Agent based Cellular Automata Simulation

P. Fornacciari, G. Lombardo, M. Mordonini, A. Poggi and M. Tomaiuolo
Dipartimento di Ingegneria e Architettura
Università degli Studi di Parma
Parma, Italy
{paolo.fornacciari,gianfranco.lombardo,monica.mordonini,agostino.poggi,michele.tomaiuolo}@unipr.it

***Abstract* — This paper presents an actor-based software library, called Actomata, for the definition and simulation of cellular automata models. Using Actomata, each cell of a model is defined by an actor and the evolution of its state is built though the interaction with their neighbor cells via the exchange of messages. This kind of implementation simplifies the definition of the code driving the behavior of each cell and the distribution of simulations on a set of different computational nodes.**

Keywords — actor model, cellular automata modelling, distributed simulation.

I. Introduction

Cellular automata are abstract and discrete computational systems that have proved suitable for modeling complex system representations in several scientific fields. Cellular automata models are composed of a grid of cells where each cell has a given "state" which can have a discrete time evolution defined through the interaction with their neighbor cells. This evolution is obtained through some simulation algorithms whose implementations are in general serial, since that is enough to represent many systems of interest; however, in some cases models involve a massive number of cells and complex algorithms to compute the next "state" of each cell and so the performance of simulations become very poor. Therefore, to cope with such a kind of problem, different researchers propose some parallel and distributed cellular automata simulation algorithms [1][2].

This paper presents a software library, called Actomata, that offers a set of features useful for simplifying the development of cellular automata models and for performing scalable and distributed simulations. Actomata has been implemented on the top of ActoDeS, an actor-based software framework aimed at both simplifying the development of large and distributed complex systems and guarantying an efficient execution of applications. Section 2 shortly introduces the ActoDeS software framework. Section 3 presents Actomata and, in particular, shows the features that make it suitable for developing cellular automata models and for performing their simulation. Section 4 discusses its experimentation in the modelling and simulation of evolution, epidemic and evacuation models. Finally, Section 5 summarizes the results and points out to future research.

II. 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 [3].

ActoDeS is implemented by using the Java language and takes advantage of preexistent Java software libraries and solutions for supporting concurrency and distribution. 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 [4]. 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

CEUR-WS.org/Vol-2215/paper_3.pdf

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 [5][6]. 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 [7].

III. ACTOMATA

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 [8]. 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 [9].

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 (broadcast) the information about their new state to the interested agents.

Automata is an actor-based software library that has been implemented on the top of ActoDeS with the goal of simplifying the definition of cellular automata models and to perform their simulation. Some of the features useful to achieve this goal are already provided by ActoDeS, but Automata makes easy the definition of neighborhoods and then exchange of messages among the cells of an automata (i.e., messages are directed to the cell neighborhood and not to its single members), and improves the performance of their simulations by providing a distributed simulator that makes transparent the partition of the cells and the communication between cells on different computational nodes.

A. *Cell Model*

The cells can be modelled as a finite state machine where each state is defined by an actor behavior that process the input messages through two message handlers: the first handler processes the messages informing about the state of the cells defining its neighborhood, and the second handler processes the "step" messages computing the new state of the cell and by sending the information about the new state to the cells in its neighborhood.

B. *Neighborhood and Communication*

Actomata provides a set of simple classes for the definition of different types of neighborhood. In particular, it provides a set of classes for the definition of the classical Von Neumann and Moore neighborhoods with different radius and a set of abstract classes for the definition of specialized neighborhoods. Moreover, each neighborhood can be defined by two different classes that allow to transparently receive all the messages sent by the cells through either point-to point messages or broadcast messages. In fact, when the neighborhood of the cells is very large, the overhead given to the exchange of messages becomes very high and then the use of broadcast messages becomes convenient. In this last case, a cell receives messages from all the other cells, and the object implementing its neighborhood acts as filter for the incoming messages. Finally, the number of messages can be reduced by eliminating the sending of messages when a cell is in a particular (default) state (e.g., the "dead" cells in the game of life simulation and the "empty" cells in the prey-predator simulation).

C. Distribution

The modeling and simulation of complex problems can require the use of large number of actors 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 perform simulations. We are working on the definition and implementation of distributed simulation algorithms and, in particular, we defined a type of scheduler whose instances can cooperate for the execution of distributed cellular automata simulations. This kind of scheduler does not require modifications on the code of the actors used for the standalone simulations (because, as introduced above, messages are transparently sent to remote actors), but requires the partition of the actors among the different computational nodes of the simulation. In particular, to reduce the propagation of messages among the different computational nodes, the scheduler of each node manages the actors representing a rectangular area of the grid defining the cellular automata.

A distributed simulation involves a set of schedulers that create the actors representing the cells in the rectangle of their competence, acquire the references of the actors, managed by other schedulers, but in the neighborhood of their cells (border cells), and then execute the simulation. One of such scheduler assumes the role of "master" and has the initial duty of partitioning the rectangle, that represent the global space, in a number of rectangles equal to the number of the actor spaces involved in the simulation. To do it, the master takes advantage of an algorithm that divides a rectangle in a set of "rectangles" with similar area and minimal perimeter [10].

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

1. Master scheduler partitions the global rectangle and sends a rectangle to each scheduler (including itself).
2. Schedulers create all the actors representing the cells positioned in its rectangle.
3. Schedulers build the neighborhoods:
   a. Define the neighborhoods and add the appropriate local actor references.
   b. Ask to the other schedulers the references of the actors necessary for completing the neighborhood of their cells (i.e., the border cells) and send the references to complete the neighborhoods of the cells managed by the other schedulers
5. Schedulers repeat until the end of the simulation:
   a. Send a synchronization message to the other schedulers and wait for the corresponding messages from them.
   b. Send a "step" message to all their actors and increment the value of "step".
   c. Perform an execution step of all their actors.

One of advantages of this kind of distributed simulation system is the limited cost of the propagation of "border cell" messages and the fact that this propagation is not a duty of the schedulers. In fact, the schedulers exchange the references of the actors of the border cells for completing their neighborhood before the beginning of the simulation and then each actor can send messages to the neighbor actors without taking care if they are local or remote actors. Another important feature is the partition of the "space" of the cellular automata in "rectangles" with similar area and minimal perimeter. The definition of rectangles with similar area can help in guaranteeing a good load balancing. The definition of rectangles with minimal perimeter reduces the number of "border cells" and so the number of messages exchanged between the computational nodes of the distributed simulation system.

IV. Experimentation

We experimented Actomata in the lab activities of a master course on distributed systems. The experimentation followed two phases. In the first phase, students modelled and simulated some very simple problems, i.e., an age-structured predator prey model and an epidemic diffusion model [11][12]. In the second phase, students tried to define and simulate a model for the evacuation from a simple room, starting from two articles that illustrated two solutions for such a kind of problem [13][14]. The result of the first phase of the experimentation was that all the students spent very few time for finding a good solution for the age-structured predator prey and epidemic diffusion models. However, the second phase of the experimentation provided a heterogeneous set of results because, of course, the problem is more complex respect to the ones proposed in the first phase and because some students found difficulties to "extract" the algorithms proposed by the authors of the selected articles. Therefore, all the students spent a long time, some found good or acceptable solutions, others proposed solutions with some problems (e.g., part of occupants did not exit from the room) and finally a small number of students do not complete the implementation of their solution.

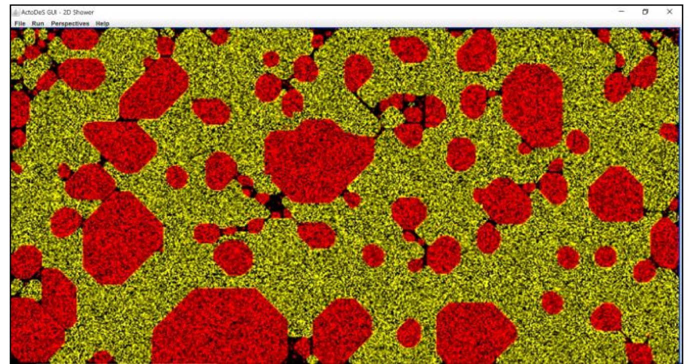

Figure 1. Graphical view of a step of the age-structured predator prey problem

Another part of the experimentation was done to measure the advantages of a distributed simulation by using one of the solutions proposed by the student for modelling the age-structured predator prey problem. Table 1 presents the execution times of the simulations with a length of a hundred of cycles and distributed on one, two and four computational nodes. These results were obtained on some laptops with an Intel Core 2 - 2.80GHz processor, 8 GB RAM, Windows 10

OS and Java 8 with 4 GB heap size. Figures 1 and 2 respectively show a view of a step of the age-structured predator prey problem (involving sharks as predators and fishes as preys), and the code of the shark behavior. In this code, the “process” method is called when the shark received a message from a fish or from another shark, and the “update” method is called when it receives a step message from the scheduler.

TABLE 1

Actors	Time(ms)		
62.500 (250x250)	15.700	10.529	7.574
125.000 (500x250)	39.259	26.716	18.681
250.000 (500x500)	80.433	62.737	43.916
500.000 (1000x500)	286.666	198.769	137.457
1.000.000 (1000x1000)	1232.665	863.865	624.293
Units	1	2	4
Simulation Steps	100		

```
public final class Shark extends SeaCell
{
  . . . class and instance variables . . .

  public Shark(final int x, final int y,
      final List<Reference> n, final MessagePattern p) {
    super(x, y, n, p);
  }

  public Shark(final SeaCell c) {
    super(c);
  }

  public void process(final Message m) {
    if (m.getContent().equals(Shark.class.getName())) {
      this.sharks++;
    }
    else if (m.getContent().equals(Fish.class.getName())) {
      this.fishes++;
    }
  }

  public Behavior update() {
    this.age++;

    if ((this.age == MAXAGE)
        || (RANDOM.nextDouble() < PROBDEAD)
        || (this.fishes > MINFISHES)) {
      new Empty(this);
    }

    this.sharks = 0;
    this.fishes = 0;

    return null;
  }
}
```

Figure 2. Shark behavior code

V. CONCLUSIONS

This paper presented Actomata, a software library that tries to simplify the development of cellular automata models and to improve the performance of its simulations. Actomata is implemented on the top of ActoDeS, that is an actor-based software framework aimed at both simplifying the development of large and distributed complex systems and guarantying an efficient execution of applications [1]. Actomata has been experimented with success in the development of cellular automata models and simulations. This initial experimentation coped with modelling of simple and well-known problems.

Several software tools can be used for cellular automata modelling and simulation; in particular, the most known ABMS platforms (i.e., NetLogo [15], Repast [16] and MASON [17]) support it. Actomata does not offer all the features of such platforms. Its main feature is the use of the actor model for the definition of cellular automata. It allows the definition of models where cells interact through the exchange of messages simplifying the development of non-trivial applications where the management of concurrent activities may be of primary importance. Moreover, the availability of techniques to reduce the overhead of the diffusion of broadcast and multicast messages and the use of a distributed simulation allow to maintain good performances even if the cellular automata involved large number of cells. Finally, the actor implementations offered by Actomata make transparent the communication between local and remote actors and it allows to use the same “cell code” both in standalone and distributed simulations.

Current and future research activities are and will be dedicated to extend the experimentation to more complex problems. In particular, part of the experimentation was oriented to evaluate the performances of cellular automata where their cells have neighborhoods of different sizes and exchange either point-to point messages or broadcast messages. In fact, one of the main problems of the use of actors for the simulation of cellular automata is the huge number of messages that are exchanged during the simulation of models that involve a large number of actors and large neighborhoods. In this case is possible to use the broadcast of messages, but in some situations each actor needs to checks very long list of messages to find the ones belonging to its neighborhood. Therefore, an important part of the future research will be oriented to check if a localized broadcast based, for example, on a bin-lattice structure [18], or a multicast, associated with each neighborhood, could extend the advantage of the use of broadcast for very large neighborhoods also in distributed simulations. Moreover, some of the work will be dedicated to simplify the development of models; in particular, given that with Automata cells are modelled as a finite state machine where each state is defined by an actor behavior, then their structure can be defined by an UML state diagram that can be used for generating the code defining the structure of the model of the cells [19][20].

REFERENCES

[1] P. Richmond, D. Walker, S. Coakley, and D. Romano. "High performance cellular level agent-based simulation with FLAME for the GPU," Briefings in Bioinformatics, Vol. 11, No. 3, pp. 334-347, 2010.

[2] P.M.A. Sloot, J.A. Kaandorp, A.G. Hoekstra, and B.J. Overeinder. "Distributed cellular automata: Large scale simulation of natural phenomena," Solutions to Parallel and Distributed Computing Problems, Lessons from Biological Sciences, 2001, pp. 1-46.

[3] F. Bergenti, E. Iotti, A. Poggi, and M. Tomaiuolo, “Concurrent and Distributed Applications with ActoDeS,” in MATEC Web of Conferences, Vol. 76, pp 1-8, 2016.

[4] G.A. Agha, “Actors: A Model of Concurrent Computation in Distributed Systems,” Cambridge, MA, USA: MIT Press, 1986.

[5] J. Dedecker, T. Van Cutsem, S. Mostinckx, T. D’Hondt and W. De Meuter, “Ambient-oriented programming in ambienttalk,” in ECOOP 2006 – Object-Oriented Programming, Berlin, Germany: Springer, 2006, pp. 230-254.

[6] M. S. Miller, E. D. Tribble, and J. Shapiro, “Concurrency among strangers,” in Trustworthy Global Computing, Berlin, Germany: Springer, 2005, pp. 195-229.

[7] F. Bergenti, A. Poggi, and M. Tomaiuolo, "An Actor Based Software Framework for Scalable Applications," in Internet and Distributed Computing Systems, Berlin, Germany: Springer, 2014, pp. 26-35.

[8] A. Poggi, “Agent based modeling and simulation with ActoMoS,” in Proc. 16th Workshop on From Object to Agents (WOA 2015), Naples; Italy, 2015, pp. 91-96.

[9] P. Mathieu, and Y. Secq, “Environment Updating and Agent Scheduling Policies in Agent-based Simulators,” in Proc. 4th Int. Conf. on Agents and Artificial Intelligence, Algarve, Portugal, 2012, pp. 170-175.

[10] N. Alon, and D. J., Kleitman, “Partitioning a rectangle into small perimeter rectangles,” Discrete mathematics, Vol. 103, No. 2, pp. 111-119, 1992.

[11] G.B. Ermentrout, and L. Edelstein-Keshet. "Cellular automata approaches to biological modeling," Journal of theoretical Biology, Vol. 160, No. 1, pp. 97-133, 1993.

[12] S.H. White, A.M. Del Rey, and G.R. Sánchez. "Modeling epidemics using cellular automata," Applied Mathematics and Computation, Vol. 186, No. 1, pp. 193-202, 2007.

[13] A.Varas, M.D. Cornejo, D. Mainemer, B. Toledo, José Rogan, V. Munoz, and J. A. Valdivia. "Cellular automaton model for evacuation process with obstacles," Physica A: Statistical Mechanics and its Applications, Vol. 382, No. 2, pp. 631-642, 2007.

[14] R. Alizadeh. "A dynamic cellular automaton model for evacuation process with obstacles," Safety Science, Vol. 49, No. 2, pp. 315-323, 2011.

[15] S. Tisue, and U. Wilensky, “Netlogo: A simple environment for modeling complexity,” in Proc. of Int. Conf. on Complex Systems (ICCS 2004), 16-21, Boston, MA, USA, 2004, pp. 16-21.

[16] M. J. North, N. Collier, and J. Vos, “Experiences in creating three implementations of the repast agent modeling toolkit,” ACM Transactions on Modeling and Computer Simulation, vol. 16, no. 1, pp. 1-25, 2006.

[17] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, “MASON: A multiagent simulation environment,” Simulation, vol. 81, no. 7, pp. 517-527, 2005.

[18] C.W. Reynolds, ”Interaction with groups of autonomous characters,” in Game Developers Conference, Vol. 2000, p. 83, 2000.

[19] F.Bergenti, and A. Poggi, “A development toolkit to realize autonomous and interoperable agents,” in Proceedings of the fifth international conference on Autonomous Agents, 2001, pp. 632-639.

[20] F. Bergenti, A. Poggi, “Exploiting UML in the design of multi-agent systems,” in International Workshop on Engineering Societies in the Agents World, Berlin, Germany: Springer, 2000, pp. 106-113.