Modeling Agent-Environment Interactions in Large-Scale Multi-Agent Based Simulation Systems

Mohammad Al-Zinati
Jordan University of Science and Technology
Irbid, Jordan
mhzinati@just.edu.jo

Rym Wenkstern
The University of Texas at Dallas
Richardson, Texas
rymw@utdallas.edu

ABSTRACT

In this paper we present the Action-Potential/Result (APR) model for agent-environment interactions in Multi-Agent Based Simulation systems (MABS) involving thousands of perception-based agents executing on a single host. The environment structure is partitioned into cells which are managed by specialized agents called *controller* and *coordinator*. The agents send their stimuli to the controller managing the cell in which they are situated. The controller assesses the received agent stimuli as well as user-triggered and events propagation stimuli. It combines them, attempts to resolve potential conflicts, communicates with adjacent controllers and the coordinator, and ensures that the updated environment state is communicated back to its agents. The APR model has been implemented as a component of the DIVAs framework and can be reused with minimal changes for the construction of agent-based simulations with DIVAs. Experimental results show a significant improvement in scalability over conventional centralized solutions.

CCS CONCEPTS

• **Computing methodologies → Agent / discrete models**;

KEYWORDS

Multi-Agent Simulation Systems; Interaction Model

ACM Reference Format:
Mohammad Al-Zinati and Rym Wenkstern. 2019. Modeling Agent-Environment Interactions in Large-Scale Multi-Agent Based Simulation Systems. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Real-world environments are complex systems which are *open* [26]. They are *inaccessible*, i.e., they cannot be observed in their entirety but partially perceived through sensors (e.g., vision, auditory, olfactory); *non-deterministic*, i.e., the effect of an action or event on the environment is not known with certainty in advance; *dynamic*, i.e., the environment constantly undergoes changes as a result of agent actions or external events; and *continuous*, i.e., the environment states are not enumerable.

The development of realistic large-scale[1] Multi-Agent Based Simulation Systems (MABS) with *open* environments poses significant challenges: thousands of virtual agents need to partially perceive their dynamic surroundings through sensors, deliberate, then act upon a spatial (i.e., non-grid, non-graph based) environment in simulated real-time; the virtual agent's actions have to obey the physical laws of the environment (e.g., two agents cannot be in the same position at the same time); and, to improve user experience, interaction mechanisms with the simulation (e.g., trigger random explosion, modify agent properties) need to be provided. Moreover, in order for researchers to benefit from the availability of such MABS, these systems need to execute on a single computer with reasonable computational resources.

From an engineering perspective, the construction of MABS with the features discussed above requires a complete separation of concerns between agents and the environment. As such, distinct abstract models for agents and the environment need to be defined [20, 34, 36]. Years of research and experimentation with various environment structures led us to conclude that open virtual environments are best modeled as multi-agent systems [1]. In order to tie the agents model with the environment model, it is necessary to define a model of interaction between these components.

Researchers have proposed models of interaction between agents and environment [12, 19]. These models are based on Ferber and Muller's seminal Influence-Reaction Model (IRM) [8] which posits that agents influence their environment which in turns reacts to these influences. Although these formal models have been validated on small examples, their extension to large scale problems has not been investigated, and their implementation in large-scale interactive simulation systems has not been achieved.

In this paper we present the Action-Potential/Result model (APR), a model of interaction between agents and environments with decentralized structure. The APR model improves existing IRM-based models as follows:

(1) It is defined to handle thousands of perception-based virtual agents situated in a virtual open environment, and executing on a single computer.
(2) It considers an underlying environment structure which is split into cells.
(3) The environment is modeled as a multi-agent system where specialized agents called *cell controllers* manage individual cells. Cell controllers are in turn managed by a *coordinator*;
(4) The two phased model involves interactions between agents and controllers, controllers and controllers, and controllers and coordinator;

[1] In the remainder of this paper the term *large* refers to the execution of thousands of perception-based agents.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

(5) It considers interactions with the user of the simulation at run time by processing externally triggered events;
(6) It has been implemented and thoroughly tested on several large-scale scenarios.

The DIVAs (**D**ynamic **I**nformation **V**isualization of **A**gent **s**ystems) framework [1] provides the basic building blocks, i.e., abstract models, architectures, software components and libraries for the development of a variety of agent-based simulation systems. APR has been implemented as a core component of DIVAs and can be reused with minimal changes for the construction of application-specific agent-based simulations. As such, it allows researchers to shift their focus from the engineering of complex structural mechanisms to the specification and simulation of application-specific virtual agents and environments.

The remainder of this paper is organized as follows: Section 2 discusses related works. Section 3 introduces fundamental concepts. Sections 4 and 5 present the APR model. Section 6 discusses experimental results.

2 AGENT-ENVIRONMENT INTERACTIONS IN MABS

Over the past decade, a plethora of application-specific agent-based simulations systems has been discussed in the literature. These systems were mainly developed by researchers to validate agent models and algorithms in various areas such as social sciences [27, 28, 31, 33], urban planning [11, 13, 25, 37] and transportation [4–6, 10] to name a few. Naturally, application-specific simulation systems do not generally account for the separation between agent and environment concepts and are not concerned with the engineering of elaborate environment models and interaction protocols to simulate *openness*. For most, either the environment has complete control over the agents' states, or agents have complete control over their own state. Only a very few application-specific simulation systems distinguish between agents and environment [9, 23], but none discusses agent-environment interaction mechanisms in detail. [23] discusses High-Density Autonomous Crowds (HiDAC), a crowd simulator where agents move according to rules of force. In HiDAC, agents and environment are tightly coupled and the environment exists as a low-level decision-making process for motion. [9] gives an overview of the environment model of JASIM, a simulator for traffic and crowd simulation. JASIM's environment model is distinct from the agent model and uses "pipelines" to compute agent perceptions from a hierarchical or graph-based data structure. It updates its state, based on the agent actions. JASIM's environment is centralized and the processing of agent perception and motion by the environment decreases the simulation performance.

In addition to application specific simulators, a number of multi-agent simulation toolkits have been proposed. These toolkits are intended to ease the development of simulation systems for specific application domains [14, 24, 35]. While these toolkits offer integrated graphical environments for the execution of simulations, they are based on simple environment structures (e.g., two-dimensional grid or graph) with centralized control. Finally, a few generic agent-based simulation frameworks have been developed and used by the agent simulation community [17, 22, 30]. While these frameworks offer libraries of components that can be used for the construction of simulators, only a few distinguish between agent and environment concepts [22, 30], but none offers mechanisms to simulate large-scale interactive agent-based simulations with open, spatial environments.

Among the first attempts at formalizing agent-environment interactions in multi-agent systems is Ferber's seminal Influence-Reaction Model (IRM) [7, 8]. The original IRM presents a general model of action which posits that agents *influence* their environment, which in turn reacts to these influences. The first phase includes agents perceiving, deliberating and executing actions. The second phase is the environment's merging and *reacting* to influences. IRM was developed to model *tropistic* and *hysteretic* agents evolving in a centralized environment. IRM set the stage for a research area on agent-environment interactions: in [19] Michel proposes IRM4S, an adaptation of the original IRM for Multi-Agent Simulation Systems (MABS). IRM4S introduces a temporal variable to clarify Ferber's original action model, and adds environment influences (e.g., a rolling ball in the environment) in the formalism. Another variation of IRM was proposed by Helleboogh et al. [12]. In this model, the environment is given the responsibility to assess the effect of influences and, when necessary, replace the original agent "activities" with ones that do not contradict the physical laws of the environment. This responsibility creates a strong coupling between agents and the environment. In [21], Morvan et al. discuss a meta-model for the specification of multi-level agent-based systems. This model called IRM4MLS is an extension of Michel's IRM4S model and considers cases where agents belong to several levels, (e.g., a lower-level an automated guided vehicle and higher-level conflict solver) or emerge from lower levels.

The Influence-Reaction models discussed above define interaction models in the context of a centralized environment structure which limits scalability. In addition, environments are considered as reactive systems. Moreover, given the focus on formal model definitions, the examples provided are simple and there is no evidence of the implementation and validation of the models for large simulations with open environments.
In this paper we present the Action-Potential/Reaction model (APR) for agent-environment interactions in large-scale simulation systems with open environments. APR extends IRM and IRM4S as follows:

(1) It defines the environment model as a decentralized structure managed by specialized agents. As such, from a structural perspective, the environment is a multi-agent system.
(2) It considers event propagation and interactions with the user of the simulation at runtime.
(3) It has been implemented and thoroughly tested on several scenarios involving thousands of agents situated in open environments.

As a core component of the DIVAs framework, an overview of an earlier version of APR was briefly introduced in [1], and an extension proposed in [3]. However, the core model definition presented in this paper has not been discussed in any prior publication.

In the following section, we define the fundamental concepts at the basis of the APR model.

3 APR CONCEPTS

For the sake of illustration, in the remainder of this paper, we use the example of a social simulation where virtual agents represent humans situated in a virtual city (See Figure 1). It is important to note that the purpose of this work is not to discuss a specific application or an application-level concern (e.g., virtual agent decision making) but rather to discuss a core building block necessary for the construction of large-scale, interactive simulation systems. As such, our discussion is related to the infrastructure layer (See Figure 1).

3.1 Open Environment Architecture

The open environment model which is at the basis of APR has the following characteristics [20, 36]: 1) the underlying structure of the virtual environment's physical space is partitioned into smaller areas called *cells* which carry a portion of the environment state; 2) Each cell is managed by a special-purpose infrastructure agent called *cell controller*. A cell controller does not correspond to a real-world concept but is defined for engineering purposes. A cell controller's role is to "manage" a cell. More precisely, it is responsible for: a) being aware of the virtual agents located in its defined area; b) providing local virtual agents with the latest environment state; c) maintaining a non-conflicting state for its cell; and d) interacting with adjacent cell controllers to exchange information about agents, events and event propagation. Figure 1 shows an environment structured as a collection of adjacent cells each managed by a cell controller. Cell controllers are in turn managed by a higher-level specialized agent called *coordinator*. The coordinator's responsibility is to resolve potential conflicts that require a higher-level of knowledge. During the execution of an application-specific simulation, the underlying partitioned structure of the environment and any action executed at the infrastructure level by the controllers or the coordinator are invisible to the agents and the user of the simulation.

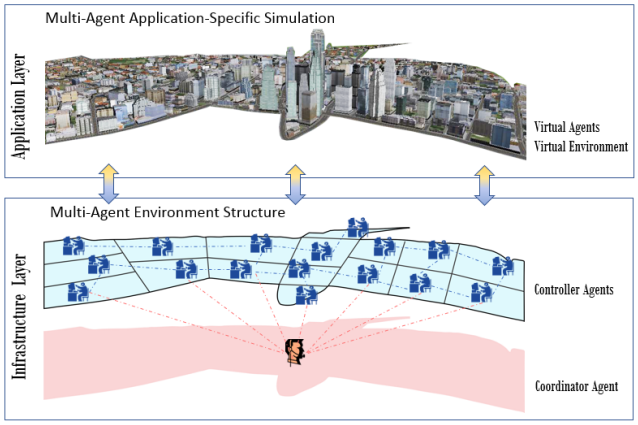

Figure 1: Environment models at the application and infrastructure layers.

3.1.1 Objects and Agent Material Forms. An environment includes *objects* and *agent material forms* (e.g., body). For the purpose of this paper, environment objects are considered to be stationary (e.g., house, tree) whereas agent material forms (e.g., bodies, vehicles) are moving objects. Since objects and agent material forms can be highly detailed with complicated shapes, the processing of these entities may take a prohibitive amount of time. In order to lower the computation time, an object or agent material form is approximated by its *bounding volume* [29], i.e., the shape (e.g., box, cylinder, sphere) with minimum volume surrounding all of the object's or agent material form's points.

In APR, agents and objects are defined by their *states*. In the case of the social simulation example, the agent state includes: the $agentID$, its $type$ (e.g., human, vehicle), $position$, $heading$, $fieldsOfPerception$, $velocity$, $sensorTypes$ (e.g., vision, auditory, olfactory), $status$ (e.g., alive or dead), $scale$, $boundingShape$ (e.g., box, cylinder, sphere) and $stimulus$. The object state includes the $objectID$, $type$, $position$, $scale$, $boundingShape$ and $rotation$.

3.1.2 Agent Perception. During the execution of the simulation, agents are not aware of the underlying partitioned structure of the environment. They assume that wherever they are, they will receive accurate information about their surroundings. Agents are equipped with sensors which filter the environmental data and determine what the agents can precisely perceive [15, 16]. To avoid processing large amount of unperceivable data, controllers are responsible for providing each agent with environment data comprising only the states of cells intersecting the agent's range of perception.

3.1.3 Events and Event Propagation. Events allow the modeling of environmental effects that may not be the direct result of virtual agent actions. These effects can be pivotal to the outcome of a simulation and introduce an enhanced sense of realism. Events are introduced into the simulation either through user intervention (e.g., trigger a bomb, move or resize an object during the simulation), virtual agent action (e.g., scream), or environment object manipulation (e.g., a building collapses). Simulated events propagate over time and space and their effects may cross cell boundaries and span multiple cells. In the APR model, events are defined by their states. In the case of the social simulation example, the event state includes the $eventID$, $type$ (e.g., bomb, fireworks), $origin$, $timeOfOccurrence$, $intensity$, $properties$ (e.g., visible, audible, smellable), $currentPropertyStatus$ (e.g., current visible, audible, smellable radius), $propertyPropagationSpeed$ (e.g., speed of sound propagation), $propertyMaximumRangeOfDetection$ (e.g., maximum visible distance)and $propertyAge$. A detailed discussion on event propagation can be found in [29].

3.1.4 Special Cases. The decentralized environment structure leads to several special cases that require special handling. These include the following.

An agent body spans multiple cells. In this case, the cell which contains the most of the agent's bounding volume is assigned the responsibility to manage the agent. In case of an even span, an implementation mechanism automatically assign an agent to a specific cell.

An agent is located in one cell but its actions will take place in another cell. In this case, the agent's intention to act is processed by the cell in which its physical body is located but its state is updated by the cell containing its final position.

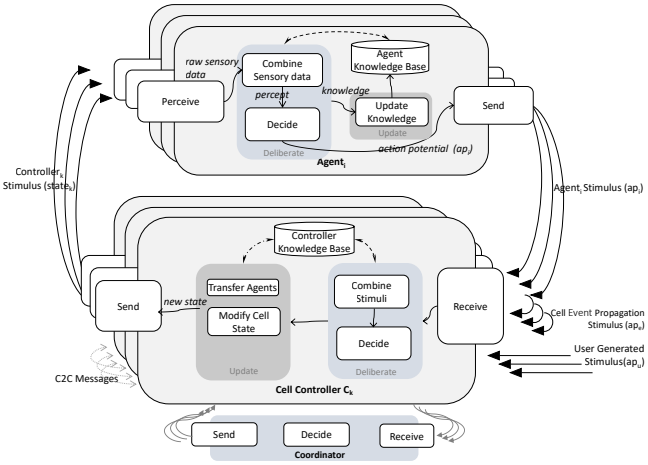

Figure 2: Agent-Environment Interactions in APR

An environment object spans multiple cells. In this case, we chose to store the object details in all intersecting cells. This decision results in storing redundant information but saves on processing.

3.2 APR Concept Definition

Action Potential An action potential is the expected result of an action that is intended to be executed. The intention of acting upon the environment can originate from the agents, environment objects, environment events and/or the user of the simulation. Since there is no guarantee that actions will be allowed by the environment or that their result will happen as anticipated, actions only have the *potential* to occur. The concept of *action potential* is similar to the concept of *influence* in IRM4S. We use the term action potential to minimize confusion: the word *influence* is commonly defined as the "act of producing an effect" [18] whereas in IRM models, it is used to refer to intentions which do not produce effects.

Stimulus A stimulus is the transmission of a signal which prompts an activity in the receiver. Stimuli are classified based on their origin: 1) agent stimuli; 2) external (user-invoked) stimuli; 3) controller-to-controller stimuli; and 4) environment stimuli. In APR, the agents stimulate the environment by transmitting their action potentials; the user stimulates the environment by transmitting their intent to change simulation properties at run-time, and the environment stimulates the environment by sending potential effects of propagating events. In turn, the environment stimulates the agents by sending its latest state.

Action Outcome An action outcome is the result of an action within a certain context.

Action Impact It is the difference between the previous state and the current state of the environment that occurs as a result of some action.

4 APR MODEL OVERVIEW

As mentioned in Section 2, the APR model is built on the concepts defined by IRM and IRM4S and as such, it also follows a two phased approach [29]. In APR, the phases are called *agent phase* and *environment phase* (see Figure 2). To ensure that no agent or cell controller advances in simulated time beyond the others, the APR model is driven by a continuous time tick initiated at the completion of each phase and cycling repeatedly. A simulation cycle corresponds to the completion of both the agent and the environment phases.

The Agent Phase

In this phase, agents *perceive* their surroundings, *deliberate*, *update* their states and *stimulate* their environment (see Figure 2). A virtual agent *perceives* the environment through multiple senses (e.g., vision, hearing, smell) [15]. Each sensor receives the environment state and executes a specialized algorithm to determine what is perceived by the agent in the form of raw sensory data. Following this step, the agent *combines* the raw sensory data received from the multiple sensors and uses its predefined knowledge to convert the combined data into useful knowledge and percepts. Using its current percepts and knowledge, the agent *decides* the next course of actions and determines the action potential which is then transmitted as an *agent stimulus*. This stimulus is synchronously communicated to the cell controller managing the cell in which the agent is situated.

The Environment Phase

In this phase, cell controllers *receive* stimuli, *deliberate*, *communicate*, *update* and *send* their states to their agents. A cell controller *combines* the agent, user triggered, and cell event propagation stimuli and *decides* which action potentials are legal with respect to the rules and constraints of the environment.

Given the level of attention given to agents and agent modeling in the literature, in the following section we focus on the environment component and give a detailed discussion of the environment phase.

5 THE APR MODEL

In this section, we discuss the APR *environment phase*. We start by defining the sets needed for the specification of the environment functions, then proceed with a detailed definition of the environment functions.

5.1 Environment Model

The input and output sets required for the definition of the environment functions are given in Table 1. The environment state is defined in terms of its cells' states. At time $t-1$, $cell_k$'s state $\Sigma_k(t-1)$ includes the parameters given in Table 2.

5.2 Environment Functions

During the environment phase, each cell controller executes three main functions: $Combine-muli$, $Decide$ and $Modify$ (See Algorithm 1). Each of these functions is executed concurrently by the controllers. In the discussion, we use the term agent to refer to the agent's material form which is situated in the environment.

5.2.1 **CombineStimuli.** Through the execution of $Combine-Stimuli$, a controller C_k combines various stimuli affecting its cell with the intent of resolving as many Action Potential (AP) conflicts as possible locally. C_k starts by processing external events affecting $cell_k$ by executing $ProcessEvents$. This function combines the APs of the user triggered event (e.g., trigger a bomb), the propagating

Table 1: Set Definition

Env	: Set of cells
Σ	: Environment State
Σ_k	: $cell_k$'s state
O_k	: Set of objects fully or partially contained in $cell_k$
A_k	: Set of agents located in $cell_k$
$A_{k\to k}$	: Set of agents located in $cell_k$ and whose actions are intended to take place in $cell_k$
$A_{k\to\{x\}}$	: Set of agents located in $cell_k$ and whose actions are intended to take place in adjacent cells x
$A_{\{x\}\to k}$	: Set of agents located in adjacent cells x and whose actions are intended to take place in $cell_k$
A_{xk}	: Set of agents who do not belong to $cell_k$ but whose bounding boxes cross $cell_k$'s boundaries
X_k	: Set of external events triggered by the user of the simulation in $cell_k$
UO_k	: Set of environment object manipulations performed by the user for objects partially or fully located in $cell_k$
Π_k	: Set of propagating events for $cell_k$
$\Pi_{k\to k}$	: Set of propagating events occurring in $cell_k$ and resulting in a propagation in $cell_k$
$\Pi_{k\to\{x\}}$	: Set of propagating events occurring in $cell_k$ and resulting in a propagation in adjacent cells x
$\Pi_{\{x\}\to k}$	: Set of propagating events occurring in adjacent cells x and resulting in a propagation in $cell_k$
E_k	: Environment events resulting from the combination of user triggered and propagating events in $cell_k$

Table 2: Environment State

$a_k(t-1)$	: the state of agents located in $cell_k$
$a_{xk}(t-1)$	: the state of the agents who do not belong to $cell_k$ but whose bounding boxes cross $cell_k$'s boundaries
$o_k(t-1)$	: the state of the objects fully or partially contained in $cell_k$
$\epsilon_k(t-1)$	: the state of the event resulting from the combination of user triggered and propagating events in $cell_k$
$\pi_k(t)$	: the action potentials of events that will propagate within $cell_k$ and/or to $cell_k$ at time t

events and the object manipulations (e.g., object rotation, re-sizing, translation) to determine: a) the combined event state at time t; b) the object states at time t; c) the APs of the propagating events within the cell at time $t+1$, and d) the APs of the propagating events leaving the cell at time $t+1$.

C_k proceeds by executing *PreProcessAgentAPs*. The purpose of this function is for C_k to make use of its local knowledge to determine which agents located in its cell have the potential to realize their action-potentials (APs) in the cell and which will likely execute their APs elsewhere. The information about each agent is stored in *prePocessingDetails* which includes the agent state and the updated APs (after pre-processing).

In *PreProcessAgentAPs*, cell controller C_k addresses straightforward cases and also special cases such as:

Algorithm 1: Environment Phase

input : for each $cell_k$: $\Sigma_k(t-1)$, $\chi_k.stimuli(t, \chi_k.ap)$, $vo_k.stimuli(t, vo_k.ap)$, $a_k.stimuli(t, a_k.ap)$
output: for each $cell_k$: $\Sigma_k(t)$
begin
  for *each* $cell_k \in Env$, *in parallel* **do**
    CombineStimuli $(\Sigma_k(t-1)$, $\chi_k.stimuli(t, \chi_k.ap)$, $vo_k.stimuli(t, vo_k.ap)$, $a_k.stimuli(t, a_k.ap)$**:**
      $\epsilon_k(t)$, $\pi_{k\to k}(t+1)$, $\pi_{k\to\{x\}}(t+1)$, $o_k(t)$, $a_{k\to k}.preProcessingDetails(t)$, $a_{k\to\{x\}}.preProcessingDetails(t)$**)**

    Decide $(\Sigma_k(t-1)$, $\epsilon_k(t)$, $o_k(t)$, $a_{k\to k}.preProcessingDetails(t)$, $a_{k\to\{x\}}.preProcessingDetails(t)$, $a_{\{x\}\to k}.preProcessingDetails(t)$, $\pi_{k\to k}(t+1)$, $\pi_{k\to\{x\}}(t+1)$:
      $a_{k\to k}.actionOutcome(t)$, $a_{\{x\}\to k}.actionOutcome(t)$, $\epsilon_k(t)$, $o_k(t)$, $\pi_{k\to k}(t+1)$, $\pi_{k\to\{x\}}(t+1)$**)**

    Modify($\Sigma_k(t-1)$, $a_{k\to k}.actionOutcome(t)$, $a_{\{x\}\to k}.actionOutcome(t)$, $\epsilon_k(t)$, $o_k(t)$, $\pi_{k\to k}(t+1)$, $\pi_{k\to\{x\}}(t+1)$:
      $\Sigma_k(t)$**)**

Case 1: An agent AP is supposed to take place in another cell but an object in $cell_k$ prevents the agent from moving to that cell. In the scenario shown in Figure 3, *PreProcessAgentAP* computes the estimated updated AP for A and stores it in $A^{1\to 1}_{1\to 2}.preProcessingDetails$.

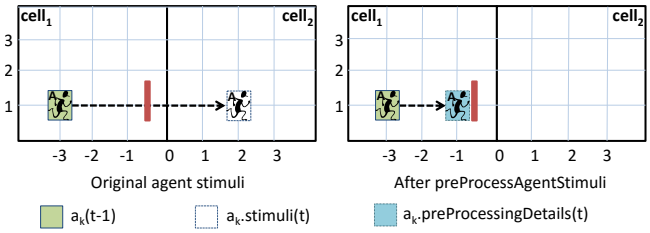

Figure 3: *PreProcessAgentAPs* **- Case 1**

Case 2: Agents have conflicting APs within the cell. For example, in Figure 4 agent A intends to move to position $(1, 3)$ in $cell_2$ whereas a faster agent B decided to move to position $(-0.25, 3)$. The controller determines that the potential positions of agents A and B are respectively $(-1, 3)$ and $(-0.5, 2)$ and stores the updated information in $A^{1\to 1}_{1\to 2}.preProcessingDetails$ and $B^{1\to 1}_{1\to 1}.preProcessingDetails$. Controller C_k proceeds by sending the pre-processing information of the agents that may leave $cell_k$ to all the controllers managing the cells that may be crossed.

Finally, C_k receives information about the agents which might move to $cell_k$.

5.2.2 **Decide.** Through the execution of Algorithm 2, C_k determines which APs can be combined locally and which need to be

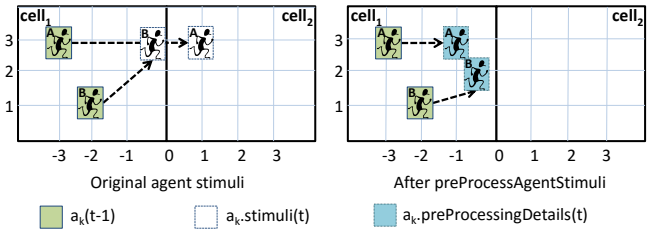

Figure 4: *PreProcessAgentAPs* **- Case 2**

passed on to the coordinator (due to limited knowledge) for further processing.

Algorithm 2: Decide

input : $\Sigma_k(t-1)$, $\epsilon_k(t)$, $o_k(t)$, $a_{k\to k}.preProcessingDetails(t)$, $a_{k\to\{x\}}.preProcessingDetails(t)$, $a_{\{x\}\to k}.preProcessingDetails(t)$, $\pi_{k\to k}(t+1)$, $\pi_{k\to\{x\}}(t+1)$

output : $a_{k\to k}.actionOutcome(t)$, $a_{\{x\}\to k}.actionOutcome(t)$, $\epsilon_k(t)$, $o_k(t)$, $\pi_{k\to k}(t+1)$, $\pi_{k\to\{x\}}(t+1)$

begin

ProcessAgentAPs $(\Sigma_k(t-1), a_{k\to k}.preProcessingDetails(t), a_{k\to\{x\}}.preProcessingDetails(t), a_{\{x\}\to k}.preProcessingDetails(t), \epsilon_k(t), o_k(t), \pi_{k\to k}(t+1), \pi_{k\to\{x\}}(t+1)$**:**
$\underset{(1)}{\sqsubset} a_{k\to k}.actionOutcome(t), a_{k\to k}.unresolvedAPs(t), a_{k\to\{x\}}.unresolvedAPs(t), a_{\{x\}\to k}.unresolvedAPs(t), \pi_{k\to k}(t+1), \pi_{k\to\{x\}}(t+1), \epsilon_k(t), o_k(t)$**)**

Send $(coordinator, a_{k\to k}.unresolvedAPs(t), a_{k\to\{x\}}.unresolvedAPs, a_{\{x\}\to k}.unresolvedAPs(t))$

Wait

DecideCoordinator
$(a_{k\to k}.unresolvedAPs(t), a_{k\to\{x\}}.unresolvedAPs(t), a_{\{x\}\to k}.unresolvedAPs(t)$:
$\underset{(2)}{\sqsubset} a_{k\to k}.actionOutcome(t), a_{\{x\}\to k}.actionOutcome(t)$**)**

Receive $(coordinator, (\underset{(2)}{\sqsubset} a_{k\to k}.actionOutcome(t), a_{\{x\}\to k}.actionOutcome(t)))$

$a_{k\to k}.actionOutcome(t) \leftarrow \bigcup(\underset{(1)}{\sqsubset} a_{k\to k}.actionOutcome(t), \underset{(2)}{\sqsubset} a_{k\to k}.actionOutcome(t))$

C_k starts by executing *ProcessAgentAPs*. This function uses the *preProcessingDetails* of all agents entering, remaining in or intending to leave $cell_k$. Through the execution of *ProcessAgentAPs*, C_k performs several tasks:

1. It determines the action outcomes for those agents within its cell who are not in conflict or whose conflict can be fully resolved locally.
2. For all agents that intend to leave $cell_k$, C_k makes provision for the worst case scenario where their APs may not take place elsewhere. The assumption that these agents may remain in $cell_k$ can result in situations of conflict within the cell. In this case, C_k resolves in advance the potential conflicts and computes estimated APs for the case where these agents remain in the cell.
3. For all agents that intend to traverse (e.g., enter or go through) $cell_k$, C_k assumes the best case scenario, i.e., that the traversal will take place. C_k resolves any potential conflicts within its cell and determines the estimated agent's APs. For example, in Figure 5, agents B and C are in $cell_2$. Given that B may or may not be in conflict with A, C may or may not be in conflict with B. In this case,
$A^{1\to 2}_{1\to 2}.unresolvedAPs(t) = (2, 3)$ and
$B^{2\to 2}_{2\to 2}.unresolvedAPs(t) = (3.5, 0.5)$ and
$C^{2\to 2}_{2\to 2}.unresolvedAPs(t) = (1.5, 1)$

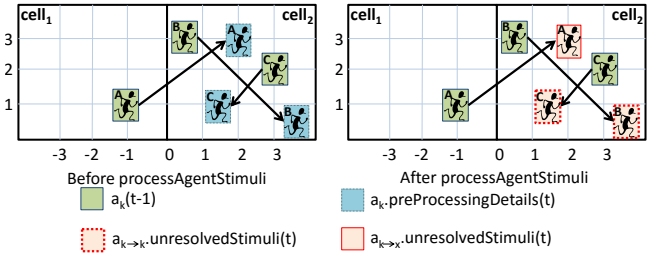

Figure 5: *ProcessAgentAPs* **- Case 3**

For all the cases based on assumptions (i.e., agents leaving or entering), controller C_k requests feedback from the coordinator which has a broader knowledge of the state of the environment.

5.2.3 Decide-coordinator. In this phase, the coordinator receives the list of unresolved APs from the cell controllers. It deliberates and returns the action outcomes to the appropriate cell controllers. The deliberation involves determining the unresolved AP of highest priority, i.e., the one which has the highest impact on other APs. Once this AP is identified and resolved, the coordinator performs an impact analysis to determine which unresolved APs are impacted by the resolution. It proceeds by resolving the next unresolved AP of highest priority. This process is followed until all APs are resolved. The AP prioritization process is context-dependent and needs to be defined by domain experts for specific simulations.

Finally, the coordinator returns the action outcomes to the controllers. At this stage, each cell controller knows which agents will be in its cell at time t.

5.2.4 Modify. Each cell controller executes the $Modify$ function to change the states of its cell. A controller C_k executes the following steps:
1. It requests that the state of agents incoming to $cell_k$ at time t to be sent.
2. It proceeds by computing the states of its agents at time t by applying their action outcome to their states at time $t-1$.
3. Following this step, it determines which agent bounding volumes

cross multiple cells. For these special cases, it exchanges those agent' states with adjacent controllers.
4. It exchanges with other cell controllers the set of event APs propagating to their cells boundaries at time $t+1$ ($\pi_{\{k\}\to x}(t+1)$). Then, it combines the received event APs with the local event APs propagating within its boundaries at time $t+1$ ($\pi_{k\to k}(t+1)$) to determine the set of event APs within its cell at time $t+1$ ($\pi_k(t+1)$).
5. For each agent a_i within its cell, C_k computes the set of cells that are perceivable by a_i. Then, it subscribes a_i to receive the updated states of those perceivable cells. This information is used later to pass onto each agent the state of cells in its subscription list. This drastically reduces the amount of environmental information passed onto an agent for perception.

To compute the agent's cell subscription list, it is necessary to implement a method for each agent that calculates the area of the environment it can perceive. For instance, in the example illustrated in Figure 6, the region that is visible by agent A (also know as its vision cone) is defined by its: 1) field of view; 2) maximum visible distance; 3) heading; and 4) position. This information is used to define the angular extents on each axis within which the agent can perceive information about the environment. The visible region is used by the cell controller to determine which cells an agent is capable of perceiving. In this example, the visible region of agent A intersects with $cell_1$, $cell_2$, $cell_4$, and $cell_5$. Hence, c_4 subscribes agent A to receive the states of $cell_1$, $cell_2$, $cell_4$, and $cell_5$ at the end of the environment phase.

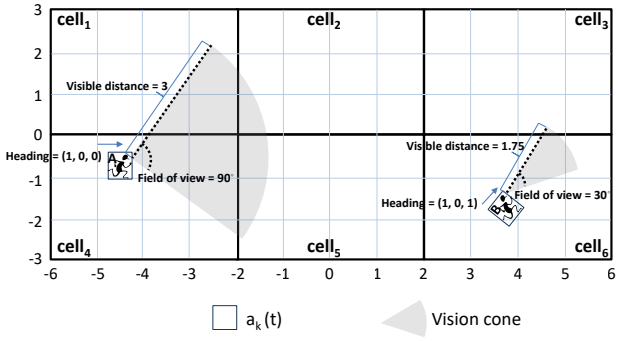

Figure 6: Agent subscription

6. Finally, controller C_k updates the state of its cell to the new state at time t.

$Modify$ is followed by sending the updated cell states to agents according to their cells' subscriptions. The subscribed cells' states are combined into a single environment state that is sent onto the subscribed agent to compute its perception. This mechanism hides the complexity of the partitioned environment from the agent who considers the environment to be a single and unified area.

6 EXPERIMENTAL RESULTS

The APR model has been fully implemented and tested in various simulation systems [2, 15, 32]. The implementation of this core model allowed the execution of thousands of perception-based virtual agents evolving an open, spatial environment on a single PC. To the best of our knowledge, no existing MABS provides this capability.
In this section, we discuss experiments conducted on environment structures consisting of 1, 2, 4, 8, 16 ,32, 64 and 128 cells.

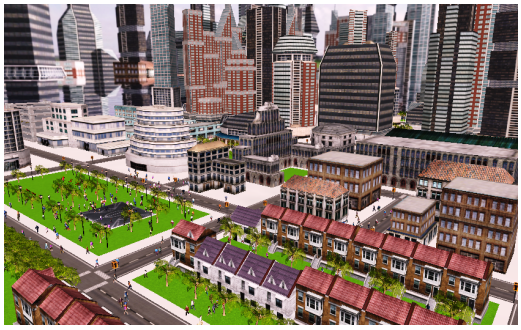

Figure 7: 3D visualization of the agents in the virtual city

6.1 Experiment Setting

The experiments discussed in this section were run on a social simulation system developed using the DIVAs framework. In the simulation, virtual agents representing humans are situated in a virtual city consisting of 850 environment objects of various types (e.g., houses, buildings, trees, traffic lights, benches, etc. (see Figure 7). Virtual agents perceive their surroundings through vision, auditory and olfactory sensors. They execute complex path-finding algorithms to reach various destinations within defined constraints. Initially, agents are scattered in various areas. In this section we limit our discussion to the evaluation of the environment phase.

Using the simulator, we ran experiments and evaluated the APR model with respect to: the 1) *time elapsed for the environment phase*. This time corresponds to the duration needed to complete the environment phase in each simulation cycle, and is measured in milliseconds; and 2) *CPU utilization*, i.e., the percentage of CPU used by the simulation. We ran this experiment 10 times and with 500, 1000, 1500, 2000, 2500, 3000, and 3500 virtual agents using environment structures of 1, 2, 4, 8, 16 ,32, 64 and 128 equally sized cells.

It is important to note that the DIVAs simulation microkernel sets the simulation cycle time to 150 milliseconds. This represents the time needed by the visualizer to display the simulation without delay. In case the simulation cycle (environment and agent phase) finishes its execution in less than 150 milliseconds, the simulation cycle time is delayed until it reaches the 150 milliseconds time limit. This is necessary to ensure a consistent visualization of the simulation. The social simulation system was executed on a multicore PC (Intel Core i7 980X CPU (3.33GHz), 12.00 GB, 64-bit Windows 7) and is implemented in Java (version 1.7.0, 64-bit). In the simulation, controllers are run as threads. Therefore, the computer can efficiently run a certain number of threads concurrently before the context switching overhead leads to a performance degradation. In this experiment, controllers ran on a thread execution pool and the total number of worker threads was set to 12 which corresponds to the number of logical cores available on the computer.

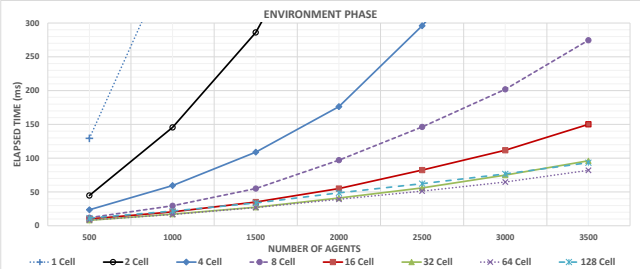

Figure 8: Average env. phase time with 500, 1000, 1500, 2000, 2500, 3000, 3500 **agents and using** 1, 2, 4, 8, 16 ,32, 64, 128 **cells**

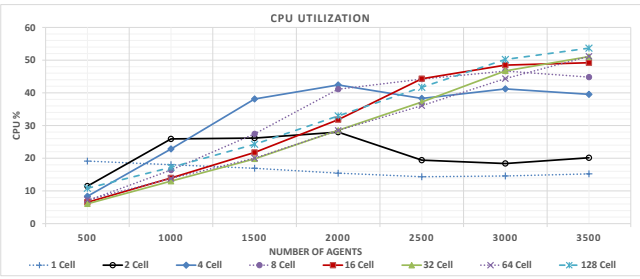

Figure 9: Average CPU utilization with 500, 1000, 1500, 2000, 2500, 3000, 3500 **agents and using** 1, 2, 4, 8, 16 ,32, 64, 128 **cells**

6.2 Results & Evaluation

6.2.1 Elapsed time for the environment phase. Figure 8 shows that, irrespective of the number of simulated agents, the elapsed time for the environment phase decreases asymptotically as the number of cells increases until reaching the size of 32 cells. Then, the elapsed time continues to slightly decrease until reaching 64 cells. The improvement in the elapsed time is accomplished by: 1) utilizing additional number of threads to run the increased number of controllers; and 2) the reduced cell size that minimizes the number of agent APs combined within each cell. After this point, the elapsed time starts to slightly increase again in case of 128 cells. This phenomenon is explained by the additional costs incurred by the very large number of cells in the 128 cell environment structure. These costs include: 1) the thread context switching overhead resulting from the increased number of controllers; 2) the additional processing required to process a larger number of agent APs leaving their cells (due to their smaller cell sizes); and 3) the additional cost required by controllers to provide their agents with environmental data needed to compute their perception. Finally, the results show that for all simulated scenarios, the 64 cell environment structure requires the least time to complete the environment phase and therefore represents the optimal environment structure.

6.2.2 CPU utilization. Figure 9 shows the CPU utilization by the simulation. The results show that the 1, 2, and 4 cell structures have the highest CPU utilization in the case of low number of agents. For instance, in case of simulating 1000 agents, the CPU utilizations for the 2 and 4 cell structures are 26%, and 23% respectively. On the other hand, the CPU utilizations for the 64 and 128 cell structures are 14% and 17% respectively. However, as the number of agents increases,

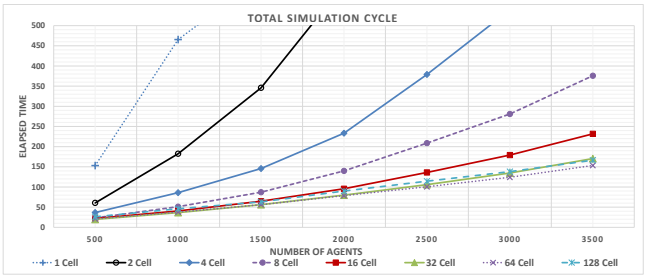

Figure 10: Total cycle time with 500, 1000, 1500, 2000, 2500, 3000, 3500 **agents and using** 1, 2, 4, 8, 16 ,32, 64, 128 **cells**

the CPU utilization for the 1, 2, 4 and 8 cell structures decreases and reaches a minimum for 3500 agents. As discussed in Section 6.1, the overall cycle time is set to a maximum of 150 milliseconds. In the case of 500 agents, all environment structures complete their cycles before the allotted 150 milliseconds (see Figure 10). However, in the case of 3000 agents, the cycle time for the 1, 2, and 4 cell structures far exceeds the 150 milliseconds limit while for the 32, 64, and 128 cell structures it is under 150 milliseconds (see Figure 10). Therefore, within the same period of time, the simulations using the 32, 64, and 128 cell structures execute more cycles and consequently utilize more CPU than the 1, 2, and 4 cell environments.

Finally, the results in Figure 9 show that in the case of 3500 agents, the 32, 64, and 128 cell environments have the highest CPU utilizations. Nevertheless, the 64 cell environment has has the lowest total simulation cycle time (see Figure 10). This implies that the 64 cell structure better utilizes the available computational resources and may be the best configuration.

7 CONCLUSION

In this paper, we presented APR, an agent-environment interaction model for MABS with open environments. APR is based on the general IRM principle and improves existing IRM-based models by considering the environment as a multi-agent system. The decentralized structure and control of the environment and the interaction model allow the execution of thousands of agents on a single computer. In addition, APR considers event propagation and user triggered events during the simulation.

The APR model has been implemented as a generic module in the DIVAs framework. The module provides the full interaction mechanisms and workflow as well as abstract methods that can be instantiated for domain-specific action combinations. Although the environment phase involves complex processing, the virtual agents are unaware of the partitioned structure of the environment. The experimental results show that the agent and environment phases are completed in far less than the required 150 milliseconds simulation cycle for simulations involving 2500 and 3500 virtual agents situated in a 64 cell environment structure.

The APR main limitation is the pre-defined static partition of the environment. In cases where agents are concentrated in a few cells, the cycle is delayed and the performance decreases. Moreover, the definition of one coordinator may result in a bottleneck when agents continuously cross cells. In [3], we proposed an adaptive environment model, but much work is needed to assess this model.

REFERENCES

[1] M. Al-Zinati, F. Araujo, D. Kuiper, J. Valente, and R. Z. Wenkstern. 2013. 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)*. Delft, Netherlands, 105–114.

[2] Mohammad Al-Zinati and Rym Wenkstern. 2015. MATISSE 2.0: A Large-Scale Multi-Agent Simulation System for Agent-based ITS. In *Proceedings of the 2015 IEEE/WICIACM International Conference on Intelligent Agent Technology (IAT' 15)*. Singapore, Singapore, 328–335.

[3] Mohammad Al-Zinati and Rym Zalila-Wenkstern. 2014. A self-organizing model for decentralized virtual environments in agent-based simulation systems. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*. Paris, France, 1583–1584.

[4] Ana LC Bazzan. 2005. A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multi-Agent Systems* 10, 1 (2005), 131–164.

[5] Kurt Dresner and Peter Stone. 2008. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research* 31 (2008), 591–656.

[6] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. 2013. Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): methodology and large-scale application on downtown Toronto. *IEEE Transactions on Intelligent Transportation Systems* 14, 3 (2013), 1140–1150.

[7] Jacques Ferber. 1999. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley.

[8] Jacques Ferber and Jean-Pierre Müller. 1996. Influences and Reaction : a Model of Situated Multi-agent Systems. In *Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96)*. The AAAI Press, Soraku-gun, Kyoto, Japan, 72–79.

[9] Stéphane Galland, Nicolas Gaud, Jonathan Demange, and Abderrafiaa Koukam. 2009. Environment model for multiagent-based simulation of 3D urban systems. In *Proceedings the 7th European Workshop on Multiagent Systems (EUMAS09)*. Ayia Napa, Cyprus.

[10] Stephane Galland, Luk Knapen, Nicolas Gaud, Davy Janssens, Olivier Lamotte, Abderrafiaa Koukam, Geert Wets, et al. 2014. Multi-agent simulation of individual mobility behavior in carpooling. *Transportation Research Part C: Emerging Technologies* 45 (2014), 83–98.

[11] Veronika Gaube and Alexander Remesch. 2013. Impact of urban planning on household's residential decisions: An agent-based simulation model for Vienna. *Environmental Modelling & Software* 45 (2013), 92–103.

[12] Alexander Helleboogh, Giuseppe Vizzari, Adelinde Uhrmacher, and Fabien Michel. 2007. Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-Agent Systems* 14, 1 (2007), 87–116.

[13] Qingxu Huang, Dawn C Parker, Tatiana Filatova, and Shipeng Sun. 2014. A review of urban residential choice models using agent-based modeling. *Environment and Planning B: Planning and Design* 41, 4 (2014), 661–689.

[14] Franziska Klügl, Rainer Herrler, and Manuel Fehler. 2006. SeSAm: implementation of agent-based simulation using visual programming. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS2006)*. Hakodate, Japan, 1439–1440.

[15] Dane M Kuiper and Rym Z Wenkstern. 2015. Agent vision in multi-agent based simulation systems. *Autonomous Agents and Multi-Agent Systems* 29, 2 (2015), 161–191.

[16] Dane M Kuiper and Rym Z Wenkstern. 2015. Agent vision in multi-agent based simulation systems. *Autonomous Agents and Multi-Agent Systems* 29, 2 (2015), 161–191.

[17] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Catalin Balan. 2005. MASON: A Multiagent Simulation Environment. *Simulation* 81, 7 (2005), 517–527.

[18] Merriam-Webster. 2018. Merriam-Webster Dictionary. (2018).

[19] Fabien Michel. 2007. The IRM4S Model: The Influence/Reaction Principle for Multi-Agent Based Simulation. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (AAMAS2007)*. Honolulu, Hawaii, 133.

[20] Rym Mili, Gary Leask, Uttama Shakya, and Renee Steiner. 2004. Architectural Design of the DIVAs Environment. In *Proceedings of the First international conference on Environments for Multi-Agent Systems (E4MAS04)*. New York, NY.

[21] Gildas Morvan, Alexandre Veremme, and Daniel Dupont. 2010. IRM4MLS: The Influence Reaction Model for Multi-Level Simulation. In *Proceedings of International Workshop on Multi-Agent Systems and Agent-Based Simulation (MABS 2010)*. Toronto, Canada, 16–27.

[22] Michael J. North, Nicholson T. Collier, Jonathan Ozik, Eric R. Tatara, Charles M. Macal, Mark J. Bragen, and Pam Sydelko. 2013. Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling* 1 (2013), 3.

[23] Nuria Pelechano, Jan M. Allbeck, and Norman I. Badler. 2007. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, (SCA 2007)*. San Diego, California, USA, 99–108.

[24] Alexander Repenning, Andri Ioannidou, and John Zola. 2000. AgentSheets: End-User Programmable Simulations. *Journal of Artificial Societies and Social Simulation* 3, 3 (2000).

[25] Andreas Rienow and Dirk Stenger. 2014. Geosimulation of urban growth and demographic decline in the Ruhr: a case study for 2025 using the artificial intelligence of cells and agents. *Journal of Geographical Systems* 16, 3 (2014), 311–342.

[26] Stuart J. Russell and Peter Norvig. 2010. *Artificial Intelligence - A Modern Approach* (Third ed.). Prentice-Hall, Egnlewood Cliffs.

[27] Emilio Serrano, Pablo Moncada, Mercedes Garijo, and Carlos A Iglesias. 2014. Evaluating social choice techniques into intelligent environments by agent based social simulation. *Information Sciences* 286 (2014), 102–124.

[28] Flaminio Squazzoni, Wander Jager, and Bruce Edmonds. 2014. Social simulation in the social sciences: A brief overview. *Social Science Computer Review* 32, 3 (2014), 279–294.

[29] Travis Steel, Dane Kuiper, and Rym Zalila-Wenkstern. 2010. Context-Aware Virtual Agents in Open Environments. In *Proceedings of the Sixth International Conference on Autonomic and Autonomous Systems (ICAS 2010)*. IEEE, Cancun, Mexico, 90 – 96.

[30] P. Taillandier, D.A. Vo, E. Amouroux, and A. Drogoul. 2012. GAMA: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. *Principles and Practice of Multi-Agent Systems* 7057 (2012), 242–258.

[31] Warren Thorngate. 2014. Minding Norms: Mechanisms and Dynamics of Social Order in Agent Societies (Oxford Series on Cognitive Models and Architectures) *by Rosaria Conte, Giulia Andrighetto and Marco Campennl (eds.)*. *Journal of Artificial Societies and Social Simulation* 17, 3 (2014).

[32] Behnam Torabi, Rym Z Wenkstern, and Mohammad Al-Zinati. 2018. An Agent-Based Micro-Simulator for ITS. In *Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems (IEEE ITSC 2018)*. Maui, Hawaii, USA, 2556–2561.

[33] Neal Wagner and Vikas Agrawal. 2014. An agent-based simulation system for concert venue crowd evacuation modeling in the presence of a fire disaster. *Expert Systems with Applications* 41, 6 (2014), 2807–2815.

[34] Danny Weyns, H. Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber. 2005. Environments for Multiagent Systems: State-of-the-Art and Research Challenges. In *Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, New York, NY, USA, July 19, 2004*, Danny Weyns, H. Van Dyke Parunak, and Fabien Michel (Eds.). LNAI, Vol. 3374. Springer.

[35] U. Wilensky. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. (1999). Accessed March 2017.

[36] Rym Zalila-Mili, Renee Steiner, and E. Oladimeji. 2006. DIVAs: Illustrating an Abstract Architecture for Agent-Environment Simulation Systems. *Multiagent and Grid Systems. Special issue on Agent-Oriented Software Development Methodologies* 2, 4 (January 2006), 505–525.

[37] Honghui Zhang, Xiaobin Jin, Liping Wang, Yinkang Zhou, and Bangrong Shu. 2015. Multi-agent based modeling of spatiotemporal dynamical urban growth in developing countries: simulating future scenarios of Lianyungang city, China. *Stochastic environmental research and risk assessment* 29, 1 (2015), 63–78.