A Study on the Parallelization of Terrain-Covering Ant Robots Simulations

Alessandro Pellegrini and Francesco Quaglia

DIAG, Sapienza, University of Rome

Abstract. Agent-based simulation is used as a tool for supporting (time-critical) decision making in differentiated contexts. Hence, techniques for speeding up the execution of agent-based models, such as Parallel Discrete Event Simulation (PDES), are of great relevance/benefit. On the other hand, parallelism entails that the final output provided by the simulator should closely match the one provided by a traditional sequential run. This is not obvious given that, for performance and efficiency reasons, parallel simulation engines do not allow the evaluation of global predicates on the simulation model evolution with arbitrary time-granularity along the simulation time-axis. In this article we present a study on the effects of parallelization of agent-based simulations, focusing on complementary aspects such as performance and reliability of the provided simulation output. We target Terrain Covering Ant Robots (TCAR) simulations, which are useful in rescue scenarios to determine how many agents (i.e., robots) should be used to completely explore a certain terrain for possible victims within a given time.

1 Introduction

Thanks to the expressive power intrinsically exhibited by agent-based models, agent-based simulation constitutes a proven solution to study complex real-world scenarios. In these models, agents exhibit individual or collective interactions, which have been shown to reliably express interactions between different objects/entities in real world phenomena such as disaster rescue [1], computational sociology [2], logistics [3], biomedical applications [4], and economic analysis [5].

On the other hand, for several application domains, one core aspect to cope with is the timeliness according to which the simulation system is able to deliver its outputs. One example is related to supporting (time-critical) decision making [6] via, e.g., what-if analysis carried out through agent-based simulation models, such as when employing agent-based models in disaster-rescue contexts. To achieve timeliness in the delivery of simulation outputs, parallelization techniques have been adopted with success in several fields (see, e.g., [7]).

Nevertheless, one peculiar aspect of agent-based simulation models is that they are employed not only to study steady state or equilibrium properties of a system, rather to determine the exact simulated-time when a given predicate becomes true. High precision in the determination of such a time instant would require frequent inspection of the state of the simulation model (ideally at each state transition, namely after the execution of each simulation event). This may result feasible in traditional sequential simulation, where the unique running thread may quickly retrieve the information for predicate evaluation from the data structures (representing the simulation model state) within its address space. On the other hand, when employing parallel/distributed simulation techniques, such a fine grain inspection along the simulation-time axis may result inviable due to the overhead for process coordination, which may hamper the achievable speedup. Also, for high performance optimistic parallel simulation, where the computation performed might be subject to rollback due to violations of event-causality caused by speculative processing, the inspection on the simulation model trajectory may be explicitly delayed to the time instant when a given portion of the computation becomes committed (namely no rollback can even occur in that portion of simulated-time while further executing the simulation model). As a consequence, a shift may appear between the simulated time when the parallel run detects that the predicate holds, and the corresponding simulated time when the sequential run tracks the holding of the same predicate.

In in this article we present a study on the tradeoff between performance and reliability of the simulation outputs when exploiting optimistic PDES techniques, particularly the ROOT-Sim PDES platform [8, 9], for the case of simulations of Terrain-Covering Ant Robots (TCAR). More in detail, we study how the frequency of inspection on committed portions of the parallel simulation run impacts both the achievable speedup and the distribution of the estimated simulated-time for fully exploring a target spatial region. This has been done while varying the population of robots (which determines the simulation model complexity), and by relying on parallel runs carried out on a 32-core HP ProLiant machine equipped with 64 GB of RAM, which is representative of current off-the-shelf commodity facilities for scientific computing. Beyond providing results for this specific application context, to the best of our knowledge, this is the first study along the direction of evaluating the reliability of asynchronous (non-time stepped) parallel simulation outputs for the assessment of non-steady state properties of the real-world phenomenon targeted by agent-based simulation.

The remainder of this paper is structured as follows. In Section 2, related work is discussed. The target TCAR model is depicted in Section 3. The parallelization approach for TCAR simulations via ROOT-Sim is presented in Section 4. Experimental data are reported in Section 5.

2 Related Work

In literature, several special-purpose simulation environments to support agent-based simulations have been presented. In the MASON framework [10] special attention was paid to performance, trying to address computing-intensive models (i.e., large scenarios with many agents), and to portability, ensuring reproducibility of results across different hardware architectures. Interfaces can be connected to simulation models, and the simulations can be paused and moved to different computers (which is regarded as a benefit in case of long simulations). A distributed version is also presented [11], which relies on time-stepped synchronization and on the master/slave paradigm. Compared to this work, we focus on performance/reliability tradeoffs for the case of asynchronous (non-time stepped) PDES, which is recognized as a more scalable paradigm for generic simulation-object interaction patterns.

Pandora [12] is a C++-based simulation framework enabling executions in parallel/distributed environments. It features several AI algorithms for supporting agents' decision making, provides python binding (which is a benefit for inexperienced programmers), and is complemented by Cassandra, a visualization tool created to detect spatial-temporal patterns generated by the simulation for post-analysis. On the other hand, we use ROOT-Sim [8, 9], a general-purpose PDES simulation framework, to support agent-based simulation. We do not rely on specific agent-based facilities, rather the simulation model is written in plain ANSI-C, giving the modeler the freedom to implement the logic using general-purpose libraries. This lead us to develop a highly efficient C-based simulation model for TCAR, which represents a stress case for parallel runs due to very reduced event granularity (about 0.7 microseconds on the used architecture).

The work in [12] has studied the efficiency and scalability of agent-based models in the Cloud. The authors analyze how the simulation performance changes when moving the simulation framework and the model from a cluster to the Cloud. We target a different aspect, because we only focus on optimistic simulation on clusters in order to show what is the price (in terms of results accuracy) to be paid for having a considerable performance gain.

The work in [13] recognizes the importance (in terms of performance) of relying on the PDES paradigm to carry on complex and resource-intensive simulations. Differently from our goal, the authors concentrate on the difficulty which might arise when mapping any agent-based model onto the PDES paradigm, and propose a middleware which can easily help in this task.

In [14], the performance of one particular simulation model run on top of GPUs is presented, showing that relying on graphic cards to carry on agent-based simulations can provide a considerable speedup with respect to sequential simulations—passing from hours to seconds. In this work, we address CPU simulation, where general models with any kind of intra-agent synchronization and any change in the size of the simulation state is allowed during the evolution of the model, thus allowing to implement complex scenarios much more easily.

The work in [15] tries to bridge the gap between the results in [13] and [14], proposing a simulation environment which can benefit from the PDES paradigm (run on CPUs) and the availability of a large number of cores in GPUs. In particular, within the PDES event handlers, the simulation-model writer can specify which code blocks can be run on top of GPUs, and then the simulation platform offloads that computation to the graphic cards. In our proposal, we target transparency, i.e. the simulation model writer simply writes the model's logic, and then the simulation framework efficiently runs it in parallel.

Another proposal related to the present study can be found in [7]. It addresses the tradeoff between reliability of the simulation output and simulation speed when running large-scale numerical simulations (such as molecular dynamics simulations) on GPU architectures. The focus of [7] is on the effects of data representation, namely single vs double precision floating point representation, on the speed of GPU based runs. This is orthogonal to the tradeoff we target in our study, namely the one between the overhead for global predicates' evaluation in parallel runs vs the precision in the detection of the simulated-time when the predicate is verified.

Finally, the work in [16] addresses the issue of parallelizing multi-agent path planning via parallelization of a forward search algorithm on GPUs, where the parallelization approach tries to cope with the exponential growth of the search space vs the number of agents. Conversely, our simulation model is targeted at executions on CPUs, and implements a map coverage scenario based on a unique dynamic robot-move plan evolving according to probabilistic rules.

3 Reference Simulation Model

The agent-based simulation model which we have targeted for our experimental study is a variant of the Terrain-Covering Ant Robots (TCAR) model presented in [17]. This type of simulation model is particularly interesting for the assessment of rescue scenarios. In particular, if some kind of accident occurs in a region which is either unknown by the rescuers or altered by the accident itself, the first action in order to actually rescue the victims is to explore the whole region.

The terrain is modeled as an undirected graph, therefore an agent (i.e., an ant robot) is able to move from one space region to another in both directions. This mapping is created by imposing a specific grid on the space region. The agents are then required to visit the entire space (i.e., cover the whole graph) by visiting each cell (i.e., graph node) once or multiple times. In our implementation, the model is able to simulate a square region (12 Km^2) divided into 4900 hexagonal cells (rather than square cells, as in the original model). This allows for a better representation of the agents' mobility in the real world, as real ant robots (as physically realized in [18]) have the ability to steer to any direction during the exploration. Robots start from specific border-cells in the terrain, and from each cell a given number of robots starts moving around (mimicking the fact that rescue teams start from specific positions, and unleash robots for discovery).

The model relies on a node-couting algorithm, where each cell is assigned a counter which gets incremented whenever any robot visits it, i.e. tracks the number of *pheromones* left by ants, to notify other ones of their transit. Whenever an agent (i.e., an ant robot) reaches a cell, it increments the counter and determines its new destination. Choosing a destination is a very important factor to efficiently cover the whole region, and to support this the trail counter is used. In particular, the ant robots adopt a greedy approach, so that when a robot is in a particular cell, it targets the neighbor with the minimum trail count. A random choice takes place if multiple cells have the same (minimum) trail count.

Although this greedy approach might not be optimal, it allows for a complete coverage of the region taking into account the simplicity of the agents, which may have a very limited and noisy sensing capability [18]. In the original model, whenever an agent is in a given cell, it accesses the information stored in the neighbor cells (i.e., trail counters) to make its decision. This is a *pull* approach, which, as we will discuss in Section 4 might entail a non-negligible performance drop when running parallel/distributed simulations.

According to the original specification of the model [17], each ant robot moves from one cell to another in a time interval of variable length (provided that the destination is reachable, i.e. no obstacle is in between the current and the destination cell). In our configuration of the model, we have made two important choices: i) there are no obstacles in the terrain; ii) time interval is drawn according to an exponential distribution with mean value 100 sec (corresponding to the typical speed of an ant robot of 50 cm per sec), which models the set of variables that can potentially impact the robot move. As for choice i), this allows us to study a *lower bound* of complete region-coverage time. In fact, inserting any obstacle will prevent agents to freely move around, not allowing certain actions, and increasing the exploration time. Choice ii), instead, allows us to study how much time is required by a group of agents to explore a small area with a very high detail, due to the intrinsic speed limitations (one meter per second [18]).

4 Parallelization

We have implemented the model[1] described in Section 3 in order to be run on top of the ROme OpTimistic Simulator (ROOT-Sim) [8, 19]. This is an open source C/MPI-based simulation platform targeted at POSIX systems, which implements a general-purpose parallel/distributed simulation environment relying on the optimistic (i.e., rollback-based) synchronization paradigm. It offers a very simple programming model relying on the classical notion of simulation-event handlers, both for processing events and for accessing a committed and globally consistent state image upon Global Virtual Time (GVT) calculation. The GVT value corresponds to the (periodically) reevaluated commitment horizon of the optimistic (speculative) simulation run. No causality violation, hence no rollback, can even occur for processed simulation events whose timestamp falls before the current GVT value. GVT updates typically trigger memory recovery procedures, e.g., of obsolete state logs.

As in typical PDES engines, in ROOT-Sim the simulation model can be partitioned into N simulation objects, each one modeling a subportion of the whole environment. Each simulation object $i \in [0, N-1]$ is associated with a private simulation state S_i, so that the global simulation state is $S = \bigcup_{i=0}^{N-1} S_i$ and $S_i \cap S_j = \emptyset \ \forall i, j \ i \neq j$. Any simulation objects is handled by a Logical Process (LP), which takes care of executing the events which are passed by the simulation framework to the application layer. Any LP implements a *process-event* callback (to be invoked by the platform), which is in charge of updating the data structures representing the current state of the simulation object. Each event passed

[1] The source code of our implementation is available at http://www.dis.uniroma1.it/∼hpdcs/ROOT-Sim/tcar.tbz.

in input to the callback is associated with a Local Virtual Time (LVT), which describes the (simulated) time advancement of the currently scheduled LP.

Events can be scheduled across simulation objects via a *schedule-event* API supported by ROOT-Sim, which maps onto application transparent message-interactions within the parallel platform. Also, log/restore of the LP state for correctly recovering causality errors within the optimistic processing scheme (caused by out of timestamp-order speculation) is fully transparent to the application programmer and autonomically optimized (e.g in terms of the selection of the frequency of log operations as a function of the rollback probability of any individual LP) [20]. Hence, the programmer can use any kind of (dynamic) data structure to implement the simulation object within the application code (which allows the LP callback function to be implemented according to the ANSI-C standard), and is provided with the illusion of a sequential discrete-event run, while the LPs are actually executed in parallel across different processes.

ROOT-Sim also supports a very peculiar service that, once a new GVT value is available, transparently rebuilds a Committed and Consistent Global Snapshot (CCGS), formed by a collection of individual LPs' states [21]. This occurs via update operations applied to local committed checkpoints of individual LPs so to eliminate mutual dependencies among the final-achieved state values. Once the CCGS is built, each LP gains control via an additional callback within the API, referred to as *OnGVT*, by also having access to the copy of its state image belonging to the CCGS. Such a service can support, e.g., termination detection schemes based on global stable predicates evaluated on a committed and consistent global snapshot. This is a relevant alternative to typical optimistic PDES engines where the run is assumed to be completed only when overstepping a given GVT value. However, the evaluation of the global predicate on the CCGS has a frequency which is bounded by the frequency of GVT calculation, thus a temporal shift can occur before the simulation application layered on top of the ROOT-Sim platform can track the holding of a global predicate while simulation time advances. This is the core point we address in our study, which is targeted at evaluating the effects of variations of the frequency of GVT calculation (which may impact the speedup of the parallel run, given that it contributes to the overhead for distributed coordination) on the estimation of the final time for area-coverage within the TCAR model.

The TCAR model depicted in Section 3 has been implemented on top of ROOT-Sim by having each LP modeling an individual hexagonal cell within the target coverag area, while robots trajectories are simulated via proper mobility events across the LPs. As mentioned, the original TCAR model adopts a *pull* approach for gathering trail counters from adjacent cells. Considering our programming model, where LPs communicate by means of (transparently handled) message passing (i.e, LPs' simulation states are disjoint), a large number of events should be exchanged to proceed in the simulation, whenever an agent must change its position. Given that the optimistic synchronization protocol [22] shows higher efficiency when the number of exchanged messages is reduced, we have rather adopted a *push* approach, relying on a notification message which is used to inform all neighbors of the newly updated trail counter whenever an agent enters a cell. Then, each LP stores in its own simulation state the neighbors' trail-counters values. In this way, agents are able to make their decisions locally. The set of events which are generated/executed by the simulation model and handled by ROOT-Sim callback operations are the following:

- **REGION_IN**: an ant robot enters a given cell. When this event is executed, the trail counter is incremented. Then, an `UPDATE_NEIGHBORS` event is scheduled at all adjacent cells, with an associated timestamp which is equal to the `REGION_IN`'s one. This means that every neighbor is immediately notified of the presence of a new ant robot in this cell at a given simulation time.
- **UPDATE_NEIGHBORS**: upon receiving this event, the LP taking care of its execution finds in the local simulation state the entry describing the trail counter for the sender of this event. Its value is updated with the one piggybacked by this event. This allows any ant robot in the cell to have (locally) a global view of the state of the neighbors.
- **REGION_OUT**: this event is associated with an ant robot leaving a cell. The logic associated with this event entails finding which is the neighbor to be reached (by consulting the locally stored information on neighbors' trail counters) and therefore scheduling a `REGION_IN` event to the destination cell. We note that, since the time spent by an agent in the cell is modeled by the difference between the timestamps associated with a `REGION_IN` event and its subsequent `REGION_OUT` event, and given that a `REGION_IN` event in any neighbor cell entails the immediate (i.e., at the same timestamp) update of all trail counters in the neighbors, upon the execution of a `REGION_OUT` event the ant robot can safely consult the locally-stored neighbors' trail counters, being sure that they contain the most up-to-date information, obtained using the aforementioned *push* approach.

ROOT-Sim natively schedules an `INIT` event at the beginning of the simulation at every LP in the system, so to allow them to initialize their private simulation states. At simulation startup, every LP determines what is its position in the square region (in terms of hexagonal coordinates) and checks whether it is a boundary cell or not. In the positive case, the LP stores this information in order to prevent ant robots to leave the terrain. The cells which (at configuration time) are selected as sources for unleashing the ant robots (e.g., the cells associated with the position of rescue teams on the terrain) detect this, and schedule at themselves an initial `REGION_IN` event, at simulation time 0.

5 Experimental Results

We have run experiments on a 32-core HP ProLiant server equipped with 64GB of RAM and running Debian 6 on top of the 2.6.32-5-amd64 Linux kernel. We have used 32 instances of the ROOT-Sim kernel (i.e., every instance takes care of about 154 cells, namely LPs) giving each instance one available CPU-core. The simulation model has been run until reaching 100% coverage of the whole region,

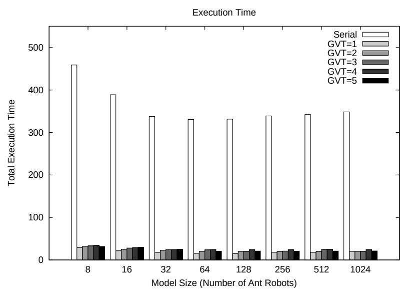

Fig. 1. Global Execution Times

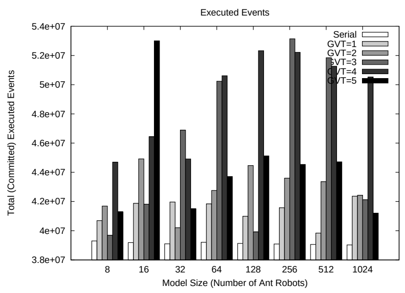

Fig. 2. Committed Events

with a visit factor of 20 (i.e., every cell must be visited at least 20 times before the simulation can complete). This has been done in order to avoid interference in the performance data due to the initial I/O (to access configuration files) and setup operations executed in the early phase of the simulation run. We have set ingress cells for robot unleash to 4, mimicking a situation where the rescue terrain is accessed by a limited number of rescue teams. Each rescue team is able to unleash a variable number of ant robots, in the range $[4, 32]$, thus having a number of agents in the simulation in between 8 and 1024.

We have compared the optimistic parallel results with sequential ones, i.e. where events are executed sequentially relying on a competitive scheduler based on Calendar Queues [23]. The GVT computation period in ROOT-Sim has been varied in between 1 and 5 sec (in wall-clock-time) to check how the confidence interval in the simulation results changes. Each plot is averaged over 20 different runs, both in the parallel and in the sequential case. The initial random seed has been varied across the different runs, in order to account for variations in the execution dynamics, but the set of seeds in the sequential and parallel executions is the same, to really compare similar execution patterns.

In Figure 1 we present the global execution times for both the sequential and the parallel execution (on top of ROOT-Sim) of the TCAR simulation model. By the results, we see that the parallel execution provides a speedup in between 15 and 22. Different settings for the GVT computation interval provide a different completion time. In particular, the higher the GVT, the higher the simulation wall-clock-time. This is not related to a loss of precision in the simulation, it is rather due to the way the simulation termination condition is evaluated. As mentioned, the GVT reduction protocol is a periodic computation. If the termination condition is verified immediately after the GVT reduction, then a non-negligible amount of wall-clock time will pass before it will be checked again (leading simulation time to advance). On the other hand, for significantly increased values of the GVT period, the opposite behavior is noted, since the delayed GVT computation allows to catch the termination condition right after it holds. This situation can be seen as well in Figure 2, where the total number of (committed) events executed by the simulation are reported. As it can be clearly seen, the parallel runs commit a higher number of events than the sequential one. Also,

Configuration		**Sequential**	**GVT=1**	**GVT=2**	**GVT=3**	**GVT=4**	**GVT=5**
16 Robots	Mean	211.86	216.31	218.27	218.69	234.99	221.81
	Std. Dev.	1,56	15.11	13.28	11.07	12.46	15.64
128 Robots	Mean	26.56	27.37	28.41	28.29	32.61	29.24
	Std. Dev.	0.16	1,08	1,37	3,25	1.83	1.01

Table 1. Mean Completion Simulation Time (in Simulated Hours)

the peak values for the parallel runs are not alway noted for larger GVT period, just for the reasons explained above.

In order to assess the results' reliability, in Table 1 we present the LVT values at which two intermediate configurations of the simulation were stopped. These are the ones with 16 and 128 robots. The reported information is useful to model writers, as it is part of the outcome of the simulation itself. In particular, this tells how much time (in simulated hours) the ant robots would need to cover the entire region. From the data (presented in form of mean time and standard deviation), it can be clearly seen that the sequential simulations offer the most stable result. Interestingly, the parallel executions show a completion simulated time which is always higher than the sequential one. This is related, again, to the way the termination condition is checked upon GVT reduction. This aspect is very relevant, showing that the outcome of parallel (optimistic) simulation does not give the most precise result, rather it places a (correct) upper bound on it. Of course, it is up to the simulation model writer to decide how much this divergence from the precise simulation results can affect her simulation, and how much benefit she can gain from the increased performance.

References

1. Takahashi, T., Tadokoro, S., Ohta, M., Ito, N.: Agent based approach in disaster rescue simulation - from test-bed of multiagent system to practical application. In: RoboCup 2001: Robot Soccer World Cup V, Springer-Verlag (2002) 102–111
2. Macy, M.W., Willer, R.: From factors to actors: Computational sociology and agent-based modeling. Annual Review of Sociology **28**(1) (2002) 143–166
3. Junli, L.: Agent-based logistics simulation system design and implementation. In: Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology. ICCSIT, IEEE Computer Society (2009) 602–606
4. Macal, C., North, M.: Tutorial on agent-based modeling and simulation part 2: How to model with agents. In: Proceedings of the 2006 Winter Simulation Conference. WSC, Society for Computer Simulation (2006) 73–83
5. Page, S.E.: Agent-based models. In Durlauf, S.N., Blume, L.E., eds.: The New Palgrave Dictionary of Economics. Palgrave Macmillan (2008)
6. Karmakharm, T., Richmond, P.: Large scale pedestrian multi-simulation for a decision support tool. In: TPCG. (2012) 41–44
7. Taufer, M., Padron, O., Saponaro, P., Patel, S.: Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs. In: IPDPS. (2010) 1–9
8. The High Performance and Dependable Computing Systems Research Group (HPDCS), Sapienza, University of Rome: ROOT-Sim: The ROme OpTimistic Simulator - v 1.0. `http://www.dis.uniroma1.it/~hpdcs/ROOT-Sim/` (October 2012)

9. Pellegrini, A., Vitali, R., Quaglia, F.: An evolutionary algorithm to optimize log/restore operations within optimistic simulation platforms. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques. SIMUTools, SIGSIM (2011)
10. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: A multiagent simulation environment. Simulation **81**(7) (July 2005) 517–527
11. Cordasco, G., Chiara, R.D., Mancuso, A., Mazzeo, D., Scarano, V., Spagnuolo, C.: A framework for distributing agent-based simulations. In: Euro-Par Workshops (1). (2011) 460–470
12. Wittek, P., Rubio-Campillo, X.: Scalable agent-based modelling with cloud hpc resources for social simulations. In: Proceedings of the 4th International Conference on Cloud Computing Technology and Science. CloudCom, IEEE Computer Society (2012) 355–362
13. Hybinette, M., Kraemer, E., Xiong, Y., Matthews, G., Ahmed, J.: Sassy: A design for a scalable agent-based simulation system using a distributed discrete event infrastructure. In: Proceedings of the 2006 Winter Simulation Conference. WSC, Society for Computer Simulation (2006) 926–933
14. Richmond, P., Walker, D.C., Coakley, S., Romano, D.M.: High performance cellular level agent-based simulation with FLAME for the GPU. Briefings in Bioinformatics **11**(3) (2010) 334–347
15. Marurngsith, W., Mongkolsin, Y.: Creating gpu-enabled agent-based simulations using a pdes tool. In Omatu, S., Neves, J., Rodriguez, J.M.C., Paz Santana, J.F., Gonzalez, S.R., eds.: Distributed Computing and Artificial Intelligence. Volume 217 of Advances in Intelligent Systems and Computing. Springer International Publishing (2013) 227–234
16. Caggianese, G., Erra, U.: Exploiting gpus for multi-agent path planning on grid maps. In: HPCS. (2012) 482–488
17. Koenig, S., Liu, Y.: Terrain coverage with ant robots: a simulation study. In: Proceedings of the fifth international conference on Autonomous agents. AGENTS, ACM (2001) 600–607
18. Svennebring, J., Koenig, S.: Building terrain-covering ant robots: A feasibility study. Autonomous Robots **16**(3) (May 2004) 313–332
19. Pellegrini, A., Vitali, R., Quaglia, F.: The ROme OpTimistic Simulator: Core internals and programming model. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques. SIMUTools, ICST (2011)
20. Vitali, R., Pellegrini, A., Quaglia, F.: Autonomic log/restore for advanced optimistic simulation systems. In: Proceedings of the Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. MASCOTS, IEEE Computer Society (2010) 319–327
21. Cucuzzo, D., D'Alessio, S., Quaglia, F., Romano, P.: A lightweight heuristic-based mechanism for collecting committed consistent global states in optimistic simulation. In: DS-RT. (2007) 227–234
22. Jefferson, D.R.: Virtual Time. ACM Transactions on Programming Languages and System **7**(3) (July 1985) 404–425
23. Brown, R.: Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem. Communications of the ACM **31** (October 1988) 1220–1227