A Self-Organizing Virtual Environment for Agent-Based Simulations

Mohammad Al-Zinati, Rym Wenkstern
University of Texas at Dallas
800 West Campbell Road
Richardson, Texas, USA
{mohammad.al-zinati, rymw}@utdallas.edu

ABSTRACT

In this paper we present a self-organizing model for *open* virtual environments in multi-agent based simulation systems. Open environments are inaccessible, non-deterministic, dynamic and continuous. A virtual environment is partitioned into areas called *cells* and is supported by an underlying autonomic software system consisting of specialized agents called *controllers* and *coordinators*. Controllers manage specific cells while coordinators monitor and guide controllers in the execution of their tasks. Controllers and coordinators continuously interact with one another and re-organize themselves and the environment structure to ensure that the simulation functional and performance requirements are met. During the execution of the simulation, virtual agents are unaware of the partitioned structure of the environment and the underlying self-organization activities. The experimental results show a significant improvement in the scalability and performance of the simulation system. Moreover, the emergence of undesired behavior is controlled.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Multiagent systems; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*self-organizing*; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Algorithms, Design, Experimentation

Keywords

multiagent simulation; decentralized virtual environment

1. INTRODUCTION

Modern Multi-Agent-Based social Simulations (MABS) has been of interest to researchers and practitioners for over a decade. The ability to create large-scale virtual societies where virtual entities are modeled individually and the effect of their interactions observed globally is valuable for the study of many important societal issues: the impact of environmental changes, the planning of emergency evacuations, the spread of contagious diseases, etc. In this paper we focus on non-distributed situated MABS, i.e., simulation systems in which virtual agents are situated in a virtual environment which executes on a single machine.

For social simulations to be meaningful, it is necessary to implement realistic models for both virtual agents and environment. A lot of attention has been given to the definition of accurate models for agents (e.g., behavioral, decision-making and interactions models). Unfortunately not much has been done for the definition of virtual environments that mimic the complexity of real-world environments. The reason is twofold:
1. The construction of realistic virtual environments (also called *open* environments) is not a trivial task [12]. Such environments are a) *inaccessible*: virtual agents do not have access to global environmental knowledge but perceive their surroundings through sensors (e.g., vision, auditory, olfactory); b) *non-deterministic*: the effect of an action or event on the environment is not known with certainty in advance; c) *dynamic*: the environment constantly undergoes changes as a result of agent actions or external events; and d) *continuous*: the environment states are not enumerable.
2. Realistic simulations involve the execution of a *large-number* of *sensor-based perception* agents in an *open* environment. Unfortunately, limited computational resources make this goal untenable on a single machine.

A few MABS have proposed models for open virtual environments. Most of these models represent the environment as a single massive component that is managed by one control unit [7, 13, 19]. Other models decompose the environment into regions that are also managed by a single control unit [3,4,21]. In both cases, centralized control creates a bottleneck and limits the scalability of the simulation. On the other hand, a very limited number of MABS have proposed a partitioned structure of the environment with control units managing specific spatial areas [11]. Unfortunately, these systems do not leverage several of the benefits enabled by decentralized control.

In an effort to address the challenge related to the execution of a large number of virtual agents situated in an open environment, researchers have followed two approaches:
1. Distribution. In distributed MABS, the virtual environment is partitioned into regions each hosted on a different machine. While distribution helps to address the problem of scalability, it introduces greater challenges in the design of

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

MABS (e.g., real-time synchronization, cell boundary management, network latency and capacity). In addition, distribution requires a deployment infrastructure that is not available to many.
2. The definition of models that allow the execution of large-scale MABS on a single machine. Given the limited computational resources available, these models aim to achieve a balance between result accuracy and execution time efficiency. In [9], Navarro et. al. propose the use of *aggregation* mechanisms: virtual agents with similar attributes are either aggregated in their entirety in a single entity with lower resolution; or parts of the agents (e.g., decision making processes) are aggregated while others (e.g., perception) continue to execute at their initial levels.

In this paper we propose a model[1] for the execution of large-scale MABS on a single host. In our approach, agents execute their behaviors and are not subjected to any external resource optimization mechanism (e.g., aggregation/disaggregation). The open environment which has a decentralized structure is supported by an underlying self-organizing system consisting of micro- and macro-level specialized agents. The specialized agents continuously interact with one another and re-organize the environment structure to ensure that both the functional and performance simulation requirements are met. During the execution of the simulation, virtual agents are unaware of the partitioned structure of their environment and the self-organization activities occurring at the supporting system layer.

We have implemented the architecture and algorithms in DIVAs, a large scale simulation framework used for the development of simulation systems. The experimental results show the superiority of the proposed self-organizing architecture over the non self-organizing decentralized architecture. The self-organizing virtual environment is scalable, performs better, and the emergence of undesired behavior is controlled.
The rest of the paper is organized as follows: in Section 2 we discuss related work. In Section 3 we discuss the environment model. This is followed by a detailed presentation of the self-organizing algorithms in Section 4. In Section 5 we discuss the implementation of the self-organizing model and algorithms, then present the experimental results.

2. RELATED WORK

A virtual open environment plays a critical role in a situated MABS. It provides the "physical" conditions for the virtual agents to exist, maintains the state of the environment objects, responds to external stimuli, enforces physical laws and informs agents about changes in the virtual world. Agents in an open environment do not have access to global information but perceive their surroundings through sensors. Agent perception is directly related to the structure of the environment and the mechanisms that allow the transfer of environmental data to agents.

Virtual environment structures and their control mechanism are classified as *centralized* or *decentralized*. Environments with a centralized structure are designed as one massive component that can be globally or partially perceived (i.e., agents have access to global knowledge but can perceive some environmental information) [7, 10, 13, 15, 19]. These environments are managed by a single control unit. These systems are unfit to simulate realistic scenarios where agents are expected to act solely upon the information they have perceived. In addition, the large amount of environmental data to be processed by agents limits the size and complexity of the simulation.

Environment with decentralized structures are partitioned into multiple distinct regions that are managed in a centralized or decentralized fashion. In a partitioned environment, the amount of data sent to virtual agents is drastically reduced. This naturally lowers the computational cost. Decentralized structures with central control have been widely used to implement simple 2D environment models in which agents have access to global [2, 3, 16, 20, 21] or partial knowledge [4]. For all MABS in this category, the centralized control creates a bottleneck and limits the scalability of the simulation.

The limitation of centralized control is addressed in environments with decentralized structure and control. In [11], Pelechano et al. discuss HiDAC, a crowd evacuation simulation system used to model realistic flow of people in evacuation scenarios. The environment represents a set of rooms, each corresponding to an environment cell. In order to obtain environmental information, HiDAC agents query the cell in which they are situated. The division of the environment into rooms simplifies the perception problem: the presence of walls eliminates the need for boundary cases, e.g., when agents belonging to a cell are able to perceive other cells.

In order to address the problem of building large-scale realistic simulations executing on a single machine, Navarro et al. [9] propose an agent aggregation/disaggregation approach at a resolution level called *mesoscopic*. Agents are decomposed into processes, each representing a skill (e.g., navigation, perception). Based on the value of an aggregation utility, some of the agent processes are aggregated while others continue to run at their initial levels. While the concept is interesting and the experimental results are strong (scenarios with 1,000 agents in a highly dynamic environment and 10,000 agents in a less interactive environment executing on a single machine), due to the proprietary nature of the simulator used to test the model, several important evaluation features are unclear: the problem of determining which agents are to be aggregated/disaggregated seems to be done by a central unit which would normally constitute a bottleneck for large scale dynamic MABS; it is unclear whether the environment is open; it is unclear how agent perception is implemented (the authors only mention the use of Smart Objects) and whether the approach can accommodate senses other than vision. In addition, as stated by the authors, the proposed approach does not lead to computational gains when the environment is highly dynamic and interactions between agents and environment are intensive.

In this paper we propose an orthogonal approach to Navarro et al. We present a self-organizing model for virtual environments that can be used to improve the scalability and performance of MABS executing on a single machine. In this model, agent execution is not altered. The decentralized environment is supported by an underlying self-organizing system that reorganizes the environment structure to en-

[1]In the remainder of this paper, the term "model" is used to refer to concepts and architecture.

sure that both the simulation accuracy and performance are met. Our proposed model improves on the state of the art in virtual open environments as follows: 1) it can be used to create any type of spatial environment; 2) virtual agents perceive their surroundings through advanced multi-sensory perception mechanisms (e.g.,vision, auditory, olfactory) [1]; 3) The underlying autonomic structure is orthogonal to the simulation and virtual agents are unaware of its execution; 4) the performance and scalability of the simulation surpass static centralized and decentralized solutions.

In the next section we give an overview of the autonomic reference architecture at the basis of this work.

3. A SELF-ORGANIZING MODEL FOR VIRTUAL ENVIRONMENTS

In order to manage a large virtual environment efficiently, it is necessary to partition the space into smaller defined areas called *cells* (see Figure 3). Each cell is managed by a specialized agent called a *controller* [1].

Cell Function Management

A controller is responsible for a) managing environmental information about its cell; b) informing its neighboring cell controllers of propagating events; and c) providing each local virtual agent with its perceivable portion of the environment. We refer to these responsibilities as *cell function management*.

In the self-organizing model, agents and environment interact continuously using the Action Potential/Result (APR) model [17], an enhancement of the influence/reaction model proposed by Ferber [5]. In APR, agents receive the state of the cells they can perceive; using their sensors, they determine what can actually be sensed; they decide which actions to perform; and send their intended actions to their cell controllers. The controllers receive the action potentials, combine them, resolve conflicts, determine the new states of their cells and send the updated states to their respective agents (see Figure 1). This entire process represents one simulation cycle. A detailed description of the APR model can be found in [17].

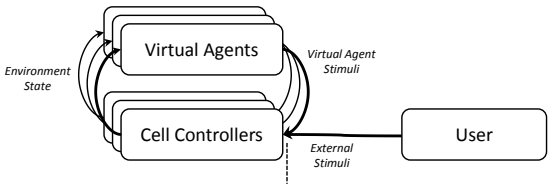

Figure 1: Agent environment interactions

When an agent receives environmental data from cell controllers, its perception sensors filter out any information that is not perceivable. This filtering can be computationally intensive when the amount of irrelevant information is large. Therefore it is a cell controller's responsibility to ensure that only the smallest aggregate of cells whose bounds fully contain the agent's perception range is passed on to the agent (e.g., cells $c3$, and $c4$ in Figure 2). This is a computationally intensive task that involves combining the agent intended actions to ensure that the results of these intentions are legal with respect to the laws of the environment; processing boundary cases; updating the cell state; and sending the relevant information to the agents [17]. Detailed discussions on agent perception can be found in [8].

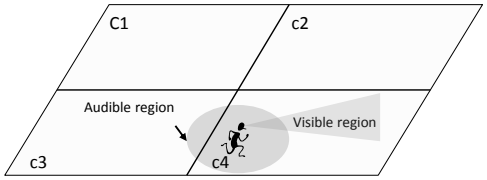

Figure 2: Range of perception within multiple cells

Cell Performance Management

In highly dynamic scenarios, a large number of agents may find themselves concentrated in only a few cells. The uneven workload distribution among controllers may lead to 1) a delayed simulation cycle and 2) a waste of computational resources. In order to address these issues, we need to supplement the controller's responsibilities with *simulation performance management*. In other words, in addition to managing their cells and providing their agents with perceivable information, each cell controller has to make sure that a) its workload is under nominal capacity and b) the performance requirements (e.g., simulation cycle time) are met. If the performance constraints are not met, a controller has to re-organize its cell by performing one of two actions: splitting its cell, transferring a portion of its agents, and spawning a new controller to manage the new cell; or merging its cell with another cell and releasing its resource. While at every simulation cycle both functional and performance tasks are executed by controllers, in this paper, we focus our discussions on the controller's performance management role.

As shown in Figure 3, the self-organizing system includes an additional category of specialized agents called *coordinators*. A coordinator's role is to ensure that the simulation performance requirements for the set of controllers it supervises are met. The definition of coordinators is necessary since purely decentralized reorganization can lead to undesired behavior and performance. This is illustrated in Section 5.4. Nevertheless, it is important to note that coordinators are centers of *knowledge* and *advice* rather than centers of control. From a behavioral perspective, since a coordinator can oversee a limited number of cell controllers, when its load increases or decreases, it either spawns a new coordinator and passes part of its load on to it, or merges its load with another coordinator and destroys itself.

During the execution of the simulation, the self-organizing system has to find a balance between multiple competing objectives: a) reduce the agent perception cost; b) reduce the cost incurred by a controller to determine which portion of the environment is perceivable by an agent. We refer to this as the *controller perception processing cost*; c) reduce the context switching cost; and d) reduce the hierarchy self-organization cost. Automating the trade-off between competing objectives is hard to achieve and is still unsupported in most current systems [6]. While considering self-organization exclusively from a computational resource

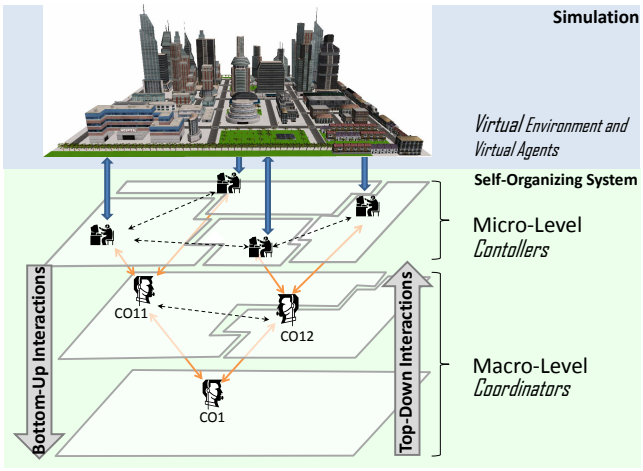

Figure 3: Self-organizing environment

utilization perspective (i.e., load balancing) is possible, it is less obvious to do so in the presence of competing goals [6].

4. A SELF-ORGANIZING SYSTEM

We start this section by defining the concepts integral to the model discussed above, then present self-organization algorithms.

4.1 Definitions

Controller workload: A controller c_k's workload ($W_{c_k}(t)$) corresponds to the amount of work a controller has to perform to: a) manage its local environmental information such as objects and b) provide its local agents with their perceivable portion of the environment. As such, $W_{c_k}(t)$ is defined as:

$$\mathrm{W}_{c_k}(t) = agents_{c_k}(t) \cdot w_1 + objects_{c_k}(t) \cdot w_2$$

where $agents_{c_k}(t)$ is the effort needed to manage agents located in c_k's cell at time t; $objects_{c_k}(t)$ is the effort needed to manage the environment objects in c_k's cell at time t; w_1 and w_2 are weights; $w_1, w_2 \in [0, 1]$.
Given that it takes more processing power to handle agent perception than environment objects, $w_1 > w_2$. These weights can be calibrated by the user of the simulation.

Simulation Resources: In the simulation, controllers are run as threads. Therefore, the maximum number of controllers Γ available for the simulation depends on the computer's processing power. For example, in case of a PC using a core i7 with 12 processors, we have found that $\Gamma = 64$. This corresponds to the maximum number of threads that can be run by the computer before context switching overhead leads to performance degradation.

At initialization time, Γ is assigned to the initial coordinator. As the simulation evolves and controllers and coordinators are created and destroyed, the distribution of Γ among coordinators adapts to the simulation's computational need. At any point in time, $\Gamma = \sum_{i=1}^{|CO|} \gamma_{co_i}(t)$ where $\gamma_{co_i}(t)$ corresponds to the number of threads assigned to coordinator co_i at time t and $|CO|$ is the total number of coordinators in the simulation.

Coordinator workload: A coordinator co_i's workload, W_{co_i} represents the workload necessary to oversee a number of controllers or coordinators.

Urgency Function: The urgency function U_{c_k} is used by a coordinator to determine which controller c_k requires immediate assistance. It is defined as:

$$\mathrm{U}_{c_k}(t) = \mathrm{W}_{c_k}(t) + \rho_{c_k}(t) \cdot w_\rho$$

where $\mathrm{W}_{c_k}(t)$ is the workload of c_k, $\rho_{c_k}(t)$ is the average workload of c_k's neighboring controllers and w_ρ is a weight having a value between $[0, 1]$. $\rho_{c_k}(t)$ is defined as:

$$\rho_{c_k}(t) = \tfrac{1}{N} \sum_{n=1}^{N} \mathrm{W}_{c_{k,n}}(t),$$

where $W_{c_{k,n}}$ is the workload of adjacent controllers and N is the total number of surrounding cells.

To illustrate the use of U_{c_k}, consider the scenario where two non-neighboring controllers c_1 and c_2 are overloaded, i.e., given a threshold α, $\mathrm{W}_{c_1}(t) > \alpha$, $\mathrm{W}_{c_2}(t) > \alpha$, and $\mathrm{W}_{c_1}(t) \approx \mathrm{W}_{c_2}(t)$. If the average workload $\rho_{c_1}(t)$ around c_1 is greater than the average workload $\rho_{c_2}(t)$ around c_2, then $\mathrm{U}_{c_1}(t) > \mathrm{U}_{c_2}(t)$ and c_1 will have a higher priority in getting assistance from the coordinator.

4.2 Algorithms for Self-Organization

At initialization time, a newly created virtual environment consists of a single cell $cell_1$. The self-organizing system assigns a single controller c_1 to that cell. Then coordinator co_1 is created, its available resources are set to Γ, and c_1 is added to its set of overseen controllers C_{co}.

In the remainder of this section, we use Figure 4 to illustrate the various steps of the self-organizing algorithms. At every simulation cycle, a controller c_k monitors its workload. If its workload is greater than a threshold α (i.e., it can no longer finish processing agent perceptions and cell information management within the real-time requirements of the simulation) it determines that it needs to split its cell and spawn a new controller to alleviate its workload. To this effect, it requests advice from its coordinator co (step 1 in Figure 4). The request message is enqueued in co's request list. co retrieves and analyzes all requests periodically. It computes and determines which cell controller has the highest level of urgency and stores the information in c_{max}. If $c_k = c_{max}$ and the number of controllers $|C_{co}|$ supervised by the coordinator is lower than γ_{co}, then co advises c_k to perform a split. In case $|C_{co}|$ is equal to γ_{co} (i.e., the coordinator has consumed all its resources and no splitting is possible), co executes an optimization algorithm. This algorithm aims at determining if the coordinator can obtain computational resources by merging controllers with less critical urgency values. This algorithm determines if the sum of the urgency values of the two less critical neighboring controllers in C_{co} is lower than the running average on the urgencies of past and current c_{max}. The running average $U_{critical}$ allows to address oscillations in urgency values related to cases when agents cross cell boundaries. If this is the case, co advises the merging of the two less critical controllers. This allows the critical controller to split and results in a more uniform distribution of workload and resources.

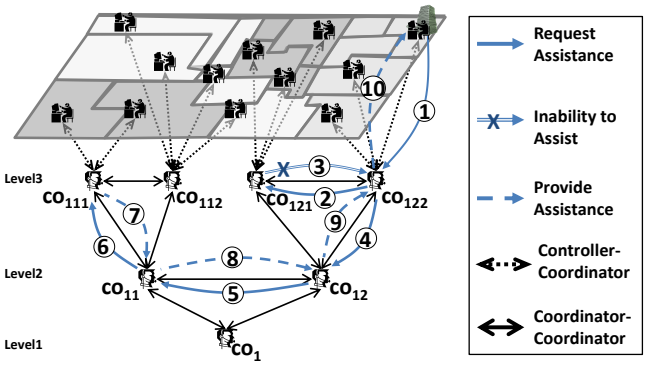

Figure 4: Request for assistance scenario

If there are no controllers with less urgency values to be merged, *co* interacts with its neighbor coordinator to request assistance (step 2 in Figure 4). Upon receipt of the request, the receiver coordinator co_{rec} checks its ability to provide assistance to *co*.

If both *co* and co_{rec} (co_{121} in Figure 4) are leaf coordinators (step2) and co_{rec} has not consumed all of its available resources, it informs *co* (co_{122}) of the availability of the requested resources by sending a reply message. In case co_{rec} (co_{121}) has consumed all of its available resources, it determines if it is possible to merge the two less critical neighboring controllers in its set to release the required resources. If it is possible, a reply message is sent to *co* (co_{122}) to inform it of the available resources. If not, co_{rec} (co_{121}) informs *co* (co_{122}) that it is not able to provide assistance (step 3).

Since *co* (co_{122}) could not receive assistance from its neighboring coordinator (co_{121}), it requests assistance from its parent coordinator (co_{12}, step 4). The parent coordinator attempts to find resources by contacting a neighboring coordinator (co_{11}, step 5). The receiving neighbor determines which among its children coordinators has the highest available resources relative to its load. This is defined as

$$\Pi_{co}(t) = \frac{\gamma_{co}(t) - |C_{co}(t)|}{\left(U_{co}(t) \times w_U + area_{co}(t) \times w_A\right)}$$

where $U_{co}(t)$ is the total urgency of the set of controllers overseen by *co* at time t, $area_{co}(t)$ is the area of the cells managed by the set of controllers overseen by *co* at time t, w_U and w_A are weights. $w_U, w_A \in [0, 1]$.

It sends a request for assistance to the child coordinator, co_{max} (co_{111},step 6). co_{max} (co_{111}) replies to its parent coordinator. If resources are available, the parent coordinator passes the information onto the requester coordinator (co_{12}, step 8). The requester coordinator updates its available resources value, and forwards a reply to the request initiated by coordinator, *co* (co_{122}, step 9). *co* (co_{122}) updates its available resources value, and informs the requesting controller c_k of the possibility of splitting.

If none of the children coordinators are able to provide assistance, the request is passed onto higher level coordinators until either it is satisfied or the top coordinator is reached and no help can be provided (exhausted all possibilities).

Once the controller receives an acknowledgement, it proceeds by splitting its cell, spawning a new controller as a result.

If during the execution of the simulation a coordinator's workload falls below a threshold χ, *co* offers to merge with its neighbor $co_{neighbor}$. $co_{neighbor}$ accepts to merge if the total workload does not exceed χ. Otherwise, if the coordinator *co* workload goes above a threshold Ω, *co* spawns a new coordinator and passes a part of its load on to it.

In addition to the autonomic behavior discussed above, coordinators exhibit a proactive behavior to ensure an optimal distribution of resources among coordinators. Each non-leaf parent coordinator periodically monitors its children and, when necessary (e.g., the coordinator realizes that due to reorganization activities, the resources are not distributed optimally over the children coordinators), sends a *redistributionAdvice* message to each child. The message provides an advice about how to redistribute the resources to achieve an optimal distribution.

The advised optimal resource distribution $\gamma_{advised}$ for a coordinator *co* is defined as

$$\gamma_{advised}(t) = \left(\frac{U_{co}(t) \times w_U + area_{co}(t) \times w_A)}{\sum_{co_c \in children(co_p)} (U_{co_c}(t) \times w_U + area_{co_c}(t) \times w_A)} \right) \times \sum_{co_c \in children(co_p)} \gamma_{co_c}$$

where $area_{co}(t)$ is the area of cells managed by the set of controllers overseen by *co* at time t, $U_{co}(t)$ is the total urgency of the set of controllers overseen by *co* at time t, w_U and w_A are weights. $w_U, w_A \in [0, 1]$.

When a child coordinator co_{child} receives the advice message, it evaluates the proposed distribution based on its current load. If all children agree to follow the advice then they proceed by redistributing their resources. Following this step, each co_{child} propagates an optimization advice down through the coordinators hierarchy.

In addition to dedicating themselves to the management of their cells, in certain circumstances, controllers exhibit altruistic behavior by self-organizing themselves to benefit the simulation system as a whole. If controller c_k's workload falls below a threshold β, it informs its neighboring controller that its resources are ready to be released by sending a message containing the value of its workload. Upon receipt of the message, if the neighboring controller c_s has the ability to handle the combined load, it initiates the merge of the two cells and destroys c_k. Since controllers are managed by separate threads and consume memory and CPU, the merging results in resource savings for the system as a whole. Examples of resource savings include optimization of memory allocated to controller threads and minimization of thread context switching.

5. MODEL IMPLEMENTATION AND EVALUATION

The proposed self-organizing model has been fully implemented and tested using DIVAs, a Java-based framework for the development of large-scale agent-based simulation systems. In order to provide a fair assessment of the three structures used in the current state-of-the-art simulators namely *centralized control* ([**?**, 2–4, 7, 10, 13, 15, 16, 19–21]) and *decentralized control with decentralized structure* ([11]) (see Section 2), we felt that it was necessary to implement these structures from scratch using the same settings, then collect and analyze data. Please note that DIVAs agent perception

(i.e., vision, hearing, smell) and agent-environment interactions (i.e., APR model) are very complex when compared to existing simulators.

The simulation scenarios were executed on a multicore PC (Intel Core i7 X980 CPU (3.33GHz), 6.00 GB, 64-bit Windows 7). Controllers run on a thread execution pool and coordinators are implemented as daemon threads. The total number of worker threads for the experiment is set to 12 which corresponds to the number of processors available on the computer.

5.1 Experiment Setting

The DIVAs framework was used to build a social simulation system where virtual agents representing humans are situated in an open environment representing a city. Using the simulator, we ran three experiments and evaluated our model with respect to *simulation cycle time*, i.e., the time elapsed in each simulation cycle measured in milliseconds and *CPU utilization*, i.e, the percentage of CPU used by the simulation.

All experiments take place in a virtual city environment consisting of 814 environment objects (e.g., commercial buildings, houses, traffic signals) and three parks (see Figure 5). Situated agents represent humans perceive their surroundings through advanced vision, auditory and olfactory sensors. They execute complex path-finding and collision avoidance algorithms to move within the environment. In addition, agents interact with other agents, plan and deliberate to achieve their goals (e.g., move to location, search for agents).

Figure 5: 3D visualization of the city park

The thresholds and parameters used to execute these experiments are: $\alpha = 25$, $\beta = 5$, $\Gamma = 64$, $w_\rho = 0.25$,$w_1 = 1$,$w_2 = 0.5$,$w_U = 0.75$,$w_A = 0.25$, $\chi = 4$, and $\Omega = 10$. Where α is the workload threshold that must be reached for a controller to request assistance, β is the workload threshold that must be reached for a controller to merge with its neighbor, Γ is the resources for the simulation, $\{w_\rho$, w_1 , w_2 , w_U , w_A $\}$ are weights, Ω is the workload threshold that must be reached for a coordinator to split its workload over a spawned coordinator, χ is the workload threshold that must be reached for a coordinator to merge.

Demonstrations for the experiments discussed below are available at [18].

5.2 Experiment 1: Scalability

The objective of this experiment is to show the impact of different environment structures on the scalability of the simulation. For each execution of the simulation, we record the maximum number of agents handled by the simulation without violating the specified simulation cycle time requirement of 150 milliseconds. This corresponds to the longest time the visualizer can display the simulation without delay. In this experiment, agents are scattered *evenly* in various areas of the city. We run this experiment using two environment structures:

Static multiple cells (MC): this structure refers to a non self-organizing environment containing equally sized cells managed by controllers. This structure does not include coordinators.

Self-organizing (SO): this structure refers to the proposed self-organizing model discussed in Section 3.

The experiment starts with a single cell and gradually increases to 64 cells.

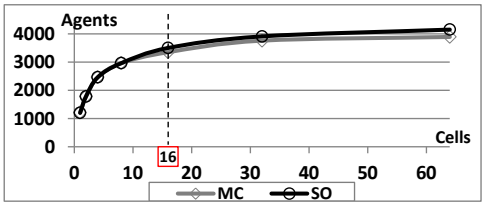

Figure 6: Experiment 1 Scalability in terms of number of agents

Figure 6 shows the number of agents that can be handled by each environment structure before reaching the maximum simulation cycle time of 150 milliseconds. These results show that MC and SO can handle a comparable number of agents. This is to be expected as an even agent distribution does not call for reorganization.

The results also show an approximate 300% increase in the number of agents that the simulation can handle from the single cell to the 64 cell environment. We also observe that as the number of cells increases, the maximum number of agents asymptotically increases until reaching 16 cells. This improvement is due to the savings in agent perception cost (discussed in Section 3). After reaching 16 cells, the controller perception processing cost and the context switching cost are such that the improvement in scalability is insignificant.

In the remainder of this section, we restrict our discussion to *MC* and *SO* with 64 cells and discuss cases where the self-organizing environment drastically outperforms the static partitioned environment model.

5.3 Experiment 2: Performance Improvement

The objective of this experiment is to demonstrate the suitability of the proposed model to cope with changes in an open virtual environments. In this experiment, virtual human agents run errands in the city when an external event related to the occurrence of a music festival is triggered. The virtual agents attempt to reach the festival that takes place in a public park. We run this scenario with 700, 1400, and 2100 agents using *MC* an *SO*.

As the simulation begins, the self-organizing environment partitions itself autonomously according to the crowd den-

sity. The movement of the crowd results in a dense population of agents flowing towards the location of the park. As the concentration of agents increases, the environment using static multiple cells has a few highly populated cells while its remaining cells are almost empty (see Figure 7). On the other hand, the self organizing environment naturally copes with the changes in the distribution of the virtual agent population: the cells are more evenly populated.

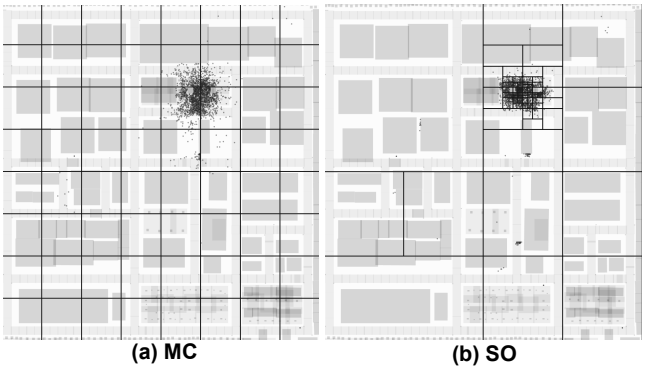

Figure 7: Experiment 2 - Final configuration with 2100 agents

Population imbalance has a direct impact on the simulation performance. This is illustrated in Figure 8 which shows the simulation cycle times for the static multiple cells and the self-organizing environment populated with 700, 1400, and 2100 agents. For these cases, the self-organizing environment outperforms the static partitioned environment in terms of average simulation cycle time. As shown in Figure 8 (a), when the population of agents is small, both structures are able to cope with the simulation real-time requirements. However, with larger agent populations (see Figure 8 (b)-(c)), as virtual agents move towards the location of the festival, the simulation cycle time increases asymptotically in the static cell structure, reaching a point where the cycle time exceeds the maximum time of 150 milliseconds needed by the visualizer to display the simulation without delay. This is not the case for the self-organizing environment which is able to promptly respond to the dynamics of the simulation.

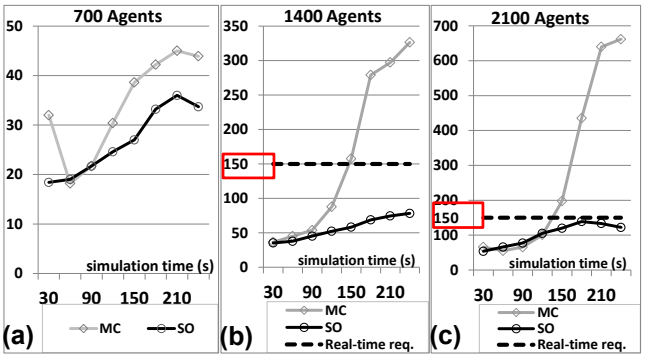

Figure 8: Experiment 2 - Average simulation cycle times with (a) 700, (b) 1400, and (c) 2100 agents

Figure 9 illustrates the utilization of CPU by the simulation with 700, 1400 and 2100 agents. The results show that the static multiple cell uses more CPU in all cases. This is due to the fact that, the restructuring of the environment results in smaller cells for highly populated areas. Therefore, perception in these smaller cells is less computationally intensive for a larger number of agents.

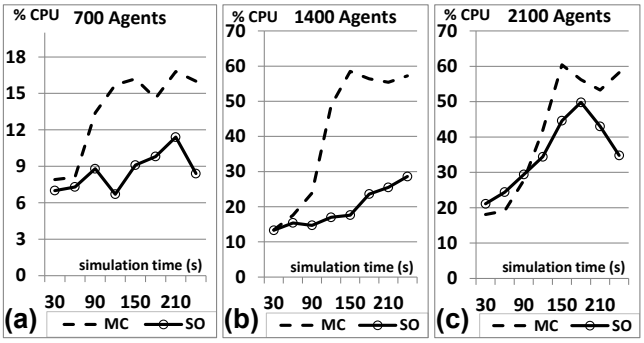

Figure 9: Experiment 2 - CPU utilization with (a) 700, (b) 1400, and (c) 2100 agents

5.4 Experiment 3: Control of Undesirable Behavior

Studies have shown that, in the absence of some level of control, fully decentralized self-organizing systems may result in undesired chaotic states [14]. The purpose of this experiment is to demonstrate how the emergence of undesired behavior is controlled in the context of the self-organizing environment structure presented in this paper. To this effect, we create an environment structure where cell controllers perform reorganization tasks autonomously, without any guidance or control from coordinators. We call this structure non-coordinated self-organizing environment (NCSO).
In this experiment, we add 3504 virtual agents in the city, initially statically partitioned into 32 cells. We run this experiment with the non-coordinated environment structure and the fully self-organizing structure.

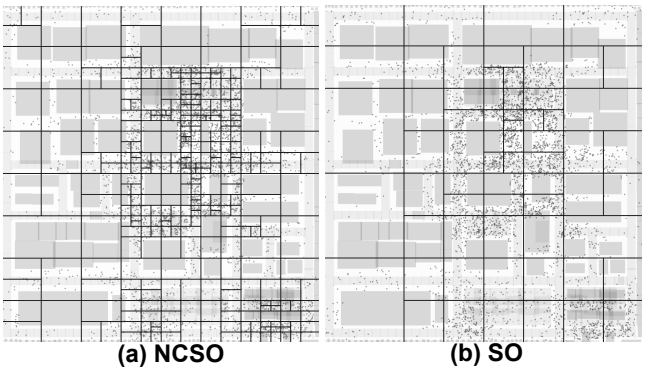

Figure 10: Experiment 3 - Emergence of Undesired State

As shown in Figure 10 (a), in the absence of coordinators, cell controllers deliberate over reorganization tasks based solely on their knowledge of their local cells and surroundings. Since they do not possess knowledge about the overall simulation properties, when their workload goes above a threshold, cell controllers decide on their own to split their

cells. This results in spawning a large number of new controllers. This undesired behavior can lead to complete starvation of computational resources and an additional computational overhead which would result in the degradation of the overall system performance. On the other hand, Figure 10 (b) shows that the splitting is more moderate. This is due to the fact that controllers are guided by coordinators whose goal is to maintain the overall system performance requirement.

6. CONCLUSION

In this paper we presented a self-organizing model for decentralized virtual environments in MABS. This model is based on the premise that a virtual environment structure is supported by an underlying software system consisting of cell controllers and coordinators. These specialized agents continuously interact with one another and re-organize themselves and the environment structure to ensure that the simulation functional and performance requirements are met. During the execution of the simulation, virtual agents are unaware of the partitioned structure of their environment and the self-organization activities.

We have implemented the architecture and algorithms using DIVAs, a framework for the development of MABS. The experimental results show the superiority of the proposed self-organizing architecture over existing architectures. The virtual environment scales very well in highly dynamic scenarios where virtual agents are in constant motion in the spatial environment. In addition, the emergence of undesired behavior is controlled.

In the current model implementation, cells are split evenly even if agents and objects are not uniformly distributed in the cell area. It may be interesting to implement an algorithm that performs the split based on the agent/object distribution or allows the cell to be split in more than two cells. Also, when providing assistance to a coordinator in need of additional resources, a helper coordinator releases one thread at a time even though the need may be for more. A possible solution would be for the helper coordinator to implement a heuristic that determines how many threads can be released based on present and predicted future loads. Finally, we believe that the work of Navarro et al. [9] is complementary to ours since it tackles the same problem from a different perspective. It would be interesting to incorporate Navarro's mesoscopic model with our self-organizing environment model.

REFERENCES

[1] M. Al-Zinati, F. Araujo, D. Kuiper, J. Valente, and R. Z. Wenkstern. DIVAs 4.0: A Multi-Agent Based Simulation Framework. In *Proceedings of the 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2013)*, pages 105–114, Delft, Netherlands, November 2013.

[2] S. Bandini, M. Federici, S. Manzoni, and G. Vizzari. Pedestrian and crowd dynamics simulation: Testing sca on paradigmatic cases of emerging coordination in negative interaction conditions. *Parallel Computing Technologies*, 4671:360–369, 2007.

[3] B. Banerjee, A. Abukmail, and L. Kraemer. Advancing the layered approach to agent-based crowd simulation. In *Proceedings of the IEEE Workshop on Parallel and Distributed Simulation*, pages 185–192, Rome, Italy, June 2008.

[4] A. Braun, B. Bodmann, and S. Musse. Simulating virtual crowds in emergency situations. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 244–252, Monterey, CA, USA, November 2005.

[5] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley, 1999.

[6] D. Garlan, B. Schmerl, and S.-W. Cheng. Software architecture-based self-adaptation. In *Autonomic Computing and Networking*, pages 31–55. Springer US, 2009.

[7] W. L. Koh and S. Zhou. An extensible collision avoidance model for realistic self-driven autonomous agents. In *Proceedings of the IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 7–14, Chania, Greece, Oct 2007.

[8] D. Kuiper and R. Z. Wenkstern. Agent vision in multi-agent based simulation systems. *Journal of Autonomous Agents and Multi-Agent Systems*, pages 1–31, 2014.

[9] L. Navarro, V. Corruble, F. Flacher, and J.-D. Zucker. A flexible approach to multi-level agent-based simulation with the mesoscopic representation. In *Proceedings of the international conference on Autonomous agents and multi-agent systems*, pages 159–166, St. Paul, MN, USA, May 2013.

[10] X. Pan, C. Han, K. Dauber, and K. Law. A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations. *Ai & Society*, 22(2):113–132, 2007.

[11] N. Pelechano, J. Allbeck, and N. Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108, San Diego, California, USA, August 2007.

[12] S. Russell and P. Norvig. *Artificial Intelligence A modern approach*. Prentice-Hall, Egnlewood Cliffs, Third edition, 2010.

[13] S. Rymill and N. Dodgson. Psychologically-based vision and attention for the simulation of human behaviour. In *Proceedings of the ACM international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 229–236, Dunedin, New Zealand, November 2005.

[14] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter. Adaptivity and self-organization in organic computing systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 5(3):10, 2010.

[15] S. Sharma. Simulation and modeling of group behavior during emergency evacuation. In *Proceedings of the IEEE Symposium on Intelligent Agents IA'09*, pages 122–127, Nashville, TN, USA, March 30 - April 2 2009. IEEE.

[16] J. Shi, A. Ren, and C. Chen. Agent-based evacuation model of large public buildings under fire conditions. *Automation in Construction*, 18(3):338–347, May 2009.

[17] T. Steel. *A Self-Organizing Environment for Distributed Multiagent-Based Simulation*. PhD thesis, University of Texas at Dallas, November 2010.

[18] The Multi-Agent and Visualization Systems Lab. DIVAs. `http://mavs.utdallas.edu/projects/divas` Accessed November 2014.

[19] K. Uno and K. Kashiyama. Development of simulation system for the disaster evacuation based on multi-agent model using GIS. *Tsinghua Science & Technology*, 13(1):348–353, October 2008.

[20] J. Was. Cellular automata model of pedestrian dynamics for normal and evacuation conditions. In *Proceedings of the IEEE International Conference on Intelligent Systems Design and Applications*, pages 154–159, Wroclaw, Poland, September 2005.

[21] M. Xiaofeng, W. Chaozhong, and Y. Xinping. A multi-agent model for evacuation system under large-scale events. In *Proceedings of the IEEE International Symposium on Computational Intelligence and Design*, pages 557–560, Wuhan, China, October 2008.