PUC

ISSN 0103-9741

Monografias em Ciência da Computação

nº 17/12

Virtual Environment Simulations (VES) with MABS: an approach to tame tardiness in Java-based simulation systems

Pier-Giovanni Taranti

Carlos J. P. de Lucena

Ricardo Choren

Pierre Bommel

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 17/12 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena December, 2012

Virtual Environment Simulations (VES) with MABS: an approach to tame tardiness in Java-based simulation systems

Pier-Giovanni Taranti Carlos J. P. de Lucena
Ricardo Choren[1]
Pierre Bommel[2]

[1]SE/8 - IME, Brazil
[2] CIRAD - UPR GREEN, France

taranti@inf.puc-rio.br , lucena@inf.puc-rio.br , choren@ime.eb.br , bommel@cirad.fr

Resumo. Simulações de Ambiente Virtuais (VES) são um tipo especial de simulação, normalmente utilizadas para implementar jogos e jogos sérios com representação espacial. Um exemplo de jogo sério é uma simulação utilizada para apoiar Jogos de Guerra. Simulações Baseadas em Sistemas Multiagentes (MABS) são adequadas para implementar estas simulações devido a sua habilidade em tratar complexidade e modelagem de atores individuais. Contudo, o avanço do tempo da simulação nestas simulações é um desafio, devido a existência de *tardiness* no sistema. Esta situação é pior em MABS que utilizem Java, devido a particularidades desta tecnologia. Este trabalho apresenta uma abordagem para controlar o *tardiness* e auxiliar o desenvolvimento de VES utilizando o paradigma de orientação a agentes.

Palavras-chave: Simulação Baseada em Multiagentes, atrasos em tempo de simulação

Abstract. Virtual Environment Simulations (VES) are a special type of simulation, often used to implement games and serious games with virtual space representation. An example of serious games is the simulation used to support War Games. MABS is suitable to implement these simulation due their ability to handle with complexity and individual actors modeling. However, the simulation time advance in this simulation is challenging, due to the existence of tardiness in the system. This situation is worst in Java-based MABS, because of Java technology particularities. This paper presents an approach to tame this tardiness and help the development of these cited VES using agent oriented paradigm.

Keywords: Multi-Agent Based Simulation, Tardiness

[0]Accepted in MAS&S'2012

In charge of publications :

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Sumário

1 Introduction

A particular type of simulation is the Virtual Environment Simulation (VES) [1], which intend to create a computer-generated virtual environment, in which users can be embedded. These simulations differ from others in one important aspect: the virtual scenario must represent a plausible scenario, that is, the general picture of the simulation need to be perceived as a possible situation by the users. Another aspect of VES is that these simulations are often Real-Time Simulations or Scaled-Time Simulations, which means that the simulation time advances in a linear relation with the physical time.

War games simulators are an example of VES usage. In these simulations, military units such as helicopters, ships and fighters are represented in a virtual space. Each unit has cinematic behavior and can interact with each other. The context of each unit is composed by social and spatial factors, besides the existence of rules and, possibly, norms to be obeyed. An overview of War Games is presented in [2]. The War Games itself will not be discussed in this paper, but the technology used to implement the simulation that supports these games.

Multi-Agent Based Simulation (MABS) are a suitable approach to model and implement these war games, since representing units as agents allows to: implements classes of units in agents and reuse them in other scenario; create a model adherent to the real world, in which each agent has the autonomy to act in the virtual environment; implement artificial intelligence algorithms in the agents, in order to provide more realistic behavior to the simulation agents.

In addition, the real system is complex, since actions of individual units at micro level could affect the overall state of the system at macro level. Multi-Agent paradigm is suitable to model and implement complex systems, due to their capability to model and represent this micro level [3, 4].

Although the advantages in using agent paradigm in the war games simulators, there is an issue that makes this use difficult: when simulation agents control their advance in the simulation time, a situation similar to parallel simulation emerges. The time synchronization takes a major role in this case: if agents are not capable to execute the simulated actions in the correct time, errors will appear in the simulation. This error is called tardiness. As an example, an agent that represents a fighter could not detect an enemy when users expect that it does.

The time synchronization is challenging when working with Java-based MABS, since details related to the Java technology will insert unpredicted delays in the simulation. Some examples are: the VM implementation, the Garbage Collector or the mechanism used to verify the correct instant to execute a scheduled task.

This paper presents a work that aims to support the development of simulators and simulations engines for MABS that use scaled time approach and developed with Java technology, and, in this way, support the development of VES with MABS with individual agents representing simulated entities. The approach is composed by an architecture to control the advance of the simulation time ensuring that the tardiness is tamed at levels established by the simulation designer. To accomplish this goal, an agent-based architecture is provided to replace the simulation clock.

This is an on-going work, and its previous results and outputs were published in [5, 6]. The specific contribution of this text is the inclusion of support for geo-referenced simulations, beside the improvements in the algorithms used to control the simulation time advance.

This paper is organized as follows: section II presents some challenges to implement VES using MABS; section III presents the proposed approach; a study of usage is presented in section IV; section V discusses Related Work; and the Conclusion is presented in section VI.

2 Challenges to implement VES using MABS

Modeling the simulated system as actors with actions is a natural option for designers of war games: in the real world military units are autonomous, they act following some plan, and each one has specific state. In addition, each military unit is situated [7], that is, each action should consider the spatial and organizational situation, respecting norms. As already stated, the use of agents in this domain provides a set of advantages such as software reuse, social ability, ease to implement IA algorithms, and others.

To gain these advantages, actors should be implemented as individual agents, and these must have the control over their state in the simulation. This means that agents have the control over the simulated events' execution and over the advance of the simulation time. As a whole, each agent acts as an individual model in the simulation, and the simulation itself takes the form of a parallel simulation. Each simulated event is a tuple (e, t_s), where e is the simulated event and t_s is the simulated time when the event must be executed. The simulated time, or logical time, is the abstraction that represents the physical time in the simulation.

However, to achieve t_s is challenging in Java-based systems. There are three main issues that influence the final delays: First, the Java Threads have a non-preemptive scheduling mechanism, and the Java VM itself does not specify how Java-Threads are mapped to the Operational System (OS) Threads. It is not possible to foresee the consequences of OS threads suspension on the Java based system in execution. Second, the garbage collector (GC) affects the Java-based system in execution every time it performs a memory collection. Finally, the time schedule mechanism in Java uses verifications such as $if(now >= t)$. Often the value of now is greater than t, and we have a delay of $((-1).(t - now))$. The delay in executing a scheduled action is called *tardiness*. The tardiness level is also affected by resources availability in the host, such as memory and CPU time.

VES often have some flexibility, admitting small delays in the instant when actions are executed, if the overall simulation consistence is not affected. As an example, if the user is observing a gunfire in the simulation, he expects that the flash will happen before the sound. Otherwise, the realism of the simulation will be lost. However, there is not any significant influence if the sound occurs 100ms or 200ms after the flash.

In these VES, and specifically in war games, the delays have consequence on the spatial representation, since a delay in updating a unit position will insert spatial error in the virtual space. Although some flexibility should be accepted, there is a limit, and this limit is the half of the least detection range in the simulated system, as a general rule.

Thus, to implement VES in MABS we do not need to eliminate the tardiness in the system, but to tame this tardiness to established limits that ensure the simulation consistence. In the next section an approach to this issue is presented.

3 Taming the Tardiness in VES

An important consideration about tardiness is its relation with the real time that agents have to compute. This time have relation with the period $t_{r1} - t_{r0}$, where t_{r1} is the instant in real time when the agent must execute $e1$ to achieve t_{s1}, in the scheduled action (e_1, t_{s1}), and t_{r0} is the instant when the simulated action entered in the scheduled events queue. The cited relation was shown in [8, 9].

In scaled-time simulations, t_s is function of t_r, that is $t_s = t_r.K$, where K is a constant Linear Coefficient. We propose that, in order to provide a sufficient period $t_{r0} - t_{r1}$ for agents executing actions $(e_1, e_2, .., e_n)$, we could replace the constant K by a function $f(x, y, z, ...)$.

The set $(x, y, z, ..)$ is composed by meaning variables which the values should be informed periodically at execution time. Some examples of these variables are: greater tardiness in a period, number of measures greater than the established limit in a period and bias of the tardiness.

3.1 The approach implementation

The proposed function is implemented in an algorithm used to control a centralized simulation clock component. This clock is responsible for: collecting all values for the meaning variables used in f; calculating the current $f(x, y, z..)$; applying this value to update the simulation time value; and providing the current simulation time to agents in execution time.

The approach considers that the simulation agents implement the interface and the algorithm to individually inform their tardiness in execution time and is an evolution of the work presented in [5, 6]. The implementation of the proposed concepts was made on the MASP platform, presented in the cited work.

The very difference between this paper and the previous two works is the ability in treating MABS with agents that move in the virtual space using different and variable speeds. This spatial representation of the virtual environment may be in the main memory or in a spatial database compliant with OpenGIS standards [10].

For accommodating this new abilities, the MASP architecture was modified to solve some concurrency issues, and a spatial representation for simulations was provided. The MASP allows to represent the spatial virtual environment in memory or in a loose-coupled approach between the MABS and the GIS database. These techniques are discussed in [11, 12].

The algorithm that implements the f function was evolved to work not only with simulations that uses the stepped-time approach to advance the simulation time, but also the next-event approach. These two approaches are the main ways to advance the simulation time, and are described in [13–15]. The current architecture is presented in the Component Diagram (fig. 1) and some components are individually treated due to concurrency issues.

The VirtualSpaceHandler component is the connection point with the GIS. The geographical positioning may be treated in memory or persisted in a database. The current implementation uses PostgreSQL with POSTGIS. The architecture main piece is the algorithm that is used to dynamically update the $f(x, y, z..)$ value that determines the Simulation Time advance over the Real Time (i.e the physical time), and this algorithm is in

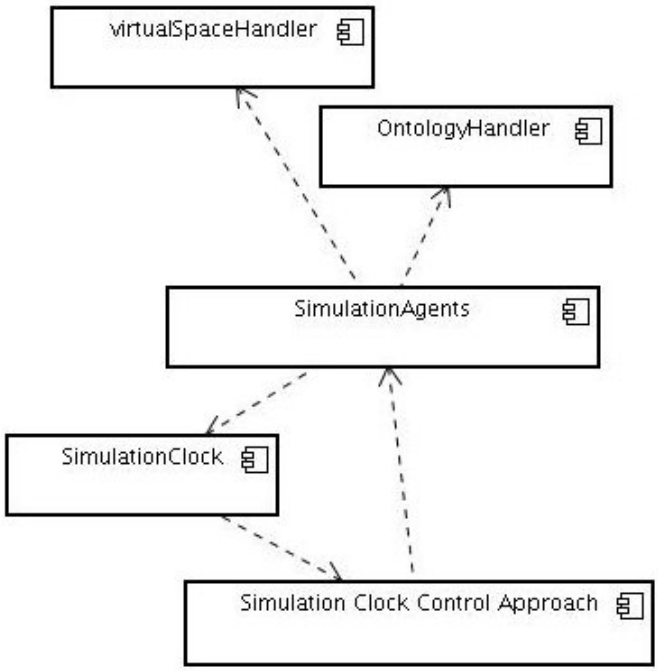

Figura 1: Proposed Architecture

the SimulationClockControlApproach component.

This algorithm was evolved to support MABS in which agents could change their behaviors period, being indifferent to these. This is important to allows simulation agents to change their speed in the virtual environment during the simulation. Beside these characteristics, the algorithm needs to be able to maintain the $f(x, y, z..)$ value at a level that ensures a tardiness below the maximum level admitted and, moreover, be able to speed-up the simulation, when it is possible.

The simplified class diagram of the current implementation for simulation-time control approach is shown in figure 2. This diagram presents the main classes that compose the simulation clock and its control mechanism.

The current algorithms periodically verifies the major tardiness, the tardiness bias, or tendency, and the number of tardiness occurrences greater than a specific level in a period. This information is used as input to define a new value to f, which will slow down, speed up, or maintain steady the simulation time advance. This code is implemented in the *SimulationClockControl* class.

As stated in the previous works, the approach is being continually improved. This occurs mainly because of the increasing complexity of the experiments used to test this approach. The goal is to obtain a solution to support VES implementation using MABS, which implies in supporting geo-referenced simulations where agents with very different speeds may control and alter their update period. The next section shows the current results.

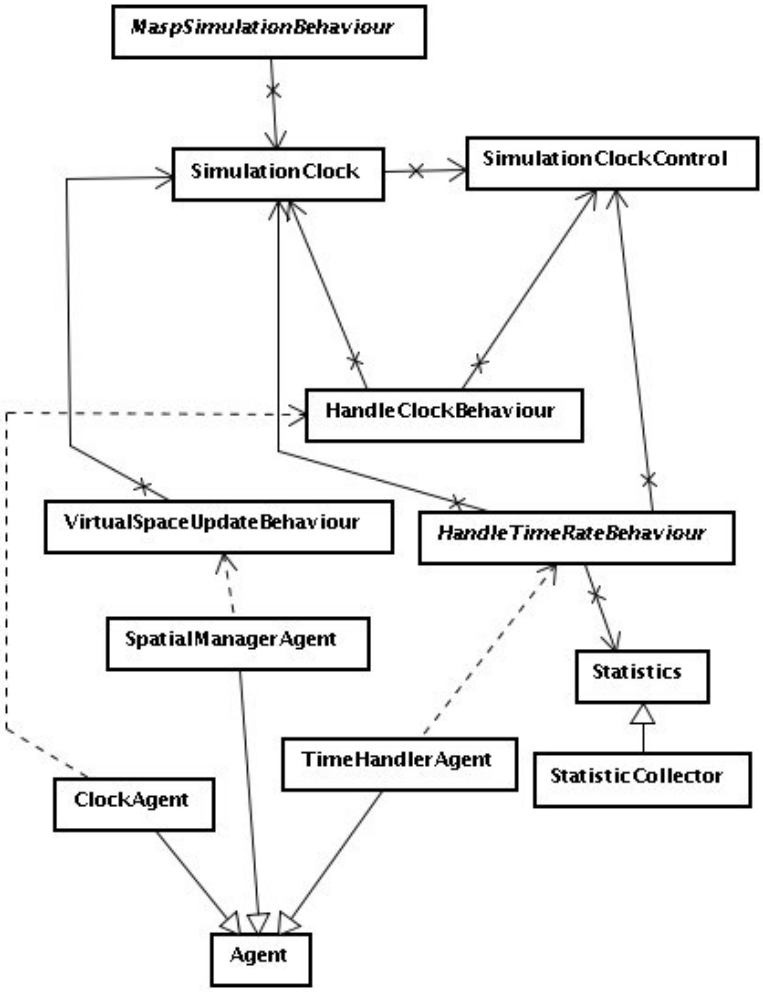

Figura 2: Class Diagram

4 A Study of Usage

In order to test the proposed approach, a scenario with 500 agents was instantiated representing actors, with different speed limits and acceleration. The values, all in the International System of Units (SI) are presented in the table 1.

For each type, 100 agents were instantiated with a behavior for modifying the speed and the course in a random way, but respecting the initial parameters. The maximum spatial error admissible in the simulation was 500 meters. The agents are programmed to adjust their control loop period in order to meet this 500 meters constrain. The initial rate for the simulation was 5:1, which means that in $t_s = 0$, $f = 5$. The maximum tardiness admissible was 10% of the expected period for a control loop execution and the tardiness level to trigger the control approach was 2% of the control-loop period. The algorithm was adjusted to perform 4 tardiness verifications and one for the simulation speed-up during the period of the major control loop informed by the simulation agents.

The experiment used two samples: the control sample ran without interference, remaining with $t_s = 5.t_r$ during all experiment, and the main sample used the settings described above and $t_s = f(tardiness, bias, number\ of\ tardiness).t_r$. The experiments are recorded as comma-separated files (.csv), and later those files were analyzed using R software [16].

Both simulation for collecting the samples were executed in the same environment, a personal computer with a i7 CPU and 8 GB of RAM. The Operational System used was the OpenSuse 12.1 (a linux flavor) in command line mode, that is, without graphical interface. Services, such as network and printer server were deactivated in order to avoid interferences in the collected data.

The results are presented in graphs, and these are discussed one by one, as follows:

4.1 Simulation time advancing

The proposed approach relies on the concept that if tardiness could be detected at execution time, then it is possible to change the f value in order to give more execution time for the simulation system and, as a consequence, it is also possible to tame the tardiness and the spatial error in MABS.

A consequence of this approach is the reduction of the simulation time performed. At constant rate, using $t_s = 5.t_r$ and 25 minutes of simulation a total of 125 minutes of simulation time is obtained. On the other hand, the main sample was able to perform only 2.3 minutes of simulation time in the same period (figure 3).

This difference occurred because of the automatic reduction of the f value, which

Tabela 1: Simulation Scenario

Agent Type	Min Speed	Max Speed	Max Acel.
Fighter	150	300	2.0
Helicopter	5	60	0.8
Frigate	4	20	0.3
Submarine	1	10	0.2
Submarine (N)	4	20	0.2

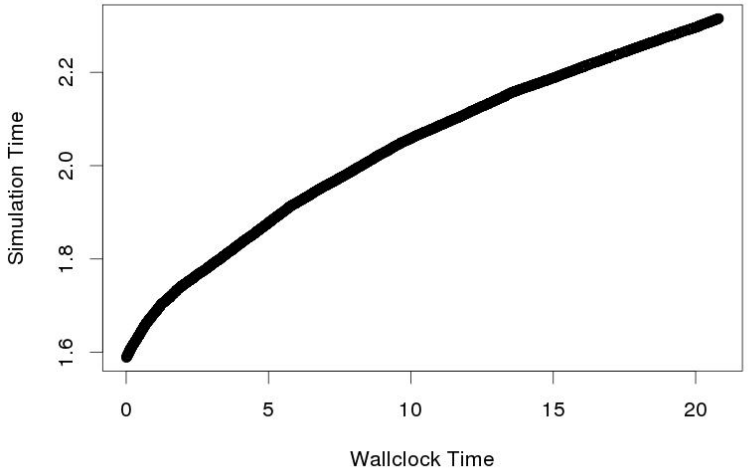

Figura 3: Main Sample – Simulation Time & Real Time

started as $f = 5$, done by the architecture implementation in execution time. The graph shown in figure 4 depicts that the f value, that had started with value 5, decreased fast and reached some stability near 0.3

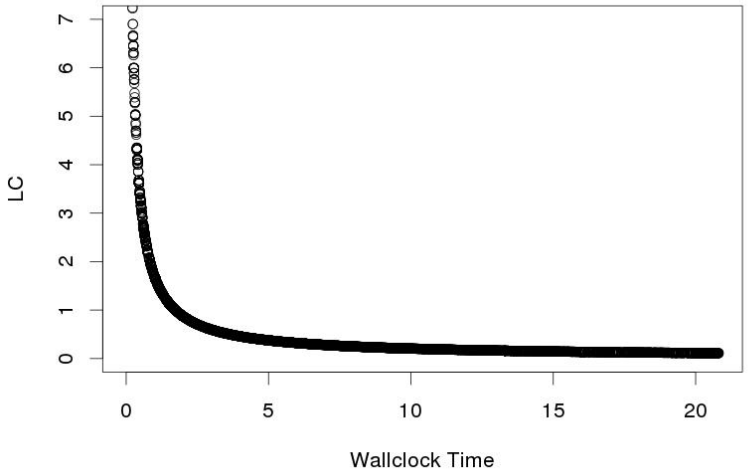

Figura 4: Main Sample – f value x Real Time

These two graphs show the cost of the approach. The next two subsections present the advantages of it usage.

4.2 Tardiness in simulation time

One of the goals of the approach is to tame the tardiness in relation to the simulation time. Initially the graphs of the figures 5 and 6 present the control sample information, that uses the fixed relation $t_s = 5.t_r$. In graph 5, the Tardiness histogram, it is observed that the tardiness varies from 0 to 6, approximately, in a distribution similar to an exponential.

The later graph shows the tardiness occurrence during the simulation execution. This graph shows that the tardiness is distributed along the time and the boxplot also confirms that the tardiness does not have a normal distribution, complying with the histogram. In this representation is possible to see that almost all tardiness measures are below 2.3, and bigger values are statistically classified as outlines.

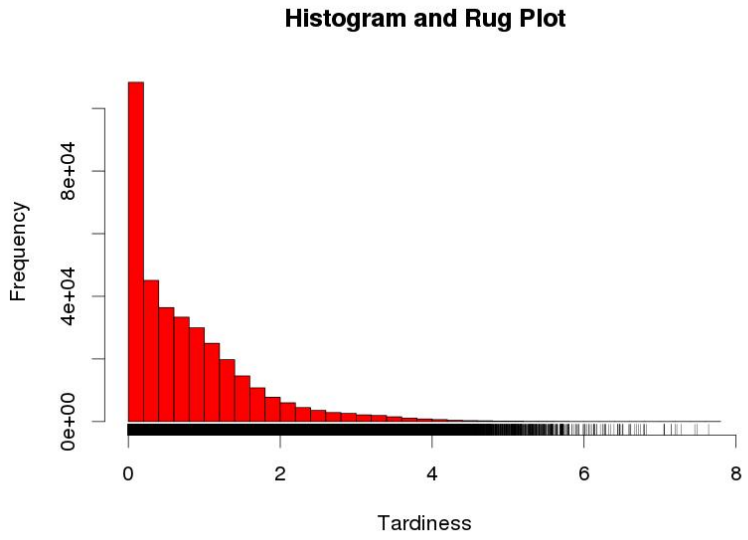

Figura 5: Control Sample – Tardiness Histogram

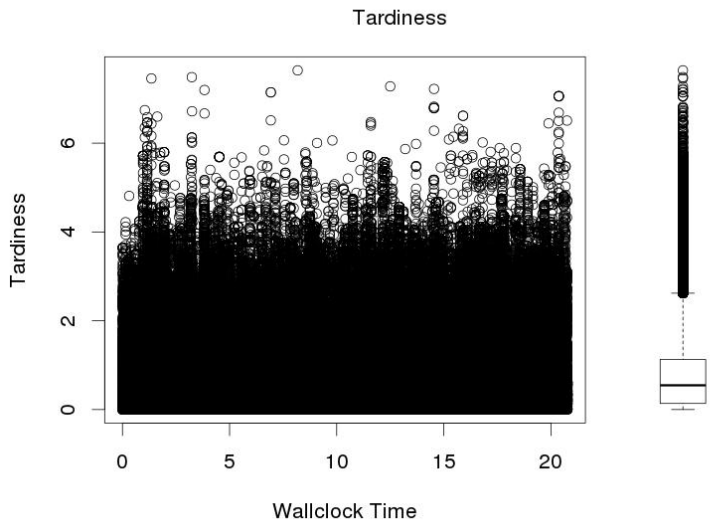

Figura 6: Control Sample – Tardiness & Simulation Time / Boxplot

The next two graphs (figures 7 and 8) represent the situation of the main sample, with the f value automatic control mechanism activated. In the first one, it is possible to see a similar behavior, however the values are very different: almost all sample is below 0.01 of tardiness, which means less than 1% of delay. Another observation is that the tardiness decreased along the time of execution (figure 8), in a contrast with the control sample behavior (figure 6). This happens because of the f value variation, saw in figure 4. After the system achieves a stable situation the tardiness tends to be distributed along the time

axis. In the boxplot, it is possible to observe that the greater values for outlines are below 0.06, which is according the limit established as a parameter (0.1).

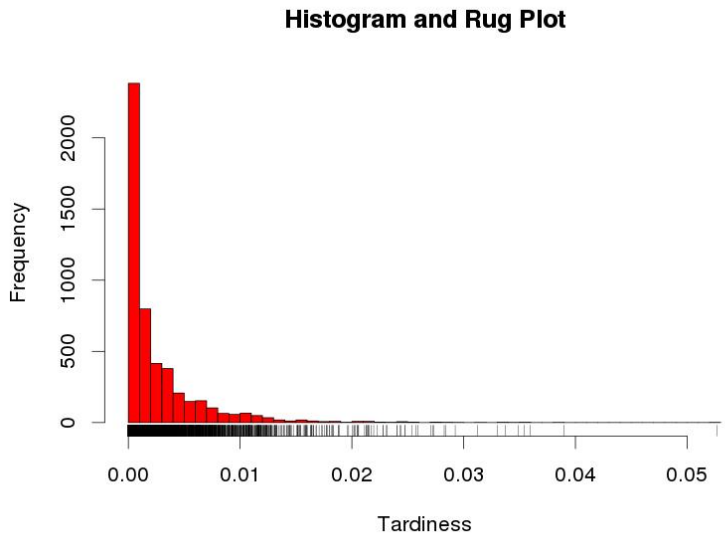

Figura 7: Main Sample – Tardiness Histogram

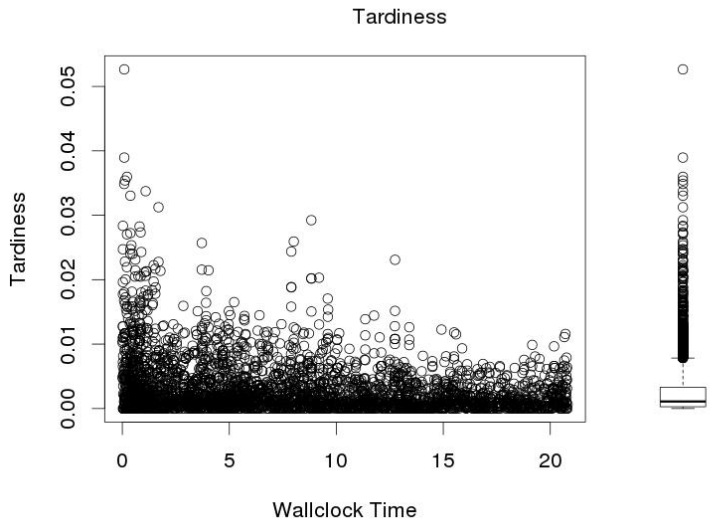

Figura 8: Main Sample – Tardiness & Simulation Time / Boxplot

4.3 Spatial Error

The spatial error is critical when working with geo-referenced actors that move around a virtual space, since a wrong positioning may prevent that agents detect each other when this is expected to happen. As a consequence, agents will not consider the presence of others when performing their decision making.

When agents have the full control of their control loop, it is expected that different levels of tardiness appear. Since the act of update the position in the virtual space (or in the GIS) is a periodical behavior, a wrong positioning occurs as a consequence of this

behavior tardiness. In other words, if an agent is expected to advance 2 meters in 100ms, and it only has success in performing this update in 120ms, the final position may be wrong by 0.4 meters. These spatial errors can lead agents to fail several times in detecting other agents inside the virtual space, affecting the overall behavior of the simulation.

The figures 9 and 10 show the spatial error recorded from the Control Sample. The histogram shows occurrences of spatial errors greater than 4,000 meters. The second shows that the error is regularly distributed during the execution time, and that the majority of the errors are below 1,500 meters. It is highlighted that approximately 25% of the measures are between 500 and 1,500 meters, a value greater than the established as acceptable in the simulation initial parameters, which compromises the results of this simulation.

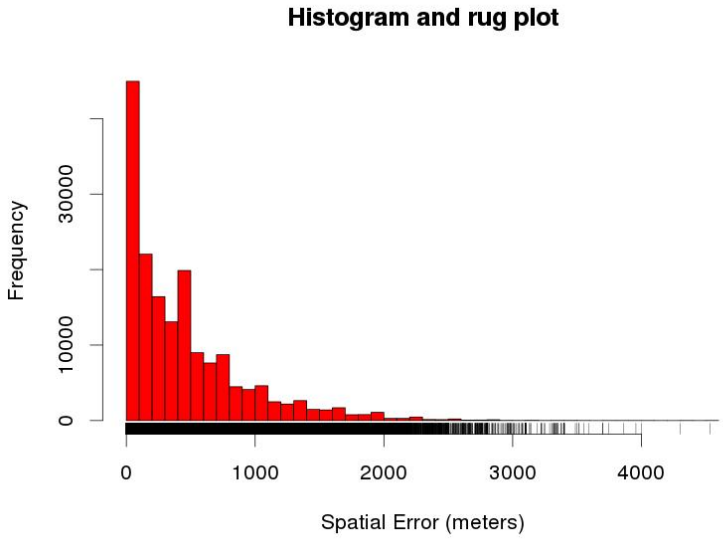

Figura 9: Control Sample – Spatial Error Histogram

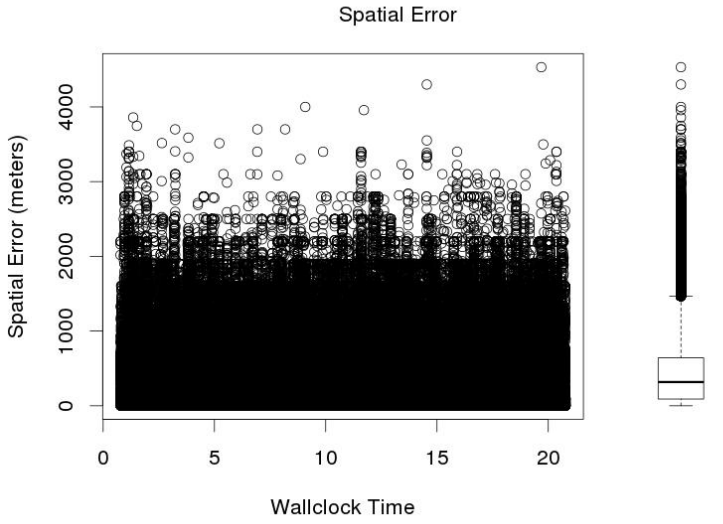

Figura 10: Control Sample – Spatial Error & Simulation Time / Boxplot

The algorithm that controls the updating of the f value was specifically modified to reduce also the spatial error. For that, agents confirm the current simulation time and

will update its position on GIS using the current information, not only the expected. The dilatation of the real time available for computing new states helps to reduce the spatial error for all simulation agents.

In figure 11, from the main sample, it is shown that the spatial error was tamed to limits between 0 and 250 meters, below the level established as critical for this simulation. An interesting point is the concentration of spatial errors in some values, such as 20, 50 and 100 meters. This information indicates that some modifications in the algorithm is still desirable, in order to search a distribution more compliant with the normal.

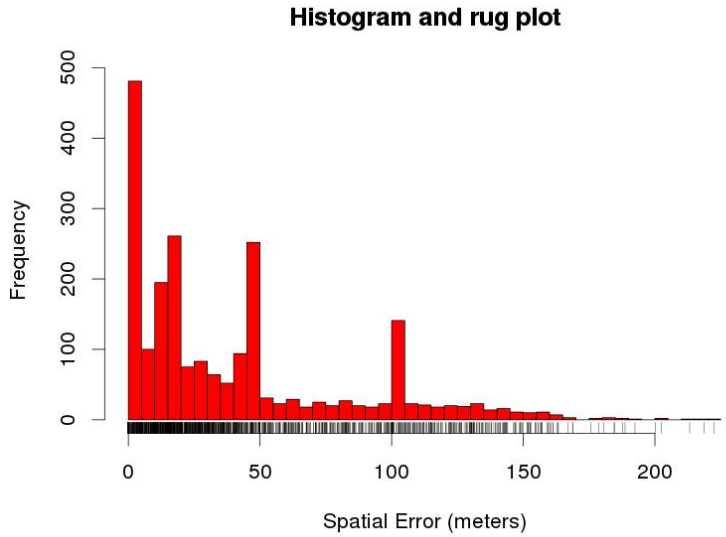

Figura 11: Main Sample – Spatial Error Histogram

The figure 12 is another spatial error view obtained from the main sample. The graph shows a regular boxplot, with outlines lesser then 200 meters, in absolute value. The quartile distribution of the measures was considered acceptable in research. The spatial error was also regularly distributed along the real time axis.

Both graphs indicate that the spatial error will occur, however they will be bounded by the established limits. This situation observed *in– silico* represents a possible situation in the real world.

5 Related Work

Our bibliographic research did not have success in locating other work that aims to solve similar issues.

A number of works were used as reference and are related to some part of this work, and some of these papers are referenced in this work. Some papers were already published by the authors, presenting other stages of the work.

Specifically about the spatial error in simulations, there are a number of approaches to reduce the error by estimation, but these approaches address the simulated word representation for each simulator in parallel and distributed simulation. Some of these approaches can be seen in [1]. However the problem addressed by this research is the spatial error caused inside the simulator due to the tardiness in performing the updates. The work also discusses the coupling of GIS and MABS.

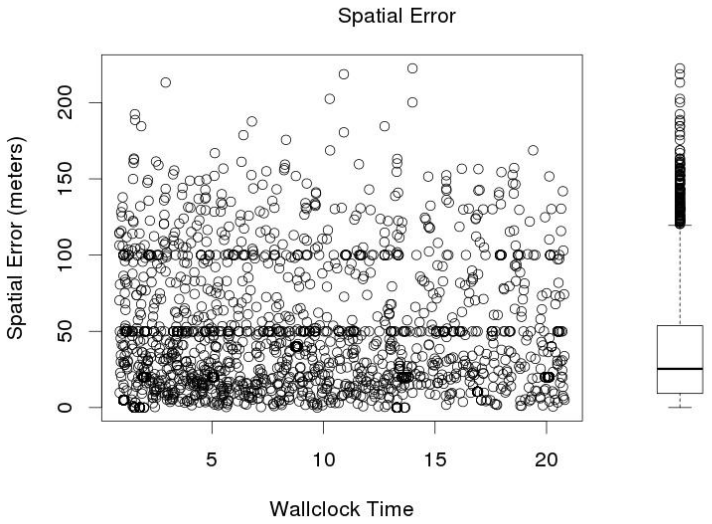

Figura 12: Main Sample – Spatial Error & Simulation Time / Boxplot

The work JiST [17] discuss simulation time issues when executing simulations implemented in Java, and provides an approach to eliminate this error by substituting the system time by a logical time advancing, without relation with the real time advance. This approach is not suitable to scaled time simulations and does not support MABS.

The work [18, 19] presents an approach to handle time issues in MAS, that is useful for MABS, too. The work differs conceptually from this, since it does not enforce a relationship between logical and real time and, in this sense, does not support the execution of VES. These works have inserted the concept of time model, in which agents' actions and deliberations have a proper logical time associated.

About the dilatation of the K value in $t_s = K.t_r$, the only reference verified was [20]. The JACK documentation informs that it is possible to handle the K, however this documentation does not inform if it is possible to act on the K value during the simulation. The tardiness and the spatial error are not discussed in the documentation.

6 Conclusion and Future Work

This work presented an approach to implement MABS in which the simulation time is advanced using an individual time-stepped mechanism. The approach is addressed to tame tardiness and spatial error to established limits and, with this, to ensure that the simulation represents a possible scenario of the real system.

The approach considers agent–centered turns, in which each agent can control its advance in relation of the simulation time and is based in a central simulation clock. Agents feedback of tardiness and step-time are used to modify the linear coefficient between real and simulation in run time. This modification is performed by an agent-based component that uses an algorithm developed for this specific task.

A conceptual agent architecture is presented, and also its implementation. The presented examples indicated that the approach has success in taming the tardiness and the spatial error. However it is desirable to refine the algorithm that controls the f value, in order to obtain a spatial error distribution more compliant with the normal distribu-

tion. This result will allow the use of some statistic tools to treat the error in real time, additionally to the current approach.

The current status of the research indicates that it is possible to develop MABS to support war games and other applications, ensuring that the error will not overpass some established limits.

Referências

[1] FUJIMOTO, R. M.. **Parallel and Distributed Simulation Systems**. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons Inc., 2000.

[2] PERLA, P. P.. **The art of wargaming: a guide for professionals and hobbyists**. Naval Institute Press, 1990.

[3] MOFFAT, J.. **Complexity theory and network centric warfare**. Information age transformation series. DoD Command and Control Research Program, 2003.

[4] PIDD, M.. **Systems Modelling: Theory and Practice**. Wiley and Sons, LTD, Chichester, England, 2004.

[5] TARANTI, P. G.; LUCENA, C. J. P. ; CHOREN, R.. **Mabs com turnos centrados em agentes e implementadas em java: controlando atrasos em relacao ao tempo de simulacao**. In: ANAIS DO II WORKSHOP SOBRE SISTEMAS DE SOFTWARE AUTONOMOS (AUTOSOFT 2011) - CBSOFT 2011, p. 40–49, Sao Paulo, 2011. SBS.

[6] TARANTI, P.-G.; DE LUCENA, C. J. P. ; CHOREN, R.. **An architecture to tame simulation time tardiness in ads**. In: PROCEEDINGS OF THE 2011 WORKSHOP ON AGENT-DIRECTED SIMULATION AT THE 2011 SPRING SIMULATION MULTICONFERENCE, ADS '11, p. 29–36, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[7] DEY, A.. **Understanding and using context**. Personal and Ubiquitous Computing, 5(1):4–7, 2001.

[8] TARANTI, P.-G.; LUCENA, C. J. P. D. ; CHOREN, R.. **A quantitative study about tardiness in java-based multi-agent systems**. In: AGENT SYSTEMS, THEIR ENVIRONMENT AND APPLICATIONS (WESAAC), 2011 WORKSHOP AND SCHOOL OF, p. 37 –44, Curitiba, Parana, Brazil, april 2011. Print ISBN: 978-1-4673-0735-2.

[9] TARANTI, P. G.; CHOREN, R. ; LUCENA, C. J. P.. **A quantitative study about the hidden risk of using time-scheduler mechanisms to control execution flow in jade-based mas**. In: PROCEEDINGS OF I WORKSHOP ON AUTONOMOUS SOFTWARE SYSTEMS (AUTOSOFT) AT FIRST BRAZILIAN CONFERENCE ON SOFTWARE: THEORY AND PRACTICE (CBSOFT), p. 61–70. SBC, outubro 2010.

[10] OPEN GEOSPATIAL CONSORTIUM. **OpenGIS standards**. http://www.opengeospatial.org/standards/, 2011.

[11] GONÇALVES, A.; RODRIGUES, A. ; CORREIA, L.. **Multi-Agent Simulation within Geographic Information Systems**. In: PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON AGENT-BASED SIMULATION (ABS-2004), p. 107–112., Lisboa, Portugal., 2004.

[12] PARKER, D.. **GIS, Spatial Analysis and Modelling**, chapter Integration of Geographic Information Systems and Agent-Based Models of Land Use: Prospects and Challenges, p. 403–422. ESRI Press, Redlands, CA, 2005.

[13] SHANNON, R. E.. **Systems Simulation: the art and science**. Prentice Hall, 1975.

[14] LAW, A.; KELTON, D.. **Simulation Modeling and Analysis 2nd**. McGraw-Hill, New York, NY, 1991.

[15] ZEIGLER, B. P.; PRAEHOFER, H. ; KIM, T. G.. **Theory of modeling and design - integrating discrete event and continous complex dynamic systems (2. ed.)**. Academic Press, 1999.

[16] R DEVELOPMENT CORE TEAM. **R: A Language and Environment for Statistical Computing**. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[17] BARR, R.; HAAS, Z. J. ; VAN RENESSE, R.. **Jist: an efficient approach to simulation using virtual machines**. Software: Practice and Experience, 35(6):539–576, 2005.

[18] HELLEBOOGH, A.; HOLVOET, T.; WEYNS, D. ; BERBERS, Y.. **Towards time management adaptability in multi-agent systems**. In: Kudenko, D.; Kazakov, D. ; Alonso, E., editors, ADAPTIVE AGENTS AND MULTI-AGENT SYSTEMS II, volumen 3394 de **Lecture Notes in Computer Science**, p. 88–105. Springer Berlin / Heidelberg, 2005.

[19] HELLEBOOGH, A.; VIZZARI, G.; UHRMACHER, A. ; MICHEL, F.. **Modeling dynamic environments in multi-agent simulation**. Autonomous Agents and Multi-Agent Systems, 14(1):87–116, February 2007.

[20] JACK. **Jack Intelligent Agents, software agent system**. http://aosgrp.com/products/jack/index.html, 2011.