A preliminary experimentation for large scale epidemic forecasting simulations

Gianfranco Lombardo, Agostino Poggi

Dipartimento di Ingegneria e Architettura, Università di Parma - Parma, Italy

Abstract

Agent-based modeling and simulation are some powerful techniques that are widely used with success for analyzing complex and emergent phenomena in many research and application areas. Many different reasons are behind the success of such techniques, among which an important mention goes to the availability of a great variety of software tools, that ease the development of models, as well as the execution of simulations and the analysis of results. This paper presents an actor software library, called ActoDeS, for the development of concurrent and distributed systems, and shows how it can be a suitable mean for building flexible and scalable epidemic forecasting simulations. In particular, the paper presents the first results of the experimentation of ActoDeS for defining a COVID-19 epidemic diffusion model and for supporting the simulation in large populations.

Keywords

actor model, epidemic diffusion model, COVID-19, distributed simulation, HPC

1. Introduction

Agent-based modeling and simulation (ABMS) has been and is widely used with success for studying complex and emergent phenomena in many research and application areas, including agriculture, biomedical analysis, ecology, engineering, sociology, market analysis, artificial life, social studies, and others fields. Such studies are possible thanks to the availability of several tools and libraries that support the development of ABMS applications. Moreover, the availability of large-scale, dynamic, and heterogeneous networks of computational resources and the advent of multi-cores computers allow the development of high performance and scalable computationally intensive ABMS applications. As a matter of fact, such applications are able to manage very large and dynamic models, whose computational needs (in space and time) can be difficult to satisfy by a single machine.

The success and the diffusion of ABMS techniques is also due to the availability of software tools that ease the development of models, the execution of simulations and the analysis of results (see, for example, Mason [1], NetLogo [2], and Repast [3]). However, all the most known and used ABMS tools have been initially designed for the execution of simulations on a single machine and, only in a second step, they were extended for supporting distributed simulations (see, for example, D-Mason [4] and HLA_ACTOR_REPAST [5]). It is worth noting that such

WOA 2020: Workshop "From Objects to Agents", September 14–16, 2020, Bologna, Italy
gianfranco.lombardo@unipr.it (G. Lombardo); agostino.poggi@unipr.it (A. Poggi)
0000-0003-1808-4487 (G. Lombardo); 0000-0003-3528-0260 (A. Poggi)

extensions show a limitation in terms of reusability. In fact, the code of agents defined for a standalone execution must be modified in order to gain suitable implementations to be used in a distributed simulation.

In this paper we present an actor based software library, called ActoDeS, that provides the suitable features for simplifying the development of scalable and efficient agent based models and simulations. The next section introduces the actor model and discusses the advantages of using actors in ABMS applications. Section 3 introduces ActoDeS. Section 4 shows the advantages of using this software library for ABMS applications. Section 5 presents the COVID-19 diffusion model that is used in the simulations. Section 6 presents the results of the experimentation on the population of Bergamo province. Section 7 introduces the work necessary to extend the simulation to all the population of Lombardia region. Finally, section 8 concludes the paper by discussing its main features and the directions for future work.

2. ABMS with acotrs

Actors [6] are autonomous concurrent objects, which interact with other actors by exchanging asynchronous messages and which, in response of an incoming message, can perform some tasks, namely, sending messages to other actors, creating new actors, and designating a new behavior for processing the messages that the actor will receive.

Actors have the suitable features for defining agent models that can be used in ABMS applications and for modeling the computational agents found in MAS and DAI systems [7]. In fact, actors and computational agents share certain characteristics: i) both react to external stimuli (i.e., they are reactive), ii) both are self-contained, self-regulating, and self-directed, (i.e., they are autonomous and can be goal directed), and iii) both interact through asynchronous messages and such messages are the basis for their coordination and cooperation (i.e., they are social).

Moreover, the use of messages for exchanging state information decouples the code of agents. In fact, agents do not need to access directly to the code of the other agents (i.e., their methods) to get information about them, and so the modification of the code of a type of agent should cause smaller modifications in the code of the other types of agent. Finally, the use of actors simplifies the development of real computational agents in domains where, for example, they need to coordinate themselves or cooperate through direct interactions.

Different researchers propose the use of actors for agent based simulation. For example, Jang and Agha [8] proposed an actor-based software infrastructure, called adaptive actor architecture, to support the construction of large-scale agent-based simulations, by exploiting distributed computing techniques to efficiently distribute agents across a distributed network of computers. In particular, this software infrastructure uses some optimizing techniques in order to reduce the amount of exchanged data among nodes and to support dynamic agent distribution and search.

Cicirelli et al. [5] propose the use of actors for distributing Repast simulations. In particular, they defined a software infrastructure that allows: the migration of agents, a location transparent naming, and an efficient communication. This architecture allows the decomposition of a large system into sub-systems (theatres), each hosting a collection of application actors, that can be allocated on different computational nodes.

Finally, de Berardinis et al. [9] propose an actor based model which allows users to get an idea of the impact of a mobility initiative prior to deployment is a complex task for both urban planners and transport companies.

3. ActoDeS

ActoDeS is an actor-based software framework that has the goal of both simplifying the development of concurrent and distributed complex systems and guarantying an efficient execution of applications [10]. ActoDeS is implemented by using the Java language and is an evolution of CoDE [11] that simplifies the definition of actor behaviors and provides more scalable and performant implementations. Moreover, it takes advantages of some implementation solutions used in JADE [12],[13],[14] for the definition of some internal components. In particular, it has been used for the development of data analysis tools [15] and their use for the analysis of social networks data [16],[17],[18],[19].

ActoDeS has a layered architecture composed of an application and a runtime layer. The application layer provides the software components that an application developer needs to extend or directly use for implementing the specific actors of an application. The runtime layer provides the software components that implement the ActoDeS middleware infrastructures to support the development of standalone and distributed applications.

In ActoDeS an application is based on a set of interacting actors that perform tasks concurrently and interact with each other by exchanging asynchronous messages. Moreover, it can create new actors, update its local state, change its behavior and kill itself.

Depending on the complexity of the application and on the availability of computing and communication resources, one or more actor spaces can manage the actors of the application. An actor space acts as "container" for a set of actors and provides them the services necessary for their execution. An actor space contains a set of actors (application actors) that perform the specific tasks of the current application and two actors (runtime actors) that support the execution of the application actors. These two last actors are called executor and the service provider. The executor manages the concurrent execution of the actors of the actor space. The service provider enables the actors of an application to perform new kinds of action (e.g., to broadcast a message or to move from an actor space to another one).

Communication between actors is buffered: incoming messages are stored in a mailbox until the actor is ready to process them; moreover, an actor can set a timeout for waiting for a new message and then can execute some actions if the timeout fires. Each actor has a system-wide unique identifier called reference that allows it to be reached in a location transparent way independently of the location of the sender (i.e., their location can be the same or different). An actor can send messages only to the actors of which it knows the reference, that is, the actors it created and of which it received the references from other actors. After its creation, an actor can change several times its behavior until it kills itself. Each behavior has the main duty of processing a set of specific messages through a set of message handlers called cases. Therefore, if an unexpected message arrives, then the actor mailbox maintains it until a next behavior will be able to process it.

An actor can be viewed as a logical thread that implements an event loop [20],[21]. This event loop perpetually processes incoming messages. In fact, when an actor receives a message, then it looks for the suitable message handler for its processing and, if it exists, it processes the message. The execution of the message handler is also the means for changing its way of acting. In fact, the actor uses the return value of its message handlers for deciding to remain in the current behavior, to move to a new behavior or to kill itself. Moreover, an actor can set a timeout within receiving a new message and set a message handler for managing the firing of the timeout. This message handler is bound to the reception of the message notifying the firing of the timeout, and so the management of the timeout firing is automatically performed at the reception of such notification message.

ActoDeS supports the configuration of applications with different actor, scheduler ad service provider implementations. The type of the implementation of an actor is one of the factors that mainly influence the attributes of the execution of an application. In particular, actor implementations can be divided in two classes that allow to an actor either to have its own thread (from here named active actors) or to share a single thread with the other actors of the actor space (from here named passive actors). Moreover, the duties of a scheduler depend on the type of the actor implementation. Of course, a scheduler for passive actors is different from a scheduler for active actors, but for the same kind of actor can be useful to have different scheduler implementations. For example, it can allow the implementation of "cooperative" schedulers in which actors can cyclically perform tasks whose duties vary from the processing of the first message in the buffer to the processing of all the messages in it.

The most important decision that influences the quality of the execution of an application is the choice of the actor and scheduler implementations. In fact, the use of one or another couple of actor and scheduler causes large differences in the performance and in the scalability of the applications [22].

4. ABMS with ActoDeS

The features of the actor model and the flexibility of its implementation make ActoDeS suitable for building agent-based modelling and simulation (ABMS) applications and for analyzing the results of the related simulations. In fact, the use of active and passive actors allows the development of applications involving large number of actors, and the availability of different schedulers and the possibility of their specialization allow an efficient execution of simulations in application domains that require different types of scheduling algorithms [23].

In particular, ActoDeS offers a very simple scheduler that may be used in a large set of application domains and, in particular, in ABMS applications. Such a scheduler manages agents implemented as passive actors and its execution repeats until the end of the simulation the following operations:

1. Sends a "step" message to all the agents and increments the value of "step";
2. Performs an execution step of all the agents.

In particular, the reception of a "step" message allows agents to understand that they have all the information (messages) for deciding their actions; therefore, they decide, perform some actions and, at the end, send information to other agents.

4.1. Distribution

The modeling and simulation of complex problems can require the use of large number of agents that may determinate unacceptable simulation times, or the impossibility to run the simulation on a single computational node. In such cases, the availability of distributed algorithms and, of course, of an adequate number of computational nodes can help to perform simulations, partitioning the agents among the available computational nodes. Moreover, depending of kind of application, agents may need to communicate with some agents running on different computational nodes. In this case, the execution of the computational nodes must be synchronized to guaranty the processing of the messages in the correct order.

Therefore, a distributed simulation involves a set of computational nodes whose execution is driven by a set of schedulers and managers. In particular, each manager has the duty of creating the subset of agents of its computational node and synchronize the execution of the simulation with the execution of the other computational nodes. Moreover, one of such managers assumes the role of "master" and has the duties of partitioning the agents involved in the simulation and sends the information necessary for the creation of the agents to the other managers.

In particular, the execution of a distributed simulation can be described by the following steps:

1. Master manager partitions the agents and sends the information for creating a subset of agents to each scheduler (including itself).
2. Managers create all the actors of its subset.
3. Repeat until the end of the simulation:
    - Managers send a synchronization message to the other managers and wait for the corresponding messages from them.
    - Schedulers perform an execution step of all their actors.
    - Scheduler send a "end step" message to all their actors and managers.

4.2. Communication

As introduced above, in different kinds of simulations the agents need to exchange data and often such an interaction is not localized, therefore, the partition of agents on several computational nodes may add an important communication cost. An important solution to reduce the cost of communication is to reduce the frequency of interactions, merging multiple interactions to one. In a conservative distributed simulation system, that synchronizes the simulation step of all the computational nodes involved in the simulation, a possible solution is to group all the messages directed to the agents of a specific computational node into the message that identifies the successive synchronization message.

5. COVID-19 epidemic diffusion model

In the last years, several computational models for the simulation of epidemic outbreaks have been used with increased frequency, moreover, the current COVID-19 pandemic has increase the interest and work on the definition and experimentation of such models [24],[25],[26]. The aim of our work is the definition of a model that allows to identify exactly how many people had actually contracted the COVID-19 virus at the beginning of the pandemic and try to simulate the real evolutionary trend of the spread of the virus that has produced such a large number of people positive for the virus. The first experimentation involved the Bergamo province, but our final goal is to simulate the propagation in the Lombardia region.

The basic idea was to model people based on their social interactions, which can vary from person to person, estimated interactions with a particular rate. In our first model, people were identified on the basis of an interaction rate (high, medium and low) and in a random way these people came into contact, creating a network of interactions.

At the beginning, all the people are in a susceptible phase, in this phase each person can be infected by an infected person. When a person is infected, she/he passes from a susceptibility phase to an incubation phase; here she/he remains there for a certain period of time and then passes to another phase in which he is infectious. When a person is in this phase he could infect other people. Once this last phase is over, the infected changes to be positive. After a certain period of time, the person changes phase: heals or dies. When a person heals, she/he can no longer be infected. Moreover, the incubation phase is made to last from 7 to 14 days, that infectious from 3 to 7 and that of positivity from 14 to 30.

In our model, each person is defined by the following attributes:

- Id of the person
- Province of belonging
- Degree of interaction
- Number of people met
- Current phase
- Id of the people met

This model was used for simulating the COVID-19 propagation from January 30, 2020 to March 31, 2020. Of course, the interaction frequency varies for all the people during the simulation because some people become positive at the end of the phase in which they are infectious; therefore, their interactions should collapse enough sudden because they should stay in quarantine. Moreover, from March 8, 2020, people reduced their interaction frequency because of the lockdown rules.

6. Experimentation and Results

In order to test the model for the Bergamo province, multiple tests were carried out. Thanks to them, it was gradually understood which parameters had to be modified to obtain a trend similar to that of the contagion curve in the real case of Bergamo province. In particular, we considered two "key" dates: March 8th (pre-lockdown date) and March 31st 2020. We focused on the parameter that represents the value of the positive swabs made. In summary, there were 997 positives on March 8 and 8803 positives on March 31. Initially, the parameter that regulates infections has been kept fixed at 0.7, given, found in the literature, which should represent the average value of infection without a mask. The use of such parameter allowed to get good results because the average number of positives obtained by ten simulations was 906. Of course

Table 1
Positives numbers for different infection parameters.

			March 8,2020			
Infection value	0.3	0.4	0.5	0.7	1	Real positives
Positives number	563	674	715	906	1297	997
			March 31,2020			
Infection value	0.3	0.4	0.5	0.7	1	Real positives
Positives number	8307	15127	22025	43304	89274	8803

such parameter value is not good for the lookdown period and so a quite good number of positives was obtained fixing the infection parameter at 0.3. In this case, average number of positives obtained by ten simulations was 8370. Table 1 shows the positives number in the two “key” dates for different infection parameters and compares them with the real number of positives.

7. Scaling to large populations

The final goal of our work is the simulation of epidemic outbreaks in the entire Lombardia. However, the number of people living in Lombardia is more than ten million and so its simulation is feasible only with a distributed implementation. The simplest solution is to partition people on the basis of their province. This solution allows to use different parameters in difference provinces, but implies an unbalanced load among the different computational nodes and it can be a big problem if the simulation should require the synchronization of the execution steps.

Therefore, we decided to combine two kinds of partition: i) people is partitioned on the basis of their province and so each person uses the parameter values associated with her/his province, and ii) people is divided in subset of equivalent size by using the number of computational nodes involved in the simulation. While the management of Lombardia population should be possible without problems, thank to the availability of the computational nodes of the High Performance Computing lab of our university, some problems can be raised by the number of the messages necessary to create consistent sets of contacts in each step of the simulation (i.e., when a person A identifies a person B as her/his contact, then she/he needs to inform person B that she/he in her/his contacts).

Currently, we are evaluating several solutions to reduce the communication overhead, some solutions avoid the exchange of messages, by saving for each step, the data of the people of each computational node on a file and then merging the data of the different files associated with this step, other solutions requires the sending by messages, but at each step a computational node sends a message to all the other computational nodes. Each message contains all the contact information for the people of a specific computational node.

8. Conclusions

This paper presented ActoDeS, an actor software library for the development of concurrent and distributed systems, and shows how it can be a suitable mean for building flexible and scalable epidemic forecasting simulations. In particular, the paper presented the first results of the experimentation of ActoDeS for forecasting COVID-19 epidemic diffusion in the Bergamo province and introduced the first work for defining a scalable distributed system able to forecast COVID-19 epidemic diffusion for all the population of Lombardia region. Our current and future work will be dedicated to complete the design and the implementation of such a distributed system, to experiment different communication solutions for the updating of the contacts of each person, and finally, the definition of more accurate person model that replaces the use of the simple interaction frequency to create new contacts, with the use of the person age and the work sector.

References

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

[2] S. Tisue, U. Wilensky, Netlogo: A simple environment for modeling complexity, in: International conference on complex systems, volume 21, Boston, MA, 2004, pp. 16–21.

[3] 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 (2006) 1–25.

[4] G. Cordasco, R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, Bringing together efficiency and effectiveness in distributed simulations: the experience with d-mason, Simulation 89 (2013) 1236–1253.

[5] F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro, Hla_actor_repast: An approach to distributing repast models for high-performance simulations, Simulation Modelling Practice and Theory 19 (2011) 283–300.

[6] G. A. Agha, Actors: A model of concurrent computation in distributed systems., Technical Report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1985.

[7] D. Kafura, J.-P. Briot, Actors and agents, IEEE concurrency (1998) 24–28.

[8] M.-W. Jang, G. Agha, Agent framework services to reduce agent communication overhead in large-scale agent-based simulations, Simulation Modelling Practice and Theory 14 (2006) 679–694.

[9] J. de Berardinis, G. Forcina, A. Jafari, M. Sirjani, Actor-based macroscopic modeling and simulation for smart urban planning, Science of Computer Programming 168 (2018) 142–164.

[10] F. Bergenti, E. Iotti, A. Poggi, M. Tomaiuolo, Concurrent and distributed applications with actodes, in: MATEC Web of Conferences, volume 76, EDP Sciences, 2016, p. 04043.

[11] F. Bergenti, A. Poggi, M. Tomaiuolo, An actor based software framework for scalable applications, in: International Conference on Internet and Distributed Computing Systems, Springer, 2014, pp. 26–35.

[12] A. Negri, A. Poggi, M. Tomaiuolo, P. Turci, Dynamic grid tasks composition and distribution through agents, Concurrency and Computation: Practice and Experience 18 (2006) 875–885.

[13] G. Adorni, F. Bergenti, A. Poggi, G. Rimassa, Enabling fipa agents on small devices, in: International Workshop on Cooperative Information Agents, Springer, 2001, pp. 248–257.

[14] A. Poggi, M. Tomaiuolo, G. Vitaglione, A security infrastructure for trust management in multi-agent systems, in: Trusting Agents for Trusting Electronic Societies, Springer, 2004, pp. 162–179.

[15] G. Lombardo, P. Fornacciari, M. Mordonini, M. Tomaiuolo, A. Poggi, A multi-agent architecture for data analysis, Future Internet 11 (2019) 49.

[16] F. Bergenti, E. Franchi, A. Poggi, Agent-based social networks for enterprise collaboration, in: 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE, 2011, pp. 25–28.

[17] G. Lombardo, P. Fornacciari, M. Mordonini, L. Sani, M. Tomaiuolo, A combined approach for the analysis of support groups on facebook-the case of patients of hidradenitis suppurativa, Multimedia Tools and Applications 78 (2019) 3321–3339.

[18] G. Lombardo, A. Ferrari, P. Fornacciari, M. Mordonini, L. Sani, M. Tomaiuolo, Dynamics of emotions and relations in a facebook group of patients with hidradenitis suppurativa, in: International Conference on Smart Objects and Technologies for Social Good, Springer, 2017, pp. 269–278.

[19] M. Tomaiuolo, G. Lombardo, M. Mordonini, S. Cagnoni, A. Poggi, A survey on troll detection, Future Internet 12 (2020) 31.

[20] J. Dedecker, T. Van Cutsem, S. Mostinckx, T. D'Hondt, W. De Meuter, Ambient-oriented programming in ambienttalk, in: European Conference on Object-Oriented Programming, Springer, 2006, pp. 230–254.

[21] M. S. Miller, E. D. Tribble, J. Shapiro, Concurrency among strangers, in: International Symposium on Trustworthy Global Computing, Springer, 2005, pp. 195–229.

[22] A. Poggi, Agent based modeling and simulation with actomos., in: WOA, 2015, pp. 91–96.

[23] P. Mathieu, Y. Secq, et al., Environment updating and agent scheduling policies in agent-based simulators., in: ICAART (2), 2012, pp. 170–175.

[24] M. Ajelli, B. Gonçalves, D. Balcan, V. Colizza, H. Hu, J. J. Ramasco, S. Merler, A. Vespignani, Comparing large-scale computational approaches to epidemic modeling: agent-based versus structured metapopulation models, BMC infectious diseases 10 (2010) 190.

[25] D. Chumachenko, V. Dobriak, M. Mazorchuk, I. Meniailov, K. Bazilevych, On agent-based approach to influenza and acute respiratory virus infection simulation, in: 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), IEEE, 2018, pp. 192–195.

[26] M. Gatto, E. Bertuzzo, L. Mari, S. Miccoli, L. Carraro, R. Casagrandi, A. Rinaldo, Spread and dynamics of the covid-19 epidemic in italy: Effects of emergency containment measures, Proceedings of the National Academy of Sciences 117 (2020) 10484–10491.