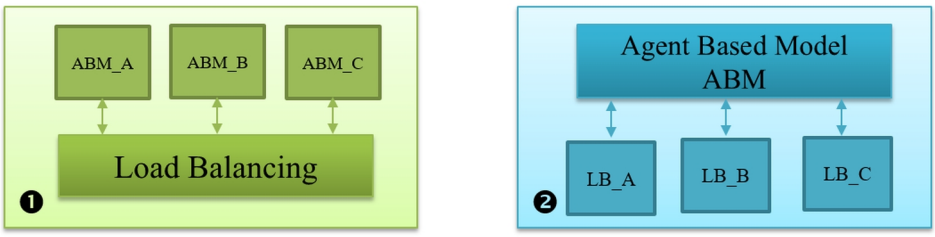


Figure 1: Different scenarios of research.

Until the ends of 2013, the architecture of our simulator was overly coupled: the fish schooling model, the kernel of simulation and the HPC solutions were all connected. Adding new techniques, such as a new partitioning algorithm, required a lot of programming effort. We were spending more time programming than doing science.

In the literature, we find several tools in which ABMS has been very well tested in specific projects. These tools also present a high maturation level. As examples: Flame in EURACE project [16, 17], which simulates the European economy. Netlogo has a huge community and a large library of sample models. Pandora is being used in SimulPast, which is an archaeological project [18, 19, 20]. Finally, Repast HPC has presented excellent weak scalability on Argonne National Laboratorys IBM Blue Gene [21]. However, most of these tools address and support HPC features using generic approaches. In addition, they do not allow the user to extend the HPC features without a huge programming time

investment. Thus, Care HPS was born as an instrument that supports our HPC experimentation for ABMS. The basic premises are:

1. To easily support the extension and reuse of our HPC techniques proposed for ABMS.
2. To easily support the addition of new HPC features.
3. To support the development of new agent-based models faster.
4. To enable the user with less programming know-how to develop parallel and distributed simulation without previous knowledge of HPC.

3. Related work

Based on the literature, we have nominally listed several tools. In addition, we checked which ones have some features and purposes of support simulations. We have found around 80 different tools referenced. After that, we then selected tools that fit with the criteria:

1. supports parallel and/or distributed simulation, and/or
2. support for agent-based modeling and simulation.

The tools found are: Ascape[22], D-Mason[23], EcoLab[24], Flame[25], MobiDyc[26], Netlogo[27], Pandora[28], Repast HPC[29], and Swarm[30, 31]. Ascape is a framework for developing general-purpose agent-based models [22]. D-Mason[23] is a parallel extension of MASON[32] library for writing and running ABMS. EcoLab [24] was initially projected to support an abstract ecology model [33]. It supports the partitioning of agents over processors and tolerance failure support. EcoLab has a component that allows reflection to be added to C++ language. Flame (FLexible Agent-based Modelling Environment) is a code-generated tool that allows the user to define its agent-based model. Flame automatically generates C code optimized for efficient parallel processing [25]. MobiDyc aims to develop individual-based modeling in the fields of ecology and biology. MobiDyc enables the user to develop their model through simple primitives [26]. Netlogo [27] was born as an educational project. Netlogo has a friendly GUI and an easy programming API for developing new agent-based models. Netlogo does not support HPC features. However, it is a powerful ABMS tool with a huge user community. Pandora[28] has been used in several archaeology projects. Pandora is an ABMS tool of the Barcelona Supercomputer Center. Repast HPC is an agent-based modeling and simulation toolkit [29]. Repast HPC implements the main concepts of Repast Simphony. Repast HPC has support for parallel distributed environments. It was designed with the goal of obtaining extreme scalability for the TOP500-class supercomputer[34]. Swarm was the first ABMS software development environment created by the Santa Fe Institute. The agents are organized through a collection called "swarm" that has a scheduled event for these agents [31]. Swarm [31] does not have any parallel capabilities [35]. Swarm uses Objective C.

As presented, we can find several ABMS tools that support several issues, features, and contexts. A few of them use HPC to execute their models.

However, none of these tools are designed with the aim of being a scientific instrument that facilitates the research of HPC for agent-based models that demand high performance solutions. Of course, many of these tools are open source and researchers can add other features. However, the time invested in programming is too high. Furthermore, it requires high level knowledge of the tool and programming. Finally, Railsback et al. [36] offer us interesting recommendations for the development of ABMS tools. We implemented many of them in Care HPS and others will compose our future work.

4. Care HPS architecture

Care HPS supports a distributed memory system which allows for the distribution of the simulation space over several processors using MPI. In addition, it may use the shared memory paradigm to compute agents which are located at the same processor through OpenMP in order to increase the efficiency of the execution of a model.

Care HPS was designed with a focus on good object-oriented programming practices. It is composed of several layers and components (see Figure 2) coded in C++ language. The design of Care HPS permits all of the complexity of the HPC to remain hidden for the application area researchers. This means that application area researchers do not need to worry about the following: how to distribute the agent in different cores; how to balance the agents over processes; or how to synchronize the processes. These are few examples of HPC issues that are hidden by Care HPS through the use of its layers and components. This enables users to use HPC without much programming effort or know-how.

In addition, the design of Care HPS enables HPC experts to extend and reuse code. This is possible because Care HPS uses several design patterns [37]: Composite, Facade, Factory method, Strategy, Singleton. These design patterns allow the HPC users to improve their HPC solutions, as well as to easily enable the addition of new features and resources. Other papers mention the importance of using a design pattern in ABMS architecture [38]. Papers [23],[28], [29] have used design patterns to extend and implement their features. For more details, definitions and examples of design patterns see [37].

We will explain the most important components of Care HPS layer by layer in the following subsections.

4.1. Agent-based model layer

The agent-based model layer is composed of the following components: Agent, Serializable, Environment, Random number generation (RNG) and Model. Application area researchers must use this layer in order to carry out their research.

The Agent component has a built-in *agent* class that has the main fields required to represent agents of an agent-based model in a Euclidean space 2D and 3D. The fields are:

1. *position* - indicates the position of the agent in space.
2. *velocity* - indicates the direction of the agent.

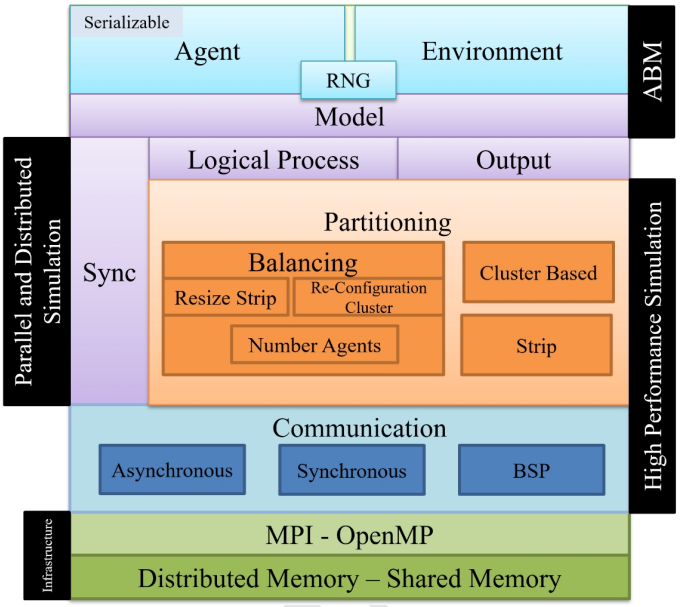


Figure 2: System architecture.

3. *covering radius* - indicates the radius of the interaction of the agent with the environment and other agents.
4. *member* - used to serialize the state of the agent (serializable component).
5. *ID* - identifies the agent uniquely.
6. *random_move* - indicates if the agent moves randomly.
7. *alive* - indicates if the agent is alive. Sometimes the agent is no longer useful along the simulation. Therefore, it can be excluded from the simulation space.

Care HPS is prepared to execute any agent upon the condition that the agent implements or extends the built-in class: *agent*. This is the most important class that has the responsibility of implementing the behavior of agents. The problem modeled sometimes requires additional fields that can be included in the user's agent classes (concrete class of agent), as well as the methods of the concrete agents which implement local rules and behavior. Therefore, the user has to implement these methods and call them inside the basic method called *update_agent*. The user must call these methods in an appropriate order using logical and/or conditional statements.

Care HPS executes the *update_agent* method many times by from the beginning to the end of simulation. Furthermore, Care HPS does not implement a schedule of events. Rather, all agents are added in a bucket. At each simulation step, the system calls the *update_agent* method of each agent. The agents are accessed randomly in the bucket to avoid the first mover advantage. The algorithm used for the randomization of the bucket has a linear algorithm complexity. The

group behavior emerges as a consequence of the agents' interactions occurring in this loop execution. The agent can interact with other agents or environment inside of its covering radius. This value can be changed in the execution time in order to implement different behaviors of agents. Other solutions, such as [23], use this same concept, known as area of interest.

Another important feature of the *agent* class is the ability to serialize the state of the agent. The *agent* class uses the Serializable component through a field called *member*. All required states are saved and loaded in the *member* data structure. This data structure supports the main primitive type data of C++ language. The user has the responsibility of saving and loading the field in the same order. The base fields of the agent class have their states saved automatically by Care HPS. Only the fields created at concrete agent classes - the user's class that inherits from the agent class - need to be specified by the user. By definition, agents have state and behavior. However, the serialization of the state of the agent is a required feature in parallel and distributed simulation. This is required because the agents might migrate from one process to another. In the migration processes, the agents are deleted from the processes and created in other processes. Therefore, the agents' states must be preserved in this operation. The migration processes are common in an agent-based model distributed solution. It can occur as a consequence of a local role, a motion in space, or because of a load balance or partitioning routines. The Serializable feature enables Care HPS to know what the new fields are. Consequently, Serializable component can use this information in the communication process. Therefore, the application area researchers do not need to implement any communication routine. This is provided by the communication component (see 4.3).

The Environment component enables the modeling of an environment in an ABMS. To create an environment in Care HPS, it is necessary to extend the two classes which compose the Environment component: *environment* and *object*. The environment is the space where the agent has "life". The *environment* class holds the agents that interact with the environment and with other agents. Also, *the environment* class will be the container of possible objects created inside of it. The objects of the environment might be created or extended from the *object* classes. Care HPS enables the user to define objects with the most appropriate representation for the problem. With the aim of implementing this feature, we are using a math approach to model and represent the objects of the environment. This approach enables us to dynamically create a representation of any type of obstacle or resource, such as buildings, food, paths, etc. For example, we can use the *linear_plan* class to simulate a coast, valley and rocks in the sea. Equation 1 gives a specific example where it intends to represent a rock:

$$2 * x - 13 * y + 5 \tag{1}$$

In this example, Care HPS checks the intersection between the plan and the vector equation of the line. This equation represents the agent's position and

direction. The vector equation of the line is defined by the equation:

$$P = A + tv \tag{2}$$

Where P indicates the next position of the agent; A represents the actual position; v the direction; and t is a constant which represents the maximum distance that the agent can move per time step. The equation of the line indicates if there is a point (P) that is the root of the equation that represents the object of the environment. Therefore, the code sums the current agent position (A) with the next agent position (tv), where t is defined in this case as covering radius.

In summary, this approach checks if the new point (next agent position) is a root of the object equation. If so, an agent's behavior should be executed in order to model the interaction with the environment. The user can create as many objects inside the environment as is required. This described procedure will be executed for each object within the environment. We chose to use a math approach instead of using agents to represent the objects in an environment. The math approach supports a wide range of representations of space. Several math expressions can be added in the same environment class as objects. Thus, many combinations can create the required environment. This approach easily supports the interaction between agents and objects of the environment. It occurs with math operations of intersection between an agent's vector and the equation that represents the object. In addition, the Environment component enables the user to replicate the environment in all processes. This means that all processes have a full copy of the environment. This option should be used when the objects do not undergo changes during the simulation. Therefore, it avoids communication with other processes.

Care HPS provides a class for random number generation (RNG). Some simulations work with stochastic data that is used to create the initial input of a simulation as well as defining random values for agent behavior, such as velocity or position. The user can create normal, exponential, gamma and uniform distributions. In addition, random numbers can be obtained within a range. RNG is a useful feature that is also available in other ABMs such as Flame [25], Netlogo [27], D-Mason [23], Pandora [28], and Repast HPC [29].

The last component of the Agent-based model layer is the Model component. The application area researchers will use this component to represent their problem. The user will specify agents, environment and other components. As well as this, the user will choose the HPC strategies that the model will use. The Model component is composed of two classes: *model* and *pds*. The concrete class *model* requires the user to implement some methods in order to define:

1. the size of the agent - the communication layer (see 4.3) uses this information to execute the pack and unpack.
2. the agent that will be created and simulated.
3. the type of object that the environment will contain
4. the type of environment.

The concrete class *pds* configures and defines the characteristics of simulation. As an example of the most important methods, we can cite: *getModel* - create or get the model; *initialize_agent* - initialize the simulation agents; *prepare* - setup and define behavior and parameters of the simulation; *simulate* - execute the simulation steps; *wrapp_to_synch* - defines a method for the synchronization.

Moreover, the concrete class *pds* represents the parallel and distributed simulation of the specific *model*. *pds* class uses *model* class to create the *agent* and *environment*. The *pds* class is an implementation of the Abstract Factory design pattern [37]. Therefore, with a concrete class *pds*, it is possible to create products related to the simulation, such as model, partitioning, load balancing, environment, agent, etc. This class has many factory methods that must be overridden by the user. This defines which object will be created in execution time. These factory methods are the key to hiding the complexity of HPC. In addition, these factory methods interface the Agent-based model layer with the Parallel and distributed simulation layer.

4.2. Parallel and distributed simulation layer

The Parallel and distributed simulation (P&DS) layer implements the issues related to parallel and distributed simulation. Executing simulations in a parallel and distributed way requires diverse techniques such as: synchronization, partitioning, and load and computing balance. This layer has the following components: Logical process, Sync, Partitioning, Balancing, and Output. Application area researchers do not need to have deep knowledge of this layer. Since the use of the components is available through the factory methods. On the other hand, the HPC expert can use this layer in order to execute their HPC experimentation and also propose new strategies and solutions.

Care HPS supports a distributed memory system. The Logical process component creates the distributed processes that communicate by message passing. Additionally, this component controls the simulations running in different processes. Care HPS was designed for simulations that advance in steps. Therefore, the synchronization between processes is necessary before the next step is taken. The Sync component implements the synchronization features. The synchronization can be used for many purposes: maintaining the simulation coherence, executing load balance, executing compute balance, or re-partitioning. Care HPS enables the application area researchers to choose which synchronization method to call, as well as to define which reduced operation will be executed. The result of this method is sent to all distributed processes. This result might be used in order to make decisions, such as finalizing the simulation, executing a load balancing, etc. For this, the application area researchers must implement the *wrapp_to_synch* callback method. Repast HPC [29] implements a similar callback method that executes a global data collection.

The design of Care HPS allows other synchronization approaches to be implemented and proposed. Care HPS implements a conservative [39] protocol in synchronization. In this implementation, each process has its own bucket with agents. The processes are blocked until all of the distributed buckets have finished the execution of their agents. In contrast with other ABMS tools, such

as Repast HPC, we do not implement a scheduler to iterate the simulation forward. In Repast HPC, the user has to add events to occur at a specific tick in the scheduler [29]. The execution approach of Care HPS is quite simple. We consider the agents to be autonomous and therefore know what they must do and when to do it.

Data partitioning is the most important aspect of parallel and distributed solutions. The distribution of data over a distributed architecture has a strong correlation with communication, load and compute balancing. Care HPS provides three partitioning strategies: cluster-based partitioning [9], strip partitioning with a pure MPI solution [15] and the new hybrid strip partitioning proposed in Subsection 6.2. Moreover, Care HPS enables HPC experts to extend new partitioning algorithms.

The cluster-based partitioning approach can be used for agent-based models, where its agents' interactions can be grouped into clusters [40]. This approach uses an unsupervised classification of patterns. The criteria used in this method is based on Voronoi diagrams and covering radius[9].

The strip partitioning approach fits better with agent-based models which have a spatially dependent problem. The strips are created in accordance with the number of MPI processes. This partitioning algorithm chooses the best partition configuration. It considers the number of processes and the distribution of objects in the environment. This strategy avoids objects being shared by processes [15].

The users choose the partitioning approach through the factory method of *model* class. The partitioning has two main methods public to the user: *distribute* and *execute*. The first method distributes the agents over the distributed architecture and the second one executes the agents' behavior, which calls the *update_agent* method (see Subsection 4.1). The distribution of the agents and their behavior's execution are completely transparent for the user. Moreover, other functions such as the migration of agents, point-point communication, and loading the state of agents are executed by partitioning algorithms.

The partitioning algorithm distributes the data throughout the architecture. In an agent-based model parallel and distributed solution, the algorithm of partitioning must distribute the agents equally among the processes. A well-done partitioning algorithm should ensure three issues. First, a load balance among the processes. Second, that all agents have similar computing and communication throughout the whole simulation. And lastly, decreasing the agent migrations. However, this situation is not common. Generally, the agents are executing different rules in the same moment. In addition, the migration between processes is a common procedure in distributed simulations, especially in Euclidean space simulation, where the agents might move constantly. Therefore, during the simulation, two situations can occur: 1) some processes can contain more agents than others; and/or 2) some processes can spend more time computing than others. The first situation is due to the migration of agents between processes. And the second one is because the distribution of agents does not take into account the computing differences between agents. These situations have a direct impact on the total execution time, especially,

if the simulations use conservative protocol. The reason is because the faster processes wait for the slower processes to finish before moving forward to the next step in the simulation.

Parallel and distribution simulations can become inefficient if they are not accomplished with a loading and computing balancing. The balancing approach can be categorized into spatial dependence problems and non-spatial dependence problems.

First of all, let's discuss when the agent-based model is a spatial dependence problem. The movement of an agent from one partition to another is quite complex when a loading or computing balance action occurs. These solutions generally already define which partition will execute that part of the problem. All of the simulation space is divided among the partitions. Each core handles one or more partitions. Therefore, if the agent is moved from one partition to another, then it will act in a different part of the simulation space. Redefining which core will execute the partition is not a problem. However, all environment, objects and agents allocated at the partition will also be reallocated and executed by this core. Therefore, increasing or decreasing the granularity of the simulation space might be a good strategy for spatial dependence problems.

On the other hand, loading or computing balancing routine is more simple when the problem is not spatially dependent. For this type of problem, the balancing policies can take into account just the objective functions, whose results can direct actions such as assigning partition/agents for specific cores, resizing partitions, etc.

However, in both categories, the loading and computing balancing actions can depend on the modeled problem. Therefore, an extension of the available solution of the balancing approach can be required. The distribution of data has attributes that should be considered in order to develop a feasible balance routine. Thus, the algorithms of load and computing balancing should act on partitioning of data algorithms. Currently, three balancing policies are implemented: re-configuration of cluster [10], re-size of the strip and number of agents. In addition, new strategies can be implemented and extended. In Care HPS, the algorithms of load and computing balancing are designed as a composition of the partitioning of data. This means that the partitioning of data might have many load and computing balancing algorithms. This way, they can be invoked in the synchronization to deal with imbalance occurrences. As an example, consider the cluster-based partitioning that the re-configuration cluster strategy uses [10]. First of all, the strategy detects an imbalance, verifying the number of agents per cluster. Secondly, the strategy checks if this number of agents per cluster is within the thresholds. It detects underused or overloaded resources. The next step is the re-configuration of the cluster with the adjustment of (1) the workload to the mean of agents per core; and, (2) core that will execute the clusters. The application area researchers set the load and computing balance strategy to use after they choose the partitioning approach. Care HPS will automatically call all balance strategies defined for the partitioning in synchronization.

The HPC expert needs to implement the subclass of the balancing interface and implement the method to execute if they want to propose a new balance

strategy.

4.3. High performance simulation layer

The High performance simulation (HPS) layer is composed of the Communication component. As can be observed in Figure 2, the P&DS and HPS layers share some components. This occurs due to the overlapped techniques used for developing the two layers. The interactions with the P&DS layer and the Communication component of the HPS layer occur via the partitioning algorithm in order to execute the communication processes for the whole simulation. Care HPS communication processes occur basically when:

- there is a migration of agents among processes.
- the processes exchange information for the synchronization process.
- initialization of the simulation.
- when the load balancing, compute balancing and partitioning are executed.

There is no direct communication between the agents through MPI routines. However, there is local communication between agents through memory copy or shared memory.

Only the partitioning algorithms interface with the Communication component. This component uses the pack and unpack pattern in order to send and receive data. Care HPS has implemented three communication patterns: asynchronous, synchronous and BSP [11]. The MPI communication routines used in Care HPS depend on the partitioning used. For example, the cluster-based partitioning needs a broadcast communication due to the fact that agent migrates to an unknown process. In the strip partitioning approach, the communication occurs just among the neighboring processes. In this strategy, the agents can only migrate to partition neighbors. Therefore, only MPI_Send and MPI_Recv routines are required.

The Serializable component (see 4.1) is the key to automating the communication process for the application area researchers. The communication process works with the agent superclass. Therefore, it does not know about the specialized fields. A similar approach is found in the Repast HPC [29]. In Repast HPC [29], the user must specify each field of the agent class, providing the serialization code, executing the pack, and executing the unpack. However, Care HPS solution is simpler. In Care HPS, the user needs only to handle the Serializable component. After informing which fields need to be saved, then the save and load methods are called at the appropriate moment. We delegate to the application area researchers to inform which fields are new and which ones must be saved. This is compensated by the fact that the application area researchers do not manipulate the Communication component.

5. Comparison: Care HPS and other ABMS tools

With Care HPS features in mind, we will present the main differences among similar tools found. We are concerned with ABMS tools that have support for parallel and/or distributed executions and with similar features to Care HPS. In the bibliographical survey carried out in Related Work 3, we found the following tools: Flame [25], D-Mason [23], Pandora [28], and Repast HPC [29]. For the main features of Care HPS, see Table 1, and we will explain how each ABMS tool addresses these issues. This comparison will focus on how each tool addresses a feature rather than strengths and weakness. In spite of supporting the parallel and distributed execution, each tool has a different domain. In addition, it was designed for a different purpose than Care HPS.

5.1. Agent

Care HPS has a built-in agent class that represents agents of an agent-based model in a Euclidean space 2D and 3D. The user can extend classes from this base class in order to model their problem. In Flame, the agents are represented by X-machine [41], and its behavior is defined by a directed acyclic graph. D-Mason implements the agent extending an abstract class called *RemoteAgent*. This class enables the agent to be serializable. Pandora uses the *Agent* class that encapsulates any entity of the model. The user must define the *updateState* method to specify the state, decision-making processes and behavior of the agent. Repast HPC represents its agents by C++ class. The user has to implements their agent class and behavior by methods in this class [29]. Moreover, all agents must implement the *Agent* interface [29]. All these tools allow for the extension of a base class (or proprietary unit: X-machine), where the user can implement the agent's behavior.

5.2. Environment

The interactions among agents occur inside the environment. Depending on the problem, the environment can be static or dynamic. There are no changes in environment properties when the environment is static. In a dynamic environment, the interactions among agents or the interactions of agents with the environment can change its properties.

Care HPS represents the environment through two classes: *environment* and *object*. The *environment* class is a container for the *object*. The interaction between agents and the objects of the environment is implemented using a math approach. Flame uses an X-machine in order to represent an environment. D-Mason has built-in classes that represent a space at 2D and 3D. Its environment can be represented as Grid or Continuous space [42, 43]. Pandora extends classes to represent an environment. These classes are abstract classes and must be implemented by the user. Repast HPC implements the environment with Context and Projection concepts [29]. A context contains a set of agents and has Projections associated with it [29]. A projection is a way of structuring agent relationships [44]. Some of the tools listed here use their own agent classes in order to represent the environment. The advantage of this approach

is that the user can handle the whole simulation composed of objects. So, the communication between an agent and the environment is through the messages of the objects.

As presented in Section 4, we chose a math approach to represent objects in an environment. The reasons are:

- we believe that representing the environment and its objects as agents do not strictly follow the agent definition in an ABMS context.
- the math approach allows for easier implementation and interactions with objects in a Euclidean space.
- this approach enables the representation of a huge number of environments and objects.

5.3. Synchronization

The synchronization operation in distributed processes is generally a critical operation. This is a block operation, thus the faster processes have to wait for the slower processes. A possible technique to minimize this problem is to use overlap routines. But this type of solution cannot always be used because of the data dependence.

We think that the synchronization can be adapted in accordance with the problem. Therefore, it is important to have a public interface for extension by users. Care HPS enables the user to define which method will be used for synchronization. A callback function calls this method and uses its returned value to make a decision. Flame controls the synchronization through the message boards. This strategy ensures that all agents can see the same set of messages [45]. D-Mason implements a local step-by-step synchronization of each region. Each region is synchronized with its neighborhood before each simulation phase [23]. Pandora defines the synchronization process through a scheduling system. The user can define its own scheduler solution [28]. Repast HPC handles any necessary synchronization using a discrete event scheduler [29].

5.4. Serializable

Serializable is a required feature in parallel and distributed simulation because the agents might migrate from one process to another. For basic fields of agents, the state is saved automatically by Care HPS. Additional fields must be added by the user in the field *member* of the agent. The serializable feature can be implemented by memory in the X-machine in Flame. D-Mason uses the publish-subscribe design pattern to propagate agent state information [23]. In this way, the agent is able to update and maintain its state at each simulation step. Pandora saves the states of the agent in a file format. This file format supports high performance computing applications [28]. Repast HPC supports the serializable feature through serialization. The user has to provide a serialization code. This code must extract the agent state, package it for transfer and then unpack the transferred package [29].

5.5. Partitioning

Data partitioning is the most important aspect of parallel and distributed solutions. The distribution of data over a distributed architecture has a strong correlation with communication, load and compute balance. Care HPS provides three partitioning strategies: cluster-based partitioning [9], pure MPI strip partitioning [15] and hybrid strip partitioning proposed in this paper (see 6.2). Beyond that, Care HPS enables HPC experts to extend new partitioning algorithms such as the hybrid strip partitioning. Flame implements two static partitioning approaches: Separator Partitioning and Round Robin Partitioning [46]. Separator Partitioning is a kind of geometric partitioning. Flame takes into account the position of the agents in order to define the partition that will be allocated [46]. In Round Robin Partitioning, Flame distributes each agent at a time to each partition in turn. D-Mason implements a space partitioning. The space simulated is partitioned into regions by the master. Each region has a set of agents and each region is assigned to a worker [23]. Pandora implements a spatial partitioning. Each node receives a part of the simulated environment and the agents located within its boundaries. Each part of the environment is divided into four different sections. Then, the sections are executed sequentially [28]. Repast HPC implements the partitioning through a projection concept. The user can choose from among three types of projection: grid, continuous space and a network [29].

5.6. Computing and loading Balancing

The loading and computing balancing in Care HPS is dependent on the partitioning strategy. Care HPS enables one or more balancing policies to be implemented for each partitioning strategy. Currently, three load balance policies are implemented, and the user can also propose and extend new balancing algorithms. Flame considers that the communication among the agents has a more significant impact on the total execution time than the light-weight computational nature of many agent types [46]. Flame does not implement a load balance routine directly. It should be reached using a partitioning strategy, as presented in [35]. In a recent paper, Kang and colleagues [47] proposed an extension of Flame with support to load balance through migration processes. D-Mason deals with load balancing through the granularity of the simulated space's decomposition [23]. The user must define the granularity of the partition, as well as the number of regions to be assigned to each worker. In Repast HPC, the load balance can be reached through projection parameters. Some examples can be found in [44].

5.7. Communication

The communication process is the key for performance in parallel and distributed solutions. Care HPS communication occurs when:

- there is a migration of agents between processes.
- the processes exchange information for the synchronization process.

- initialization of simulation.
- when the load balancing, compute balancing and partitioning are executed.

In Flame, the agent communicates with the environment and other agents through a Message board [45]. The communication is all-to-all through message boards, and there is no direct agent-to-agent communication [45]. D-Mason uses the publish-subscribe design pattern in order to propagate agent state information. Each worker receives relevant messages through a multicast channel [23]. Pandora supports OpenMP and MPI APIs in its communication routines. Local communication uses OpenMP inside a given interaction range [28]. On the other hand, neighbor nodes receive border information by MPI every time a step is executed [28]. Repast HPC automates much of its communication processes, but, the user must provide a serialization code to pack and unpack the agent information [29].

5.8. Model

High performance solutions require parameterizations that will be somehow translated into code. For an application area research user, the meaning of the parameters can be complicated. Therefore, this increases the complexity for the user. On the other hand, decreasing this complexity for the user means hiding the technical details. Consequently, this will assume default values that probably will not bring the best performance results. As examples: number of threads for memory shared solution, the affinity of processes, size of messages, and many others. Therefore, the bigger trade-off of ABMS tools resides in modeling the facilities for users that are not experts in computing. A Care HPS user must define its model by implementing and overriding some classes using the C++ language specification. Care HPS was designed with the purpose of hiding all the possible complexity of HPC techniques from the application research user. The Flame model specification is defined in XMML, which is Flame's agent specification language based on the XML standard [46]. The XMML must represent the model. The user must also provide information such as: the agent properties; a list of functions with current state and end state; user defined data types; possible inputs; and/or conditional statements. Each function must be implemented by the user in a C program source. A model in D-Mason is defined by using the subclass of Mason's model class called *SimState* [32]. It contains a discrete-event schedule, a random number generator, and fields. Pandora has two abstract classes (*World* and *Agent*) to represent the content of any model [28]. Repast HPC does not provide any interface or extend any class to represent a model [29]. The user has to create and initialize the Repast HPC components to model their problem.

The literature has many ABMS tools with different approaches, solutions, implementations and domains. Each tool achieves varying levels of performance and facilities for different types of users. After this comparison, we can summarize that Care HPS differs from other ABMS tools in the following point:

- Care HPS was designed to be a tool to do science with agent-based models that require high performance simulations as the main objective, for both the application area user and for the HPC expert user.

Table 1: Comparative table of Care HPS with other ABM tools

ABM Tool	How does it addresses							
	agent	environment	sync	serialization	partitioning	balancing	communication	modelling of a problem
Care HPS	Extends a class	Extends a class	Callback method	Using built-in class	Built-in or Extended	Built-in or Extended	Built-in or Extended	Extends a class
Flame	X-machine	X-machine	Message board	X-machine	Built-in	Feature implemented indirectly through of other feature.	Message board	XMML
D-Mason	Extends a class	Built-in	Built-in	Feature implemented indirectly through of other feature.	Built-in	User must provide the solution	Publish-Subscribe	Extends a class
Pandora	Extends a class	Extends a class	Built-in or Extended	Serialization	Built-in	User must provide the solution	Built-in	Extends a class
Repast HPC	C++ class	Built-in	Built-in	Serialization	Built-in	Feature implemented indirectly through of other feature.	Built-in	C++ class

6. How to use Care HPS as a scientific instrument

Care HPS was designed to be a scientific instrument for two type of users: application area researchers and HPC experts. In this section, we will give some examples of how to use Care HPS from two points of view. In the first, we will focus on the application area researcher's point of view with the following examples: 1) how to implement an environment with an obstacle for a fish; and 2) how to create and extend an agent. We will not explain all of the implementation detail, but instead we will focus on the most important aspects. In the second point of view, we present the implementation of a new hybrid strip partitioning based on [15]. We will show how to use Care HPS if the HPC expert user wants to propose a new HPC solution.

6.1. Application area researchers' point of view

Application area researchers are interested in how to model their problems in ABMS. Therefore, in this section, we demonstrate how to represent the Agent, Environment and Model using Care HPS.

Agent-based model requires an environment where interactions occur between an environment's objects and the agents. The agents must execute local rules when these interactions happen. As an example of this kind of interaction, we can cite the repulsion behavior in the Fish Schooling model. The new fish's orientation depends on its position and orientation. If there is an obstacle ahead, the fish must change its direction in order to avoid a collision with the environment's objects. Consequently, this fish behavior influences its neighbors after the local interactions. Listings 1 and 2 show implementations of this fish's behavior. We create one linear plan, then we check the intersection between the plan and the vector equation of the line representing the fish's position and direction. The code checks if the next fish position is a root of environment equation. If so, the repulsion behavior should be executed in order to avoid the collision.

Listing 1: Creating an environment and its object for the fish schooling

```
pds_fish* PDS_FISH = new pds_fish();

/** Creates an environment with a
plan(ax + by + cz + d = 0) defined by
equation: 2*y - 3 */

environment* ENV = PDS_FISH->createEnvironment(
                new linear_plan(0,2,0,-3));
```

The agent must call the *check_environment* method 2 in order to interact with the environment. This method implements the interactions between the agent and the environment. The code checks if the collision happens for each object that has been created in the environment. If there is a collision, then the repulsion behavior of the agent is executed. Any other behavior of the agent can be used.

Listing 2: check_environment implementation

```
void fish::check_environment(void){

vector<object*> obj_env = ENV->getObjects();
for (vector<object*>::iterator ob=obj_env.begin();
                               ob!=obj_env.end();
                               ob++)

 if ((*ob)->check_collision(this->get_position(),
                            this->get_velocity(),
                            MAXIMUM_VISION_RANGE))
            this->repulsion(*this);
}
```

Until now, the objects in Care HPS have been represented by linear plan and circle. Care HPS enables the users to extend the object class by adding other types of objects, such as surfaces, objects with special characteristics, square or other types of geometric shapes.

The other important entity of ABMS is the agent, so the application area researchers need to create or extend an agent to model their problem. To create a new agent, the user has to follow two steps:

- The user has to create a class that inherited from the *agent* class.
- The user has to define the methods that represent the behavior of the agent and call these methods inside the update_agent method.

Application area researchers can face a situation in which extending an agent is a better option than starting from scratch. To extend an agent, three actions are required:

- The user just needs to create a new class extending another agent class and overriding the methods required.
- In Care HPS, a model is related with an agent. Therefore, in order to use the new agent class, the user has to create a class that extends a concrete model class. The next step is to override the factory methods that create the agent in the new model class. These factory methods must call the constructor of the new agent. Through these factory methods, Care HPS can dynamically create any agent. This is a key point of Care HPS.
- Finally, the user has to create a *pds* class extending the previous *pds* class. The *pds* class defines which model will be used in the simulation. The user must redefine the constructor, destructor and override the *getModel* method in order to specify the model that will be created at runtime.

This section briefly showed how simple it is for the user to represent an environment, model and agent using Care HPS.

6.2. Expert HPC user

One of the main purposes of Care HPS is to enable the expert HPC user to be able to develop and propose new techniques at HPC for ABMS. In this section, we give an example of how to develop a new partitioning approach using Care HPS. The partitioning of data is the distribution of space simulated throughout the distributed architecture. The space simulated in this context is composed of agents and an environment. These techniques try to reduce the communication time among the distributed processes. They also try to evenly distribute the agents throughout the cores.

We use the strip partitioning proposed in [15] as a base, which is implemented with a distributed memory paradigm. In this strip partitioning, the agents and environment are partitioned. This partitioning avoids the objects of the environment being shared between processes. Consequently, it avoids extra communication in order to update the state of the objects that are changed because of the interactions with the agents. In paper [15], we used an Ant Colony as a case study where we had observed a concentration of ants near nests and food. This is a natural and correct behavior of the ants in this model. The problem is that some cores become idle when this behavior occurs. Therefore, we propose a hybrid strip partitioning with the aim of decreasing the idleness of cores. We decrease the idleness of these cores through the creation of OpenMP threads, which are used to compute the extra agents that are in other cores. This partitioning checks the proportion of the quantity of agents inside a strip and dynamically creates a number of threads. First, we calculate an initial number of threads using Formula 3. Where the *number_agent* is the total number of agents inside a partition. The *total_number_agents* is the total number of agents simulated. And *number_cores* is the total number of MPI processes divided by 4. We use four because it fits better with the CPU architecture but this value is parameterized. Finally, the number of threads is defined when we apply the rules in accordance with Equation 4.

$$threads = number\_agent / total\_number\_agents * number\_cores \tag{3}$$

$$threads = \begin{cases} 2 & \text{if } threads = 1 \\ threads & \text{if } threads >= 2 \text{ and } threads <= 8 \\ 8 & \text{if } threads > 8 \end{cases} \tag{4}$$

The expert HPC user needs to follow some simple steps to implement a new partitioning strategy. Note that other extensions can follow the same sequence. It is possible because all other features have a similar design. The first step is to create a new partitioning class that inherits *partitioning_strip* class. The new partitioning class could be inherited from a partitioning interface, yet we chose inheriting directly from the *partitioning_strip* because we reuse all methods, except the method *execute*. The next step is the implementation of partitioning strategy overriding the method called *execute*. Listing 3 presents

the implementation of the *execute* method. In this code, we do a loop parallelization. We dynamically create the number of threads that will execute the behavior of a set of agents.

Listing 3: Implementation of partitioning strategy.

```
void partitionig_strip_hybrid::execute(){

          //here goes the other partitioning codes

long thread = number_agent/total_number_agents*number_cores;
if (abs(num_thread)>=1){
         if (abs(thread)==1)
           omp_set_num_threads(2);
         else if ((abs(thread)>=2) and (abs(thread)<=8))
           omp_set_num_threads(abs(thread));
         else if (abs(thread)>8)
           omp_set_num_threads(8);

         #pragma omp parallel default(none) firstprivate(i)
                         shared(_bucket)
         {
          unsigned long j;
          #pragma omp for private(j) schedule(dynamic) nowait
          // For each agent execute its rules.
          for(j=0;j<_bucket->size();j++)
                  _bucket->at(j)->update_agent();
         }
}
else{
  // Execute the loop without thread just with pure MPI
}
};
```

We have calculated the number of threads required. If the number of threads is higher than one, it indicates that there is a concentration of agents in the strip. The higher the number of threads calculated is, the higher the concentration of agents. We never create more than eight threads. It avoids potential saturation, resource contention and excessive locks. After finishing the partitioning strategy implementation, we have to change the concrete class of model that will use the new partitioning. The only change in this class is the implementation of the factory method *factory_partitioning()*. This factory method defines which partitioning algorithm will be created. The other option is for the user to create a subclass of a model and to just reimplement this factory method.

The user only has to change the factory method if they want to test other already-implemented strategies of partitioning, for example. No other changes are required. Therefore, the expert HPC user can invest time in the HPC technique instead of doing model implementations or other necessary implementations.

7. Experimental results

We have executed four experiments that explore different abstraction levels. In order to show that Care HPS reaches the proposed aim. These experiments

were carried out with three agent-based models as case studies: Fish Schooling, Ant Colony and Shopping Agent models. In the first experiment, we show that Care HPS is able to reproduce the interactions among the agents. We will compare the results of the simulation obtained by Care HPS with the results obtained using the Netlogo version of the Buyer model proposed by [48]. We used a simple alignment and replication process in this comparison. Gilbert and Troitzsch compare three different versions of buyers. They show the increase in agent intelligence by presenting the number of ticks required by all agents to buy all of their products. In the second experiment, we will present the result of the implementation of the fish repulsion behavior. This behavior avoids the collision of agent with environment. This implementation uses our math strategy to represent the objects of the environment. In the third experiment, we will present the results obtained for the hybrid partitioning strip for the Ant colony proposed in Section 6.2. Lastly, we will present the scalability of Care HPS using the Buyer model.

All experiments were carried out with an environment with the following characteristics: 16 nodes with 128 cores distributed in 8 sockets with three cache levels (AMD Opteron 6200 1.6 GHz, L2 (2MB), L3 (6MB)) , 64 GB RAM per node) with Gigabit Ethernet. Care HPS was developed using C++ (gcc 4.5.2), STL (C++ standard template library), MPI namespace (openmpi 1.4.3).

7.1. First experiment

ABMS tools must be able to model agent rules and behaviors. So, this model can create collective and emergent behavior. It is important that these tools can reflect the interaction among agents. Moreover, these tools must enable researchers to improve and analyze the agent behavior under study. The aim of this experiment is to show that Care HPS goes in this direction.

In this experiment, we accomplish a simple alignment and replication of the Gilbert and Troitzsch model [48]. The alignment of computational models or docking is a process to determine whether two models can produce the same results [49]. Linked to this is the replication, which refers to the creation of a new implementation of a conceptual model [50].

This model [48] was implemented in Netlogo [27]. Each agent has a list of products that must be bought in the stores. Which are distributed throughout an environment. Each shop sells just one type of product. The goal of the agent is to purchase all of the products on its list. Gilbert and Troitzsch [48] have developed three versions of the model:

- The first version is the simplest version. Agents randomly walk around the environment in order to find a shop which sells some product from their list.
- In the second version, the authors add another behavior. The agents can see neighboring shops that are inside their covered radius.

- In the third version, we find the smartest agents. This version has all of the behaviors of Versions 1 and 2. Moreover, the agents exchange information about the known shop positions.

These are the two main changes made to replicating the Gilbert and Troitzsch model with Care HPS. In the first, the environment of the Netlogo is composed of patches. The environment of Care HPS is represented by a Euclidean space. Thus, we use the same proportion of space and agents found in Netlogo to represent the Gilbert and Troitzsch model in Care HPS. In the second change, the Netlogo implementation is supported by cellular automata theory. Therefore, some functional features such as *seeing neighboring shops* are implemented using the Moore Neighborhood concept. This concept was replaced in Care HPS by covering radius. Thus, the agents interact with objects of the environment or agents that are inside this radius. We are concerned with establishing whether the replicated model creates output sufficiently similar to the outputs of the Gilbert and Troitzsch model. If we are able to establish this, then we can assume that Care HPS implemented a successful replication [50]. For this purpose, we use a category of replication standard defined by [49] called distributional equivalence. Whose aim is "showing that two models produce distributions of results that cannot be distinguished statistically".

For this statistic proof, we used the data presented in Table 2. This table depicts the average of the ticks of each version of the model for Netlogo and Care HPS. Gilbert and Troitzsch [48] executed their simulation 100 times. They got an average of 14310 ticks for the first buyer agent implementation. For the second buyer agent version, the authors decreased the number of ticks to 6983, and the last version of the buyer agent obtained approximately 2000 ticks. The faster the buyers buy all of their products, the smarter the model is. Therefore, the smartest version tends to finish the simulation in fewer simulation steps (ticks).

Table 2: Comparison of ticks of the solutions

	Version 1	Version 2	Version 3
Netlogo: Gilbert and Troitzsch	14310	6983	2000
Care HPS	19082	1361	348

We executed the t-test on these data in order to check if the average number of ticks was significantly different. There is a difference between the group average, if the p-value is less than 0.05. In this case, the resulting p-value was 0.81. Therefore, there is no significant difference between the average of ticks of these two samples. This means that the ticks produced by the two models are similar.

Moreover, the data of Table 2 shows that the number of ticks in Care HPS version decreases as well in the different versions. Thus, this experiment shows that Care HPS can model agents and their behaviors. The decrease in ticks in the versions shows that there are interactions among the agents, as well

as showing that changes in the agent's behavior influence the results of the simulation.

7.2. Second experiment

In this experiment, we will use Fish Schooling as a case study. Fish Schooling is a behavior in which a group of fish are swimming in harmonious patterns. It is composed of many fish of the same species. And it can be observed in almost 80 percent of the more than 20,000 known fish species in some phase of their life cycle. Fish schooling is one of the most common social groups in nature. This is an example of a self-organized system, where we find a leaderless, non-hierarchical and decentralized structure. This social aggregation shows complex emergent properties such as strong cohesion and a high level of synchronization. A collective behavior emerges as a result of local interactions between fish that are within a limited vision range.

We have implemented the fish schooling biological model described in [51] and [52]. This model takes into account that a new fish's position and orientation depends on the position and orientation of a fixed number of nearest-neighbors. The model identifies three vision areas: attraction, repulsion and parallel orientation (see Figure 3). Depending on the position of its neighbors, the fish chooses one of these behaviors. In the parallel orientation behavior, the group moves in the same direction. Attraction behavior maintains the cohesion of the group. And, the repulsion behavior avoids a collision between the fish.

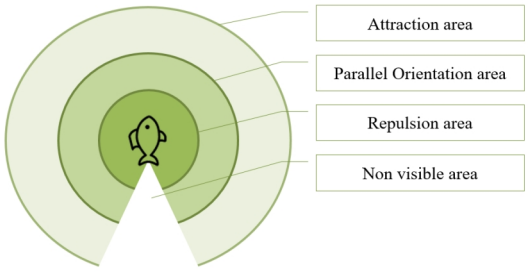


Figure 3: Fish's vision areas.

Figure 4 shows the result of the fish repulsion behavior for avoiding collision. This figure is an environment that represents a sea. In this environment, there is a school of fish swimming represented by agents in red and green. The yellow lattice is the obstacle created in order to show the interactions among the fish and the objects of the environment. This figure has a sequence of images of the fish school swimming towards the obstacle in different moments of the simulation (steps). When the fish "see" the obstacle (around steps 205 and 375), each fish executes its repulsion behavior and the collision is avoided.

Each fish can see the obstacle because each fish asks the environment if there is some obstacle inside its radius. In terms of code, it is implemented just by

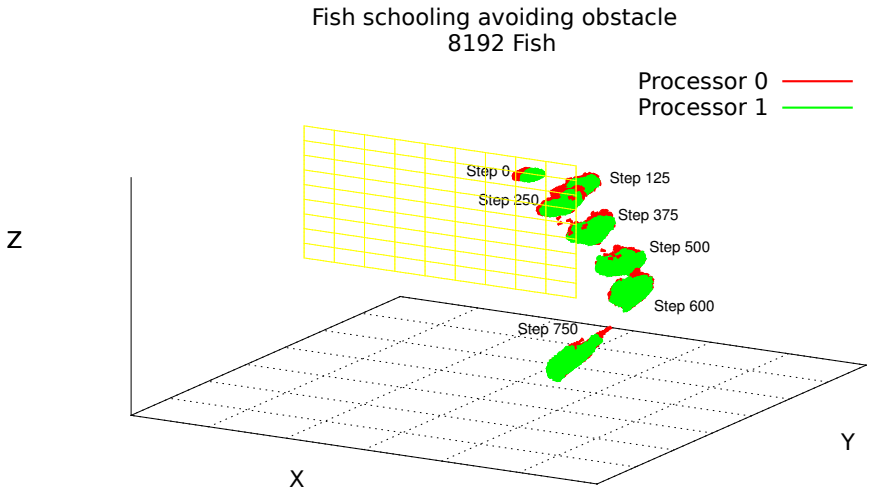


Figure 4: Fish repulsion behavior to avoid the collision. This experiment was executed in two cores using 8192 agents.

checking the intersection between the plan created and the vector equation of a line that represents the agent's position and direction (see Subsection 4.1).

This approach can be used to model other behaviors and other types of interactions between agents and the environment. As an example, we can cite the Ant Colony model [53], which has two main rules: searching for food and bringing the food to the nest. The ant moves randomly throughout the environment. The ant executes three procedures. Where we can find different interactions reproducing different behaviors, when it finds food:

- The ants drop a chemical in the environment. The chemicals are objects of a type circle that are created dynamically to represent the pheromone. The smell of the pheromone is reduced, decreasing the radius of circle.
- The ants "eat" the food when they find it. The food is represented per circle. The radius of the circle is decreased when an ant finds it. This means that the food was eaten by an ant.
- The ants carry the food to the nest. Another example is the Shopping model, where the buyer buys its product when it finds the store.

This experiment highlights three things:

- Care HPS is able to represent an environment and its objects using a math approach.

- Care HPS enables the interaction between agents and the objects of the environment.
- Care HPS enables the representation of simple rules in agents, and these simple rules can generate a collective behavior after agent interactions.

In addition, it is important to note that the user can choose between two built-in (circle and linear plan) objects. Alternatively, the user can create their own object to represent their model. The user also defines the behavior executed after the interaction with the environment. Therefore, Care HPS offers many possibilities for representations of the environment and its interactions with agents. Additional built-in objects will be included in future versions of Care HPS.

7.3. Third experiment

The strip partitioning proposed in [54] has some heuristics in order to distribute the strips throughout the architecture. The experiments conducted in that paper ([54]) shows that the ants concentrate their movements around the nest and food. The problem is that some cores have an overload while others are underloaded. The hybrid strip partitioning creates threads dynamically in order to compensate the idle cores in this scenario.

The experiments were carried out for 1000, 2500, 5000 and 10000 agents, and, they were executed in 64 cores for the pure MPI strip partitioning and hybrid strip partitioning. For hybrid partitioning, n threads were created between the intervals [2..8] for each MPI process. The number of threads created depended on the load of MPI processes. This follows the formula defined in Equation 4. Figure 5 presents the total execution obtained from the two versions of the strip partitioning algorithm.

As we can observe in Figure 5, the hybrid partitioning strip approach considerably reduces the total execution time of the simulation. Two aspects are important to note:

- All ants in this experiment emerge from the nest as specified in the Ant Colony model. This means that all of the ants in the simulation are in the same core at step zero. During the simulation execution, the agents start to migrate to other strips. The cores which contain the nest and food will always be in higher demand, while others might not be completely busy.
- We execute the experiment taking into account the worst partitioning heuristic in accordance with [15]. In the horizontal fixed heuristic, there is no special mapping policy nor is the computing cost of each strip taken into account. In this heuristic, each strip is allocated to one core.

After the results presented in Figure 5, we executed another test in order to confirm the overload effect of centralized agents in a few strips. We have executed the experiment distributing 10000 ants uniformly throughout the environment. For MPI and Hybrid version approaches, we got 2168.65 seconds

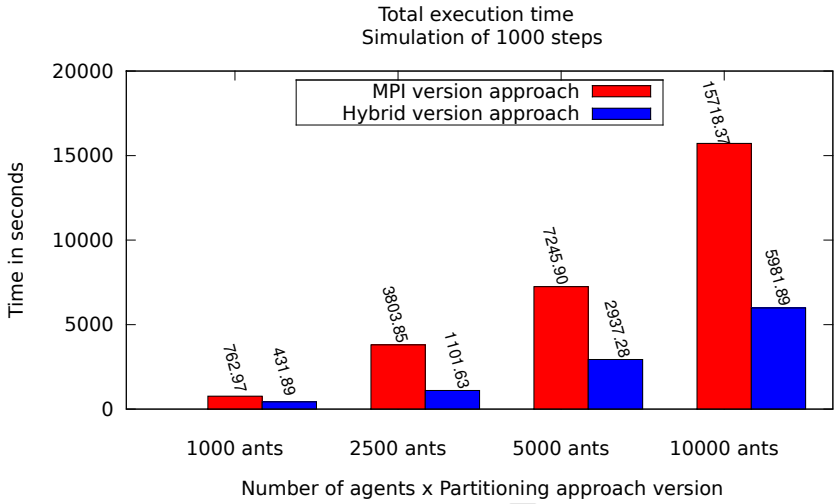


Figure 5: Total execution time of pure MPI and Hybrid strip partitioning algorithm.

and 3264.45 seconds, respectively. As we can see, the performance results are completely different. Firstly, all of the total execution times are lower than the previous experiments. This occurs because the ants are distributed over all of the strips. Therefore, the ants can find all of the food in the environment faster. Secondly, the agents are better distributed throughout the environment. Thus, the dynamic creation of threads overloads the cores because the number of idle cores is low. As a consequence, this results in a worse total execution time for the hybrid approach.

These results show that the hybrid partitioning strip is an interesting approach when there is a high concentration of agents in the same region of a simulated space. However, some questions emerge: Is the number of threads created appropriate for the size of the problem and the cores available? Does the hybrid partitioning strip have good results if a heuristic with better mapping is applied? What are the results of this hybrid partitioning if we change the agent to a Buyer? Care HPS proposes to provide features to easily answer these types of questions with just a few extensions and implementations of code.

7.4. Fourth experiment

In this experiment, we will approach scalability. Scalability is an important measure for parallel and distributed simulation. Simulations require a minimum number of executions in order to obtain data that is statistically reliable for a correct output analysis. In [13], it is possible to find some techniques for identifying the appropriate number of steps or the number of complete executions required for a simulation. Generally, this number of required simulations is high enough to require a fast single execution. Therefore, the solution must be scalable in order to adjust the number of cores to the time restrictions. This can

be illustrated briefly by Figure 6. The time required to execute in one core is around 4610 seconds. Suppose this model requires at least 10 executions of the simulation to give an appropriate output. This same result could be obtained using 64 cores. But the total execution time would decrease from almost 13 hours to approximately 1 hour.

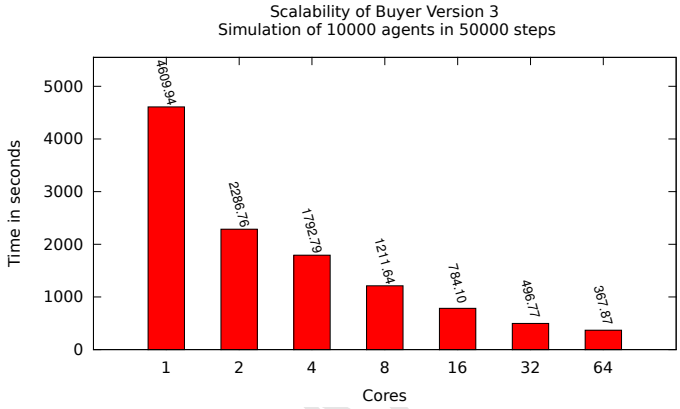


Figure 6: Scalability of Buyer Version 3 with 10000 agents in 50000 steps.

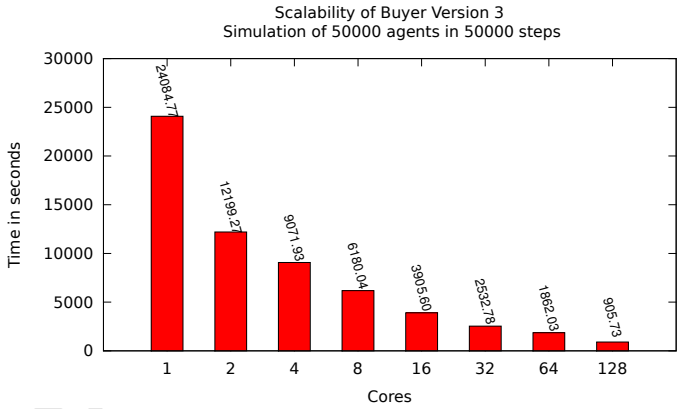


Figure 7: Scalability of Buyer Version 3 with 50000 agents in 50000 steps.

We have executed all experiments of this section with Buyer Version 3, where each one had to buy 10 products. Figures 6 and 7 provide the results obtained from the execution of the model. They show the decrease in time when we include more processors. This occurs because more processors are available to execute the same workload. Therefore, the total workload is divided among more units of processing. If the solution is scalable, the execution time will decrease when the number of processors increases. As it is possible to note, the

decrease in time is not proportional to the number of cores. It occurs because of the communication between the processes. In addition, there are random movements of agents in the environment. The other factor is the idleness of the processors due to the movement of agents on the partitions. All these issues are recurrent aspects in parallel and distributed research for ABMS.

In order to show that Care HPS can handle large models, we executed another test with a greater number of agents (Figure 8). If we consider a fixed number of cores (64 cores) and we increment the complexity of agents or the number of agents in the simulation, the execution time will increase. As we can see, Care HPS is able to carry out large numbers of agents. Therefore, Care HPS can execute larger simulations by just including more cores to solve the problem modeled, as depicted in Figure 9. We executed the simulation for 200000 and 250000 agents and increased the number of cores to get a lower execution time. The results show that scalable behavior is sustained in Care HPS. Besides scalability, Care HPS can handle high quantities of agents and models with high complexity. This also enables the user to adjust the parallel and distributed simulation to their needs and computational resources.

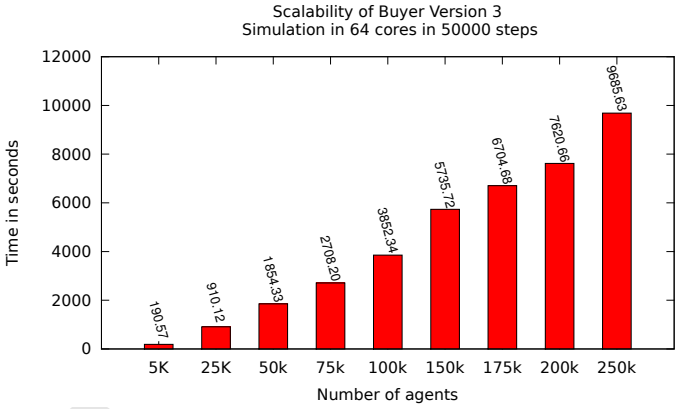


Figure 8: Scalability of Buyer Version 3 in 50000 steps executed in 64 cores.

8. Conclusion

In this paper, we introduce Care High Performance Simulation (HPS). Care HPS is more than a tool for agent-based modeling and simulation in a parallel and distributed architecture. Care HPS is a scientific instrument that enables both:

- application area researchers to gain knowledge about the system under study using agent-based models that require a high performance computing solution. This is possible because Care HPS offers a well defined and simple interface for this type of user in which all HPC complexity is hidden.

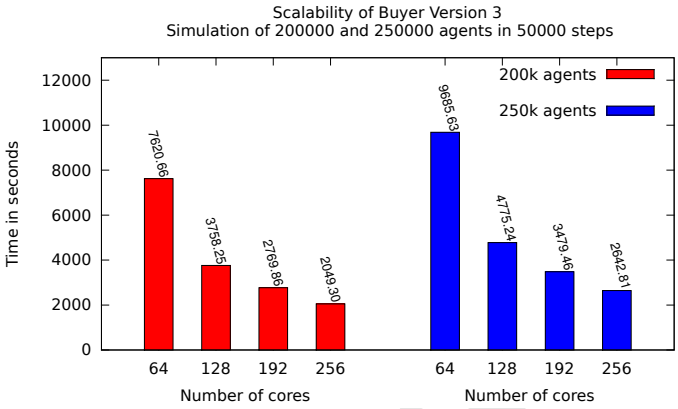


Figure 9: Scalability of Buyer Version 3 in 50000 steps to 200k and 250k agents executed in 64, 128, 192 and 256 cores.

- HPC expert users to develop techniques of high performance parallel and distributed simulation for agent-based model problems without high programming effort. Care HPS was projected using good object-oriented design practices, which enable the extension and reuse of the main HPS features.

As part of our main findings and contributions, we present Care HPS. We show through experimentation that Care HPS meets its objective, and it can be used as a scientific instrument for agent-based modeling that requires high performance parallel and distributed simulations. Currently, we are implementing the agent-based model for the assessment of *Aedes Aegypti* pupal productivity proposed in [54] using Care HPS. The aim is to use Care HPS to simulate a dengue outbreak that is both a demand computing problem and a real problem. At present, Care HPS continues in the development and improvement of its features. We have planned the implementation of the following features:

- Include a new layer in Care HPS in order to support the execution of the application area researchers' models in a cloud.
- Support for many types of agents within the same simulation space.
- Automatic visualization generation of the simulation.
- Create an output analysis module.
- GUI interface to define the agent and environment.
- Provide more built-in load balancing, load computing, partitioning strategies, environment and objects.

- Provide a script for model generation for the user. The idea is for some user input to automatically create the model classes for the application area researchers;
- Provide affinity of processes and threads.

Acknowledgment

This research has been supported by the MINECO (MICINN) Spain under contracts TIN2011-24384 and TIN2014-53172-P.

References

[1] P.-O. Siebers, C. M. Macal, J. Garnett, D. Buxton, M. Pidd, Discrete-event simulation is dead, long live agent-based simulation!, Journal of Simulation 4 (3) (2010) 204–210.

[2] D. Helbing, S. Balietti, How to Do Agent-Based Simulations in the Future: From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design, Springer, Berlin, 2012, Ch. Agent-Based Modeling, pp. 25–70, available at SSRN: http://ssrn.com/abstract=2339770.

[3] D. Mostaccio, R. Suppi, E. Luque, Using distributed events simulation for individual oriented models, in: International Mediterranean Multimodeling Multiconference, Vol. I, 2005, pp. 105–110.

[4] D. Mostaccio, R. Suppi, E. Luque, Simulation of ecologic systems using mpi, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, 2005, pp. 449–456.

[5] D. Mostaccio, C. Dalforno, R. Suppi, E. Luque, Distributed simulation of large-scale individual oriented models, Journal of Computer Science & Technology 6(2) (2006) 59–65.

[6] C. Dalforno, D. Mostaccio, R. Suppi, E. Luque, Increasing the scalability and the speedup of a fish school simulator, in: O. Gervasi, B. Murgante, A. Lagan, D. Taniar, Y. Mun, M. Gavrilova (Eds.), Computational Science and Its Applications ICCSA 2008, Vol. 5073 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 936–949. doi:10.1007/978-3-540-69848-7_74.
URL http://dx.doi.org/10.1007/978-3-540-69848-7_74

[7] J. C. Gonzlez, C. Dalforno, R. Suppi, E. Luque, A fuzzy logic fish school model., in: ICCS, Vol. 5544 of Lecture Notes in Computer Science, 2009, pp. 13–22.

[8] R. Solar, R. Suppi, E. Luque, High performance individual-oriented simulation using complex models, Procedia Computer Science 1 (1) (2010) 447 – 456, iCCS 2010.

[9] R. Solar, R. Suppi, E. Luque, High performance distributed cluster-based individual-oriented fish school simulation., Procedia CS 4 (2011) 76–85.

[10] R. Solar, R. Suppi, E. Luque, Proximity load balancing for distributed cluster-based individual-oriented fish school simulations, Procedia Computer Science 9 (0) (2012) 328 – 337, proceedings of the International Conference on Computational Science, ICCS 2012.

[11] R. Solar, F. Borges, R. Suppi, E. Luque, Improving communication patterns for distributed cluster-based individual-oriented fish school simulations, Procedia Computer Science 18 (0) (2013) 702 – 711, 2013 International Conference on Computational Science. doi:http://dx.doi.org/10.1016/j.procs.2013.05.234.
URL http://www.sciencedirect.com/science/article/pii/S1877050913003773

[12] F. Borges, A. Gutierrez-Milla, R. Suppi, E. Luque, A hybrid mpi+openmp solution of the distributed cluster-based fish schooling simulator, Procedia Computer Science 29 (0) (2014) 2111 – 2120, 2014 International Conference on Computational Science. doi:http://dx.doi.org/10.1016/j.procs.2014.05.195.
URL http://www.sciencedirect.com/science/article/pii/S187705091400372X

[13] C. A. Chung, Simulation modeling handbook: a practical approach, CRC press, 2003.

[14] F. Borges, A. Gutierrez-Milla, R. Suppi, E. Luque, Optimal run length for discrete-event distributed cluster-based simulations, Procedia Computer Science 29 (0) (2014) 73 – 83, 2014 International Conference on Computational Science. doi:http://dx.doi.org/10.1016/j.procs.2014.05.007.
URL http://www.sciencedirect.com/science/article/pii/S1877050914001847

[15] F. Borges, A. Gutierrez-Milla, R. Suppi, E. Luque, Strip partitioning for ant colony parallel and distributed discrete-event simulation, Procedia Computer Science 51 (0) (2015) 483 – 492, international Conference On Computational Science, {ICCS} 2015 Computational Science at the Gates of Nature. doi:http://dx.doi.org/10.1016/j.procs.2015.05.272.
URL http://www.sciencedirect.com/science/article/pii/S1877050915010807

[16] C. Deissenberg, S. van der Hoog, H. Dawid, Eurace: A massively parallel agent-based model of the european economy, Applied Mathematics and

Computation 204 (2) (2008) 541 – 552, special Issue on New Approaches in Dynamic Optimization to Assessment of Economic and Environmental Systems. doi:http://dx.doi.org/10.1016/j.amc.2008.05.116.
URL http://www.sciencedirect.com/science/article/pii/S0096300308003019

[17] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, C. Greenough, Flame: Simulating large populations of agents on parallel hardware architectures, in: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1, AAMAS '10, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2010, pp. 1633–1636.
URL http://dl.acm.org/citation.cfm?id=1838206.1838517

[18] X. Rubio-Campillo, Large Simulations and Small Societies: High Performance Computing for Archaeological Simulations, advances in geographic information science Edition, Springer, 2015, Ch. Part II, p. 119–137. doi:10.1007/978-3-319-00008-4_6.
URL http://link.springer.com/chapter/10.1007/978-3-319-00008-4_6

[19] X. Rubio-Campillo, P. V. Matías, E. Ble, Centurions in the roman legion: Computer simulation and complex systems, Journal of Interdisciplinary History 46 (2) (2015) 245–263. doi:10.1162/JINH_a_00833.
URL http://dx.doi.org/10.1162/JINH_a_00833

[20] M. Madella, B. Rondelli, C. Lancelotti, A. Balbo, D. Zurro, X. Rubio-Campillo, S. Stride, Introduction to simulating the past, Journal of Archaeological Method and Theory 21 (2) (2014) 251–257. doi:10.1007/s10816-014-9209-8.
URL http://dx.doi.org/10.1007/s10816-014-9209-8

[21] M. North, N. Collier, J. Ozik, E. Tatara, C. Macal, M. Bragen, P. Sydelko, Complex adaptive systems modeling with repast simphony, Complex Adaptive Systems Modeling 1 (1) (2013) 1–26. doi:10.1186/2194-3206-1-3.
URL http://dx.doi.org/10.1186/2194-3206-1-3

[22] M. T. Parker, What is ascape and why should you care, Journal of Artificial Societies and Social Simulation 4 (1) (2001) 5.
URL http://jasss.soc.surrey.ac.uk/4/1/5.html

[23] G. Cordasco, R. D. Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, Bringing together efficiency and effectiveness in distributed simulations: The experience with d-mason, Simulation (2013) 1236–1253.

[24] R. K. Standish, R. Leow, Ecolab: Agent based modeling for C++ programmers, CoRR cs.MA/0401026.
URL http://arxiv.org/abs/cs.MA/0401026

[25] M. Holcombe, S. Coakley, R. Smallwood, A general framework for agent-based modelling of complex systems, in: Proceedings of the 2006 European Conference on Complex Systems, 2006.

[26] V. Ginot, C. L. Page, S. Souissi, A multi-agents architecture to enhance end-user individual-based modelling, Ecological Modelling 157 (1) (2002) 23 – 41. doi:http://dx.doi.org/10.1016/S0304-3800(02)00211-9.
URL http://www.sciencedirect.com/science/article/pii/S0304380002002119

[27] U. Wilensky, Netlogo, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Accessed Apr. 02, 2015. http://ccl.northwestern.edu/netlogo/ (1999).
URL http://ccl.northwestern.edu/netlogo/

[28] X. Rubio-Campillo, Pandora: A versatile agent-based modelling platform for social simulation, in: Conference Proceedings SIMUL 2014, The Sixth International Conference on Advances in System Simulation, 2014, pp. 29–34.

[29] N. Collier, Repast hpc manual, Accessed Aug. 27, 2015. http://repast.sourceforge.net/docs.php (August 2010).

[30] D. Hiebeler, The swarm simulation system and individual-based modeling, Santa Fe Institute, 1994.

[31] N. Minar, R. Burkhart, C. Langton, M. Askenazi, The swarm simulation system: A toolkit for building multi-agent simulations, Santa Fe Institute Santa Fe, 1996.

[32] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, G. Balan, Mason: A multiagent simulation environment, Simulation 81 (7) (2005) 517–527. doi:10.1177/0037549705058073.
URL http://dx.doi.org/10.1177/0037549705058073

[33] R. J. Allan, Survey of agent based modelling and simulation tools, Tech. rep., Science and Technology Facilities Council Daresbury Laboratory (2010).

[34] TOP500.org, Top500 supercomputer sites, Accessed Sep. 03, 2015. http://www.who.int/denguecontrol/en/ (2015).

[35] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, C. Greenough, Exploitation of high performance computing in the flame agent-based simulation framework, in: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, 2012, pp. 538–545. doi:10.1109/HPCC.2012.79.

[36] S. F. Railsback, S. L. Lytinen, S. K. Jackson, Agent-based simulation platforms: Review and development recommendations, SIMULATION 82 (9) (2006) 609–623. arXiv:http://sim.sagepub.com/content/82/9/609.full.pdf+html, doi:10.1177/0037549706073695.
URL http://sim.sagepub.com/content/82/9/609.abstract

[37] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[38] M. J. North, N. T. Collier, J. R. Vos, Experiences creating three implementations of the repast agent modeling toolkit, ACM Transactions on Modeling and Computer Simulation (TOMACS) 16 (1) (2006) 1–25.

[39] S. Jafer, Q. Liu, G. Wainer, Synchronization methods in parallel and distributed discrete-event simulation, Simulation Modelling Practice and Theory 30 (0) (2013) 54 – 73. doi:http://dx.doi.org/10.1016/j.simpat.2012.08.003.
URL http://www.sciencedirect.com/science/article/pii/S1569190X12001244

[40] A. K. Jain, M. N. Murty, P. J. Flynn, Data clustering: A review, ACM Comput. Surv. 31 (3) (1999) 264–323. doi:10.1145/331499.331504.
URL http://doi.acm.org/10.1145/331499.331504

[41] S. Coakley, R. Smallwood, M. Halcombe, Using x-machines as a formal basis for describing agents in agent-based modelling., in: Agent-Directed Simulation, SpringSim 06, Huntsville, AL, USA, 2006.

[42] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, Mason: A new multi-agent simulation toolkit, in: Proceedings of the 2004 swarmfest workshop, Vol. 8, 2004, p. 44.

[43] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, G. Balan, Mason: A multiagent simulation environment, Simulation 81 (7) (2005) 517–527.

[44] Argonne National Laboratory, Repasthpc tutorial, Accessed Sep. 02, 2015. http://repast.sourceforge.net/hpc_tutorial/main.html (2015).

[45] L. Chin, Science, T. F. C. G. Britain), Flame-ii: A redesign of the flexible large-scale agent-based modelling environment, Tech. rep., Science and Technology Facilities Council Rutherford Appleton Laboratory (2012).
URL https://books.google.es/books?id=b3bRlwEACAAJ

[46] A. L. Chin, A. D. Worth, A. C. Greenough, S. Coakley, M. Holcombe, M. Kiran, Flame: An approach to the parallelisation of agent-based applications, Tech. rep., Science and Technology Facilities Council Rutherford Appleton Laboratory (2012).

[47] G. Kang, C. Márquez, A. Barat, A. T. Byrne, J. H. M. Prehn, J. Sorribes, E. César, Colorectal tumour simulation using agent based modelling and high performance computing, Future Generation Computer Systems (2016) –doi:http://dx.doi.org/10.1016/j.future.2016.03.026.

[48] N. Gilbert, K. G. Troitzsch, Simulation for the Social Scientist, 2nd Edition, Open University Press, 2005.

[49] R. Axtell, R. Axelrod, J. M. Epstein, M. D. Cohen, Aligning simulation models: A case study and results, Computational & Mathematical Organization Theory 1 (2) (1996) 123–141. doi:10.1007/BF01299065.
URL http://dx.doi.org/10.1007/BF01299065

[50] U. Wilensky, W. Rand, Making models match: Replicating an agent-based model, Journal of Artificial Societies and Social Simulation 10 (4) (2007) 2.
URL http://jasss.soc.surrey.ac.uk/10/4/2.html

[51] A. Huth, C. Wissel, The simulation of the movement of fish schools, Journal of Theoretical Biology 156 (3) (1992) 365 – 385.

[52] A. Huth, C. Wissel, The simulation of fish schools in comparison with experimental data, Ecological Modelling 75-76 (1994) 135 – 146, state-of-the-Art in Ecological Modelling proceedings of ISEM's 8th International Conference.

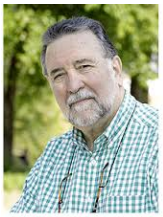
[53] U. Wilensky, Netlogo ants model, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., visited on 30-11-2014 (1997).
URL http://ccl.northwestern.edu/netlogo/models/Ants

[54] F. Borges, A. Gutierrez-Milla, R. Suppi, E. Luque, M. de Brito Arduino, An agent-based model for assessment of aedes aegypti pupal productivity, in: Proceedings of the 47th conference on Winter simulation, 2015.

AUTHOR BIOGRAPHIES

FRANCISCO BORGES received his BsC in Data Processing (2000) and postgraduate in Web Application and Distributed Components (2003) by Faculdade Ruy Barbosa, Brazil. Later, he obtained two MsC degree: Computational Modelling by Fundação Visconde de Cairu (2007, Brazil), and High Performance Computing and Information Theory by Universitat Autonoma de Barcelona (UAB) (2013, Spain). He works for Brazilian Government and he is PhD candidate and research fellow at UAB.

ALBERT GUTIERREZ-MILLA received a BsC degree from the Universitat Autonoma de Barcelona. He has worked in scientific projects as ALBA Synchrotron, the High Energy Physics Institute (IFAE) or CERN in Geneve (Switzerland). He is currently working towards his PhD at UAB, where he is a research fellow and teacher.



EMILIO LUQUE was awarded his degree in physics and his PhD from the University Complutense of Madrid (UCM) in 1968 and 1973, respectively. Between 1973 and 1976, he was an associate professor at the UCM. Since 1976, he has been a professor of Computer Architecture and Technology at the University Autonoma de Barcelona (UAB). Professor Luque has been the Computer Science Department Chairman for more than 10 years. He has been an invited lecturer/researcher at universities in the USA, Argentina, Brazil, Poland, Ireland, Cuba, Italy, Germany and the PR of China.

REMO SUPPI received the diploma in Electronic Engineering from the Universidad Nacional de La Plata (Argentina), and the PhD degree in Computer Science from the Universitat Autonoma de Barcelona (UAB) in 1996. At UAB he spent more than 20 years researching on topics including Computer Simulation, Distributed Systems, High Performance and Distributed Simulation applied to ABM or Individual oriented Models. He has published several scientific papers on the topics above and he is associate professor since 1997 at UAB and member of the High Performance Computing for Efficient Applications and Simulation Research Group (HPC4EAS) at the UAB.