

Asynchronous and Distributed Multi-agent Systems: An Approach Using Actor Model

Felipe D. Reis¹, Tales B. Nascimento¹, Carolina G. Marcelino^{2,3},
Elizabeth F. Wanner¹, Henrique E. Borges¹, and Sancho Salcedo-Sanz³

¹ Federal Center of Technological Education of Minas Gerais (CEFET-MG),
Department of Computing, Brazil

{felipeduarte,henrique}@lsi.cefetmg.br, efwanner@cefetmg.br

² Federal University of Rio de Janeiro (UFRJ), Institute of Computing, Brazil
carolina@ic.ufrj.br

³ University of Alcalá, Department of Signal Processing and Communication, Spain

Abstract. Agent-based and individual-based modeling have been widely used to simulate ecological systems. The historical architectures designed to artificial life simulation, namely LIDA and MicroPsy, rely into classical concurrence mechanisms based on threads, shared memory and locks. Although these mechanisms seem to work fine for many multi-agent systems (MAS), notably for those requiring synchronous communication between agents, they present severe restrictions in case of complex asynchronous MAS. In this work, we explore an alternative approach to handle concurrency in distributed asynchronous MAS: the actor model. An actor is a concurrent entity capable of sending, receiving and handling asynchronous messages, and creating new actors. Within this paradigm, there are no shared memory and, hence, no data race conditions. We introduce L2L (a short for: Learn to Live, Live to Learn) architecture, a biological inspired distributed non-deterministic MAS simulation framework, in which the autonomous agents (creatures) are endowed with a functional and minimal nervous system model enabling them to learn from its own experiences and interactions with the two-dimensional world, populated with creatures and nutrients. Both creatures and nutrients are encapsulated in actors. The system as a whole performs as a discrete non-deterministic dynamical system, as well as the creatures themselves. The scalability of this actor-based framework is evaluated showing the system scales up and out – many processes per processor node and in a computer cluster. A second experiment is realized to validate the architecture, consisting of an open-ended foraging simulation with both one or many creatures and hundreds of nutrients. Results from this specific actor-based version are compared to those from a classical concurrency version of the same architecture, showing they are equivalent, despite the fact that the former version scales a lot better. Moreover, results show that exploration of the world is unbiased, leading us to conjecture that our system follows ergodic hypothesis. We argue that the actor-based model proves to be very promising to modeling of asynchronous complex MAS.

Keywords: Distributed MAS · complex agents · situated cognition.

1 Introduction

It is well known that simulating ecological systems in an individual-based (IB) or an agent-based (AB) perspective, is more intuitive and faithful to reality instead of pure mathematical approaches [1]. Furthermore, the complexity of those systems is naturally handled with AB models (ABM) [2]. Although the ABM is consolidated in literature, there is no general technique to handle the continued growth of complexity of multi-agent systems (MAS) [2].

Despite the fact IB and AB modeling are not equivalent in concept, both have been used to understand the complexities of ecological management. Bousquet & Le Page[3] show many concepts based on ecosystems modeling and point out the possibility of rich hierarchy simulation to understand natural phenomena from different perspectives. However, there is no mention concerning technological difficulties related to this kind of simulation.

In the field of cognitive science, the ABM has been used to simulate and investigate the possibilities of various mind and brain theories. As an example, the LIDA architecture is based on Global Workspace Theory [4]. It specifies for the agents a cycle of sensing, processing sensor signals and decision making that occurs in parallel and asynchronously, while respecting the order of component stimulation to maintain the cognitive process coherence.

In the model MicroPsi - a multi-agent cognitive framework based on the Psi theory [5, 6] - the agent is composed of several components, including sensors and modulators which interacts with the environment, and a motivation set defined by the needs of the agent (*i.e.*, energy, hunger, thirst). These needs trigger the internal cognitive process of the agent and modulate its behavior. There is also a component that coordinate the allocation of the resources to internal components during execution.

Modeling natural phenomena has to take into account the simultaneity of various events. In real cases, the agent and the environment interacts through a stimulation process simultaneously [7]. From this perspective, the actor model is a powerful technique to design this kind of interaction. In the actor model, "computation is conceived as distributed in space where computational devices communicate asynchronously and the entire computation is not in any well-defined state" [8]. Following this definition, actor model helps to design and implement systems in which the events must happen asynchronously and simultaneously. Furthermore, as the computation is defined as distributed, this approach gives the model the ability to scale over many machines, without changing the fundamental implementation decision.

Given that the actor model is very congruent with the need in agent-based modeling systems (ABMS), its usage now is widespread over the literature. *Angiani et al.* [9] presents a framework based on actors to simulate agent-based modeled systems. The authors claim that the actor model of computation has affinity with designing ABMS since both are grounded on reactive behaviour, isolating its internal state from the outside world and communicating through message exchange. The framework is distributed and can be executed in a cluster. A proof of concept is presented for study crowd-behaviour in a building

evacuation scenario. Another framework designed for ABMS is AgentScriptCC [10], a domain-specific language for designing rule-based agents. The language is translated to an execution environment that is based on Akka actor framework. According to the authors, defining the agent via actor model results in a more fine-grained architecture and the execution of agent-oriented program is enhanced, both in scalability and performance perspectives.

Other architectures that take advantages of the actor computational model, *e.g.* the distributed-first and asynchronous-first perspectives, can be found in the literature. Leon [11] creates a .NET framework based on multi-agent systems inspired by actor model. A new framework for the analysis of financial networks is proposed by Crafa [12]. The approach uses agent-based to actor-based reactive systems. Results indicate that the systemic effect of initially defaulting the banks of some country with price shock parameter is approximately the same that is obtained by shocking all the European system. In [13] agent-systems based on actors are proposed to optimize the resources in IoT systems. Actor modelling can also be used to improve optimization algorithms, as ant colony optimization [14] and multi-objective in a hierarchical approach [15]. In addition, multi-agent system can be applied to reinforcement learning models [16, 17].

Given that the actor model has been shown to be useful for constructing distributed and efficient MAS, we propose here a new architecture called L2L (Live to learn, learn to live) for simulating artificial creatures. This simulator, a biologically inspired MAS, is a new implementation of a former version previously described by [18]. The later could only be executed in one machine with a narrow limit on the number of agents, thus preventing ecosystems simulations. The current implementation specifies each creature and nutrients as actors, which interact through message passing. The results show a remarkable enhancement in system scalability, allowing further simulations and involving hundreds of complex creatures in a computer cluster.

The organization of this paper is as follows: the L2L architecture is presented together with the actor model in section 2. Experimental results are discussed in section 3. The, section 4 concludes the work.

2 Proposed Model

2.1 L2L architecture

L2L is a biologically inspired architecture of embodied and embedded cognition that has been used to study foraging and action selection [18]. The architecture consists of a two-dimensional surface of a toroidal world, which is populated by artificial creatures (active components) and nutrients (passive components). Both of them interact asynchronously exchanging stimuli. The nutrients have different nourishing values and the creature does not have *a priori* knowledge about those values neither the food distribution around the world.

The creature is endowed with a basic nervous, emotional and sensory-motor systems, and various internal “organs” that interact asynchronously with each

other. This interaction represents the biological nervous and hormonal stimulation and is implemented via threads and shared memory. The access to shared memory is controlled by locks, and is implemented by all threads.

The creature internal state is modulated, but not determined, by the external stimuli received through the sensory system. The whole system operates in a non-deterministic dynamics through time and space. It is expected that the behaviour of adaptation, and thus the establishment of food preferences, emerges from the inner emotional-cognitive process. However, it is not the focus of this paper to describe that process (for more details see [18]).

The intrinsic model's complexity combined with implementations of decision about concurrency (use of shared memory, threads and locks) have made a single creature computationally heavy, limited to two creatures by computer core. Increasing this number causes the creature behavior becoming incoherent in the sense that, behavioral dynamics loose congruence with inner nervous system dynamics. This upper bound severely limits the studies involving ecological simulations, e.g. socio-genesis, semiotics, population dynamics, etc. Aiming to use more machines to run a large number of creatures in one simulation, it is necessary to decouple them from the shared memory. The chosen alternative to achieve this goal relies on the actor model.

2.2 The Actor Model

An actor is a mathematical concept of a universal primitive of digital computation [8]. The actor is an entity capable of sending messages, receiving and handling messages, as well as creating new actors. The actor model is by conception concurrent, and all the communications are asynchronously done through exchange, *i.e.*, there is no shared memory and hence no data race conditions. The messages in the actor model are decoupled from the sender and delivered by the system on a best efforts basis[19]. There are no guarantees as to the ordering of messages, so, the system must be carefully prepared to handle the incoming messages no matter their order.

There are many implementations of the actor model, *e.g.*, Erlang - a programming language which its concurrency mechanism works asynchronously as actors [20], and Quasar¹ library that runs over Java Virtual Machine (JVM). The Akka toolkit² is one of these, and we chose it because it best fit the technologies we already used. It does not respects all of the theoretical description from above due to technological constraints and practical factors. Since Akka runs on a JVM and is a library, it cannot strictly guarantee data separation from the actors. Also, the toolkit implements message ordering between pairs of actors. All actions taken by an Akka actor are in response to some received message, whether sent by itself or by another actor, *i.e* the system is reactive. Yet, it does not guarantee message delivery between two computers.

¹ <http://docs.paralleluniverse.co/quasar/>

² <http://akka.io/>

The advantage of using the actor model is that it provides a higher level of abstraction when compared with the classical concurrent model. Due to the fact that there are no data race conditions, the model scales up, using efficiently the available resources in one machine. Also, the toolkit we used provides an implementation that works with multiple machines, enabling scaling out. In the new implementation, the actor model is used to simulate a two-dimensional world which is composed of artificial creatures and nutrients. Either creatures and components are encapsulated in actors and they exchange stimuli with each other. The creatures' and nutrients' implementations are kept as proposed by *Campos et al.*[18], being just the communication between them implemented using the actor model. There is an underlying control structure responsible for managing the simulations and replenish the nutrients, if necessary. This structure is shown in Fig. 1.

As said previously, the creature is an actor, and it encapsulates its nervous systems and its other subsystems. Those creature components still use threads as the concurrence model and must run in the same machine. For managing those creatures, there is a holder actor that creates and keeps track of them, and a repository actor that delegates which holder will receive the next creation order. The nutrient repository and holder are analogous to its creature counterparts. The simulation manager is an actor responsible to create, start and stop a given simulation, *i.e.*, given how many creatures and nutrients are populating the world, it asks the repositories to create them and wait for the ready response. When everything is configured, it send messages to start the simulation.

For the control messages (those exchanged by control actors), we implement a synchronous protocol. It increases the reliability of message delivery with a penalty on the overall performance. A miscommunication between this kind of actors would result in simulation failure. Since this kind of communication is not prevailing, the performance cost is acceptable. The creature and nutrient actors communicate with the collision detector actor using a fire-forget (best effort) method, that is faster than the synchronous method. Since these actors produce the statistics for the simulation, some message loss is tolerable.

3 Experimental Results

We propose two experiments to validate the present model and evaluate its scalability. The experiments have been performed in a small computer cluster, composed of eight slave machines and a master connect through a Gigabit Ethernet network. The cluster machines have 3 Intel i5-3470 processors, 32 GB dual channel DDR3 RAM. In those simulations, a number of artificial creatures has been created, without any kind of *a priori* knowledge about the world, in a virtual world filled with different kind of nutrients. In both experiments the nutrient density has been maintained constant during the executions, *i.e.*, if a nutrient is eaten, another one of the same type is promptly created in an aleatory position (uniform distribution).

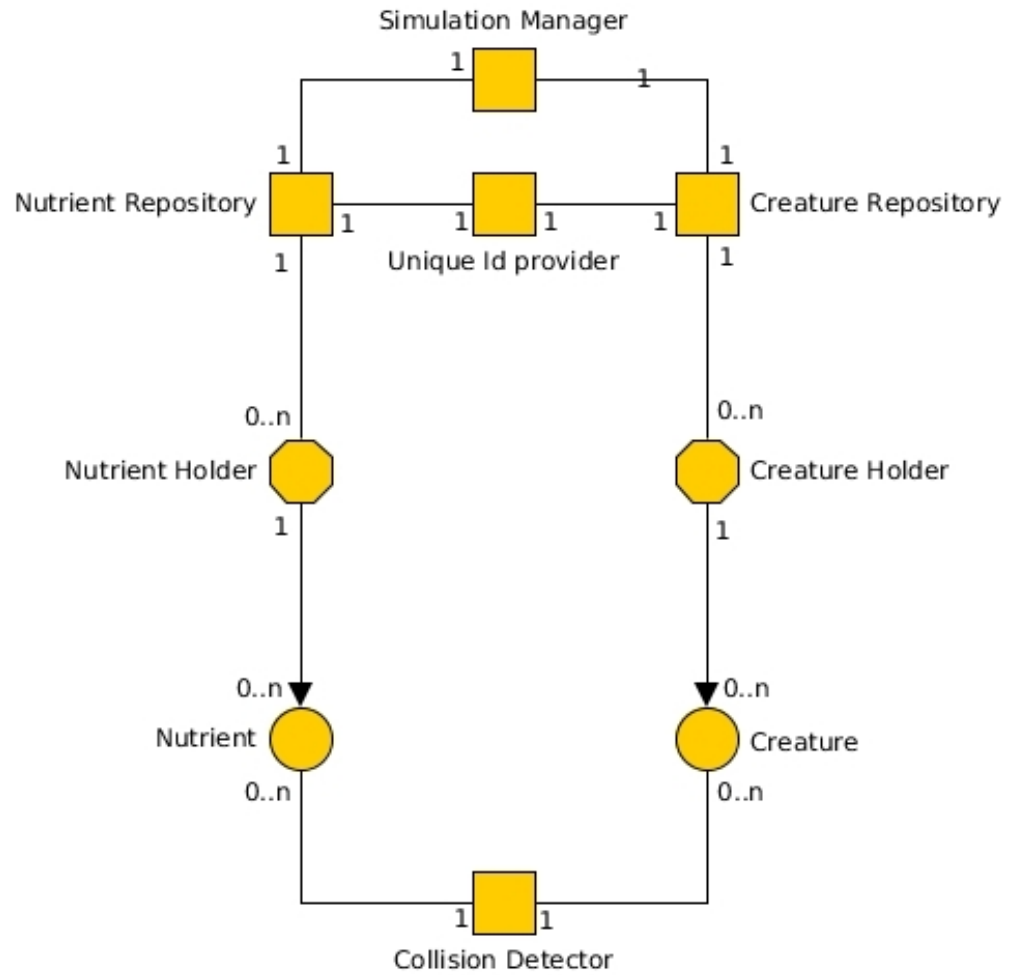


Fig. 1: Block diagram for the implemented model. All but the Creature and Nutrient are control actors. Each simulation must have at least one of each type of actor, and at most one actor represented by a square. The communication between the Creature and Nutrient actors with the Collision detector are fire-forget. The other communications have a acknowledge mechanism

3.1 Model scalability

For evaluating the scalability of the implementation using the actor model, we measure the average number of exchanged external stimuli between actors as a function of computer nodes running, while keeping the ratio of creatures per machine constant. We start with 10 creatures and 90 nutrients, using one machine for the creatures and another one for the rest of the system. A simulation comprises 30 runs for each configuration, and each run is interrupted after 300 seconds, as our purpose is to evaluate only the system scalability. It is worth noticing, however, that L2L is designed for open-ended simulations (see Fig. 2).

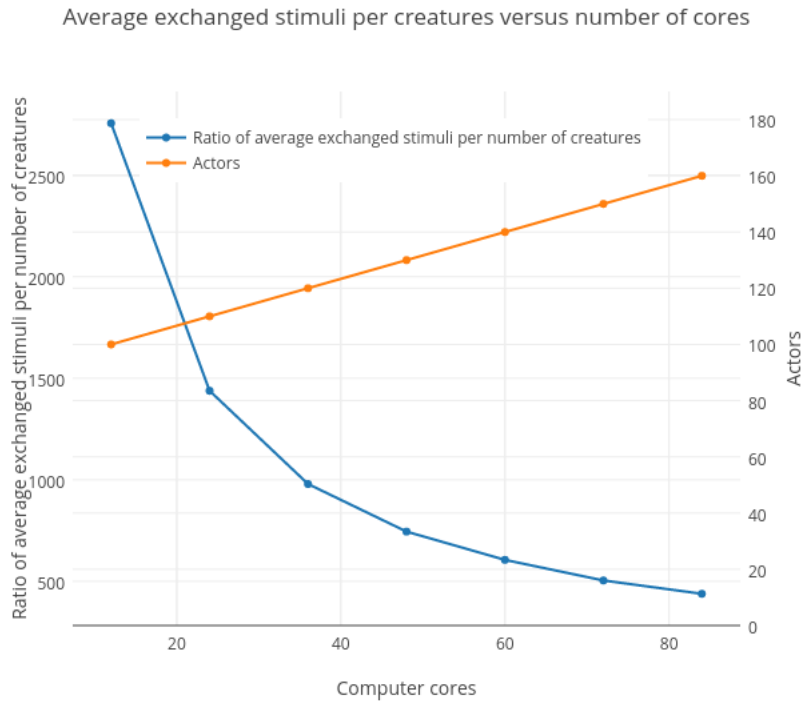


Fig. 2: Average of exchanged stimuli over number of creatures versus number of computer cores. Each computer node possesses 12 cores. The experiment shown that it is possible to increase the number of actors by adding more computer cores in the simulation. This doesn't cause a degradation in the chosen performance indicator (ratio of exchanged stimuli per creature). In fact, this PI tends to a constant, showing that the number of creatures grows faster than the average of exchanged stimuli.

Fig. 2 shows, in the left axis, the ratio of average of exchanged stimuli over number of creature, as well as the total number of actors in the right axis. By increasing both the number of computers and creatures, the ratio curve tends to a constant, which means that the number of creatures grows faster than the average exchanged stimuli. This is the expected behavior because the nutrient density in the world does not change and all the creatures operate in the same manner, thus exchanging approximately the same amount of stimuli.

3.2 Model validity

A experiment has been designed to compare the version using actor model and the former version, based on threads and shared memory, and to verify the correctness of the first. It consists of a simple open-ended (the simulation finish when the creature dies) foraging simulation with one creature and ninety nutrients. It has been repeated 50 times for both versions of architecture. Three measures are collected during the experiment: the creature position updates, the hunger deprivation, and the behavioural efficiency over time. The system has a spacial dynamics. Fig. 3 shows the superimposed path covered by 30 creatures in both implementations. It can be seen the creatures tend to cover all space. Simulations with more creatures show an increased covered area and indicate that the system as a whole seems to be ergodic [21]. It also shows that the creatures are not skewed in just one direction.

Hunger deprivation is the difference between the metabolic consumption of a creature and its energetic income from nutrients, measured in arbitrary units [18]. The metabolism is constant, thus the hunger deprivation grows at a constant rate. It reduces when the creature eats, with intensity equivalent to nutrient energetic value. When the deprivation reaches 7, the creature dies. Fig. 4 shows the hunger deprivation of a typical creature, (*i.e.*, a creature whose lifetime is close to the mean lifetime of all creatures) of both implementations.

Behavioral efficiency is a function of the hunger deprivation that determines the speed and the opening of the creature's vision field, thus changing its ability to find food. It has been modeled according to Yerkes-Dodson law [22] and differs between simple and complex tasks efficiency. A simple task is the one made by a creature when sensing only one nutrient while the complex task happens when there are more than one in the sensitive field.

Fig. 5 shows the temporal mean of behavioral efficiency for both implementations. The creature behavior is similar in both versions, which means the creatures of the new implementation retains a coherent behavior. In the Fig. 5a, there is a shrinkage on time axis, caused by changes made on the system to implement the actor model, which can be corrected by tuning the creature metabolic rate. In either Fig. 5a and 5b, the majority of the creatures is dead from 0.15 hours on, and the results are no longer statistically valid, due to the sampling error.


Looking into the 30 superimposed tracing of both akka and no-akka implementations in Fig. 3, both experiments explore evenly the artificial world space. The akka version looks more dense, and this is given by the fact that we are running more agents in this experiment. If we look inside the indicators' average over time, arousal in Fig. 4 and behavioural efficiency in Fig. 5, we can see the curves exhibit strictly the same shape. Both the experiments have, for either simple and complex tasks, a period of reaching the maximum efficiency, a small period of stability for complex task and then its decay, while the efficiency for complex task continues growing. This is also in accordance with the results shown in the original where L2L is presented [18].

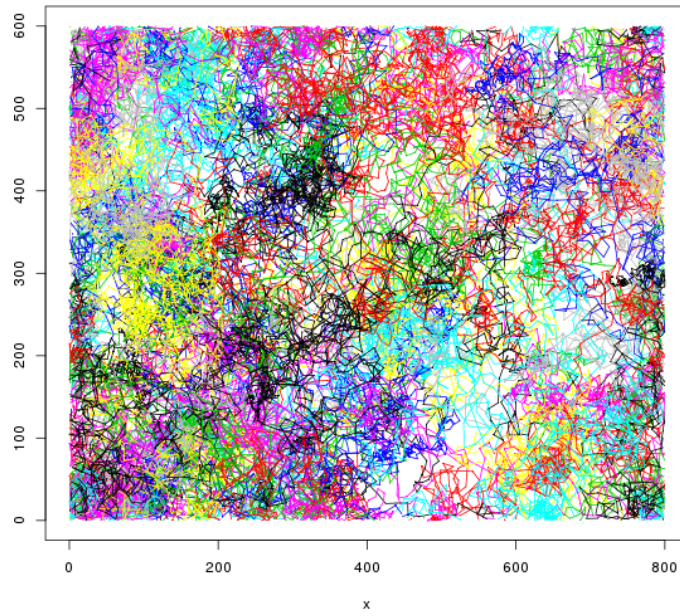
Given that functionally both implementations are similar, the main difference between the original architecture and the version presented in this paper is that, in the last one is possible to execute many creatures in the same simulation, and scale that with the number of machines in the cluster. In the first version of L2L, this was not possible due to the limitations the concurrency model based on shared memory imposes on the communication between the simulation entities. The main shared resource was the environmental shared pool where creatures and nutrients communicate. Removing this single point of communication, enabling direct communication between creatures and the environment enabled us to run many-creature simulations without performance degradation. We consider this the main contribution of this work. Future work is to enable direct communication between the internal creature components, so we expect to improve more the performance and be able to implement new structures that could show more fine social behaviors such as reproduction.

4 Conclusion

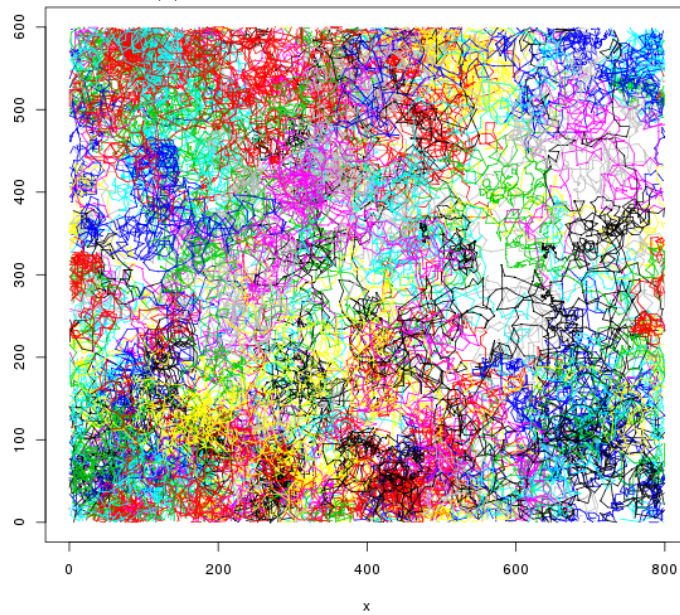
The L2L architecture was used as a foraging simulator, and while there are many more simulators of this kind, L2L has a non-deterministic spatially-explicit and temporal dynamics. Also, the system appeared to be ergodic, a point which needs further investigation. It was built having in mind the perspective of simulating population dynamics. However, the scalability problems we faced made it infeasible. Nonetheless, using the actor model, we were able to scale out the architecture to multiple machines, clustered or not, and reopen new research possibilities, including ecological simulations. A work in this direction is currently being done. Finally, the toolkit we have adopted fits well enough our purposes as our system model is asynchronous and fault tolerant. Actor-based approach may not be appropriate to all MAS, mainly those not presenting these traits. Albeit other nontraditional implementations of actor model may be useful in such scenarios.

Acknowledgment

 This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754382. This research has also been partially supported by Comunidad de Madrid, PROMINT-CM project (grant ref: P2018/EMT-4366) and by the project PID2020-115454GB-C21 of the Spanish Ministry of Science and Innovation (MICINN). The authors thank UAH, UFRJ and CEFET-MG for the infrastructure, and Brazilian research agencies for partially support: CAPES (Finance Code 001), FAPERJ, FAPEMIG and National Council for Scientific and Technological Development – CNPq. “The content of this publication does not reflect the official opinion of the European Union. Responsibility for the information and views expressed herein lies entirely with the author(s).”

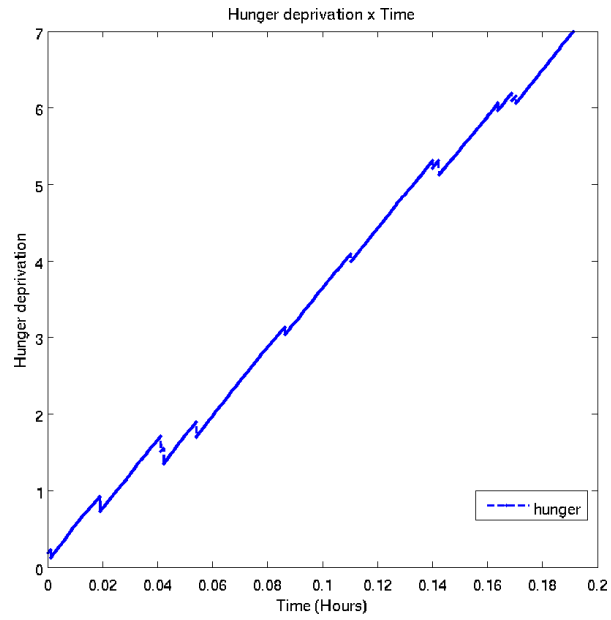


(a) implementation with actor model

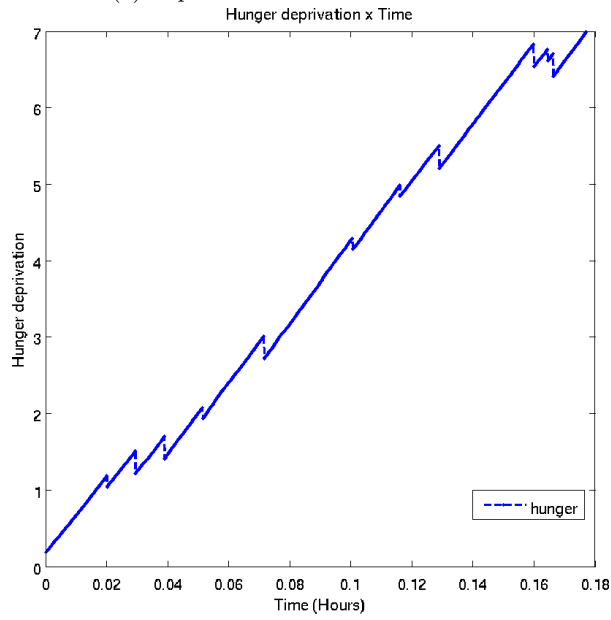


(b) implementation without actor model

Fig. 3: Two dimensional plot of superimposed world exploration by 30 independent creatures. The x and y axis are Cartesian coordinates. a) in an actor-based implementation b) in a classical implementation (using threads and shared memory)

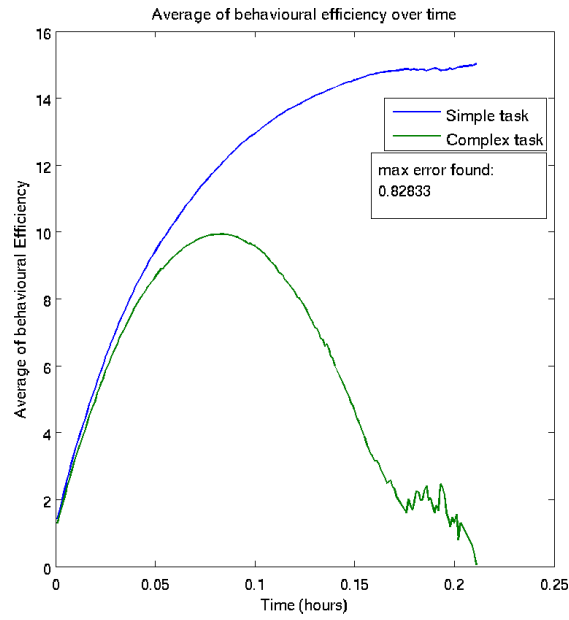


(a) implementation with actor model

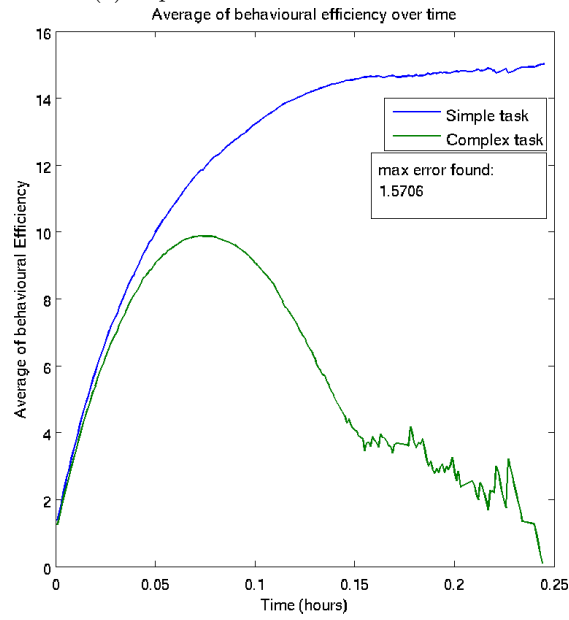


(b) implementation without actor model

Fig. 4: Hunger deprivation a typical creature of both implementations.



(a) implementation with actor model



(b) implementation without actor model

Fig. 5: Behavioral efficiency of creatures in both implementations.

References

1. J. A. Beecham and K. D. Farnsworth. Animal foraging from an individual perspective: An object orientated model. In *Ecological Modelling*, 1998.
2. Li An. Modeling human decisions in coupled human and natural systems: Review of agent-based models. *Ecological Modelling*, 2012.
3. François Bousquet and Christophe Le Page. Multi-agent simulations and ecosystem management: a review. *Ecological modelling*, 176(3):313–332, 2004.
4. David Friedlander and Stan Franklin. Lida and a theory of mind. In *Artificial general intelligence, 2008: Proceedings of the first agi conference*, volume 171, page 137. IOS Press, 2008.
5. Joscha Bach. The micropsi agent architecture. In *Proceedings of ICCM-5, international conference on cognitive modeling, Bamberg, Germany*, pages 15–20. Citeseer, 2003.
6. Joscha Bach. Micropsi 2: the next generation of the micropsi framework. In *International Conference on Artificial General Intelligence*, pages 11–20. Springer, 2012.
7. H. R. Maturana. Everything is said by an observer. *Thompson, W.I. Gaia: a way of knowing. Political implications of the new biology*, pages 11–36, 1987.
8. Carl Hewitt and H Zenil. What is computation? actor model versus turing’s model. *A Computable Universe: Understanding and Exploring Nature as Computation*, pages 159–85, 2013.
9. G. Angiani, P. Fornacciari, G. Lombardo, A. Poggi, and M. Tomaiuolo. Actors based agent modelling and simulation. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 443–455. Springer, 2018.
10. M. Parizi, G. Sileno, T. van Engers, and S. Klous. Run, agent, run! architecture and benchmarking of actor-based agents. In *Proceedings of the 10th ACM SIG-PLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, pages 11–20, 2020.
11. F. Leon. ActressMAS, a .NET Multi-Agent Framework Inspired by the Actor Model. *Mathematics*, 10(3), 2022.
12. S. Crafa. From agent-based modeling to actor-based reactive systems in the analysis of financial networks. *Journal of Economic Interaction and Coordination*, 16:649–673, 2021.
13. H. Nguyen, T. Do, and C. Rotter. Optimizing the resource usage of actor-based systems. *Journal of Network and Computer Applications*, 190:103143, 2021.
14. M. Starzec, G. Starzec, A. Byrski, and W. Turek. Distributed ant colony optimization based on actor model. *Parallel Computing*, 90:102573, 2019.
15. M. Idzik, A. Byrski, W. Turek, and M. Kisiel-Dorohinicki. Asynchronous actor-based approach to multiobjective hierarchical strategy. pages 172–185. Springer International Publishing, 2020.
16. D. Cao, W. Hu, J. Zhao, Q. Huang, Z. Chen, and F. Blaabjerg. A Multi-Agent Deep Reinforcement Learning Based Voltage Regulation Using Coordinated PV Inverters. *IEEE Transactions on Power Systems*, 35(5):4120–4123, 2020.
17. J. Li, T. Yu, and X. Zhang. Coordinated load frequency control of multi-area integrated energy system using multi-agent deep reinforcement learning. *Applied Energy*, 306:117900, 2022.
18. L. Campos, R Dickman, F. Graeff, and H. Borges. A concurrent, minimalist model for an embodied nervous system. *International Brazilian Meeting on Cognitive Science*, 2015.

19. Carl Hewitt. Actor model of computation: scalable robust information systems. *arXiv preprint arXiv:1008.1459*, 2010.
20. Joe Armstrong. *Programming Erlang: software for a concurrent world*. Pragmatic Bookshelf, 2007.
21. Isaac P Cornfeld, Sergej V Fomin, and Yakov Grigor'evič Sinai. *Ergodic theory*, volume 245. Springer Science & Business Media, 2012.
22. Robert M. Yerkes and John D. Dodson. The relation of strength of stimulus to rapidity of habit-formation. *Journal of Comparative Neurology and Psychology*, 18(5):459–482, 1908.