Self-Adaptive Agents for Debugging Multi-Agent Simulations

Franziska Klügl
Modeling and Simulation Research Center
Örebro University
Örebro, Sweden
Email: franziska.klugl@oru.se

Carole Bernon
Institut de Recherche en Informatique de Toulouse
Paul Sabatier University (Toulouse III)
Toulouse, France
Email: carole.bernon@irit.fr

***Abstract*—In this contribution, we propose an adaptation-driven methodology for the technical design and implementation of multi-agent simulations that is inspired by the concept of “living design”. The simulated agents are capable of evaluating their behavior and self-adapt for improving the overall model. For this aim, the modeler describes critical, non valid situations in the life of an agent, or the complete agent system, and explicitly specifies repair knowledge for these situations.**

***Keywords*-multi-agent systems, simulation, methodology.**

I. Introduction

The development and usage of appropriate methodologies for developing multi-agent simulations has become a focus of scientific attention as it became obvious that traditional approaches are not sufficient to handle the complexity of design, implementation, testing or validation of multi-agent simulations as they do not account for generative nature of these form of simulation models. Our paper contributes to this research on techniques supporting multi-agent simulation development. We propose a procedure where agents themselves improves the quality of the technical design and implementation of an agent-based simulation model. The part of the process that we address is often also referred to as debugging in a quite wide sense.

The development of such a high quality model is challenging: first of all, agent-based models are *generative* [1]. That means the overall dynamics is generated from micro-level agent behaviors and interactions. There is no explicit relation between what is defined in the model and what is the produced overall behavior, but this relation is just established by simulation. Thus, there is a profound uncertainty on assumptions mainly about agent-level structures and behavior model but also on the constituents of the environmental model. The appropriateness of assumptions can only be tested and determined using a fully calibrated, runnable simulation. On a more technical level, other challenges have to be taken up such as the mere size of the simulation in terms of number of agents or their level of heterogeneity. The brittleness of a model coming from non-linear interaction outcomes and from the usage of so-called “knife-edge thresholds” [2] in agent models may also aggravate the general problem. From a practical point of view, the size of a model with many parallel acting and interacting agents, with different categories of agents in different local contexts causes problems for implementation: it is problematic to oversee the dynamics, to identify problems and locate bugs in the implemented model.

In this contribution, we propose an approach for design and implementation of a multi-agent simulation model that is inspired by the “Living Design” concept [3]: in addition to the behavior that shall be actually simulated, the agents in the simulation are equipped with an additional meta-level module for monitoring their performance, identifying problems and repairing them. The necessary knowledge is provided by the modeler either explicitly, for example, in terms of constraints, or implicitly by interacting with the simulated agent. Our suggestion is appropriate for those agent-based simulations that possess an original system and are developed with a clear question; in terms of [4] a case study rather than a model abstraction.

After a short introduction to the background of this work, the concept of self-debugging agents is introduced and discussed. After comparing our approach to related works, a conclusion and some outlook to future work are given.

II. Agent Adaptation instead of Manual Repair

Consider the following situation: a modeler has to develop a multi-agent evacuation simulation where pedestrian agents have to exit at best a train or a building. The initial test runs produce a number of situations where agents are blocking each others at exits or bottlenecks, agents are getting stuck with obstacles. Clearly, the behavior model of the agents is not valid, that means it is neither optimally calibrated nor complete or correct. In a painful trial and error procedure, the modeler now might adjust minimal acceptable distances, add behavior elements for explicitly giving way to others or for pushing past the other agents. Our idea is to avoid this iterative manual improvement by the modeler by giving the pedestrian agents the capability to recognize such critical situations and modify their behavior and the elements of the simulated environment to avoid them in the future.

In the Agent-Oriented Software Engineering (AOSE) area, *Adelfe* [5], related to the concept of Adaptive Multi-Agent Systems (AMAS) [6], is one of the rare methodologies that explicitly suggests using self-designing agents to build a multi-agent system. Based on meta-rules describing the prerequisites for "useful" interacting agents, the AMAS theory identifies 7 categories of "Non-Cooperative Situations" (NCS) in that these prerequisites are not fulfilled. NCS have to be anticipated, avoided or eventually repaired by every agent. The processing of NCS is separated from the actually intended behavior of an agent. It possesses a "nominal" behavior for performing the usual behavior and an additional meta-level "adaptive" behavior to discover and eliminate the NCS it encounters. Thus, the agent learns to adapt to its environment when the nominal behavior is not sufficient.

Transferred to the problem of technical design and implementation in multi-agent simulation, NCS correspond to situations that do not occur in the original system. We call these Non-Valid Situations (NVS) for denoting the particular relation to producing valid simulations. They are not restricted to blocking situations as sketched in the beginning, but may also characterize discrepancies between simulated and real data on a macro-level, the missing production of organizational structures, etc. Situations that might be judged as non-cooperative for an artificial multiagent system may actually occur in the real world and thus must occur in the corresponding context in the simulation. Looking through a developers eye, there is no basic conceptual difference between NCS and NVS; Nevertheless, we think that "NVS" is more appropriate in the simulation context focusing on the model development level.

III. Adaptive Agents for Model Debugging

A. General Concept

The general concept of a self-debugging agent is based on the idea that an adaptation component can be added to the standard agent model for identifying and repairing NVS. The respective phases (illustrated in fig. 1) are the following:

1) The starting point is a runnable initial prototype for the full agent-based simulation model.
2) The modeler develops the adaptive module for identifying NVS and how the simulated agents deal with these critical situations.
3) The original nominal agent behavior is connected to the adaptive behavior and both are running concurrently while the simulation is executed and repeated.
4) The nominal model and the adaptive behavior are separated again when all identified NVS are solved. Parts of the adaptive elements may remain in the agent program.
5) The validity of the final model is tested thoroughly again. If previously hidden or new NVS are discovered, an adaptive behavior has to be added and the adaptation starts again, otherwise the model is ready for final documentation and deployment.

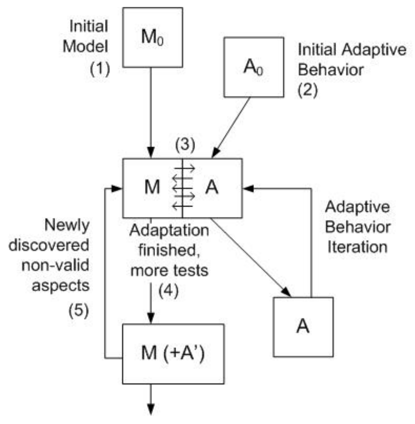
Initial Model (1)
M0
A0
Initial Adaptive Behavior (2)
(3)
M
A
Adaptive Behavior Iteration
Newly discovered non-valid aspects (5)
Adaptation finished, more tests (4)
A
M (+A')

Figure 1: Adaptation-driven design process starting from an initial model M_0 and a corresponding adaptive repair component A_0. After iteratively testing and adapting, an improved model is produced that may contain some built-in adaptive elements.

B. Agent Architecture

The particular agent architecture for self-debugging agents consists of two parts: the original nominal behavior and the adaptation module. All agents, a subpopulation or even just one agent, may be equipped with an adaptation component. In principle, also entities without agent properties – such as resources, obstacles or the modeled environment – may be equipped with an adaptation component that modifies their parameters. This enables handling adaptations that are not isolated to an entity but need to affect more than one entity in a coordinated fashion. The overall architecture is sketched in Figure 2.

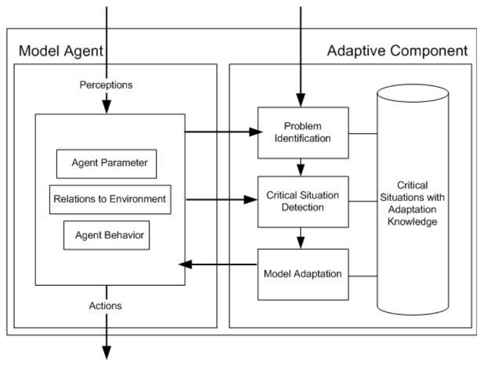

Figure 2: Architecture of a self-debugging agent with both nominal (left side) and adaptive behavior (right side).

The adaptation module has its own perception that may include the agent's status and perceptions, but also additional perceptions from the simulated environment or even connections to external data sources. The actions of the adaptation module may directly effect only the agent to which it is assigned.

C. Self-Debugging Phases

In the following, the three major (iterated) tasks that a self-debugging agent has to execute are discussed. First, the adaptation module has to notice that something in the simulation is going wrong. The identified problem has to be reduced to one or more NVS that are basically capturing assumptions for possible causes for the problems. Instructions for repairing the model in reaction to the NVS are then associated with the NVS. In the third phase, the instructions are executed and the model is adapted. This cycle has to be repeated again and again until no problem is observable any more.

A problem might be the availability of appropriate meta-knowledge to run this cycle. This problem is depending on the particular model that is to be debugged. With an appropriate – model depending – interface between adaptive module and human system expert, human intelligence and implicit knowledge can be used for filling the gaps in explicitly formulated knowledge.

1) Problem Identification: The identification of a problem is directly connected to criteria for determining the validity of a simulation model. As we are addressing multi-agent simulation, these criteria can be found on different levels of observation. If a criteria for model validity does not hold, a problem can be detected. Criteria may be both quantitative or qualitative.

1) Most frequently, values, trends and dynamics of global output variables are used for validation: macro-level descriptors are compared to corresponding values obtainable from the simulation. Invariants and similar conditions involving one or more output values may be formulated as a criterion of model validity that must be satisfied. An adaptation module that should be capable of identifying a problem on the macro-level needs a global perspective.
2) Output values may serve as a basis for comparison also on a meso (group) or micro (individual agent) level. The latter may be important if prominent, unique agents are part of the simulation.
3) Macro-level structures, that shall emerge from the interaction of the whole collective of agents, may be characterized. Their failure to express emergent behavior or the possible inadequateness (shape, time, etc.) of this latter may lead to identification of a problem.
4) Interaction outcomes can be characterized and may form a source for criteria determining whether a simulation is not running smoothly. Agents block each other where they should not, do not interact when they should or interact with the wrong partners.
5) A similar source of criteria for validity are individual behavior paths. Agents decide for other than the rational option or stop when they should not.

Clearly, these criteria are not independent from each other as macro-level outcomes are generated from the micro-level. Nevertheless, it is useful to define them on different levels of abstraction.

2) NVS Identification: Having perceived that there is a problem, a reason for the problem has to be found for being able to modify the agent behavior in a way that the problem is avoided. As in multi-agent simulations all dynamics are generated by the agents behavior and interactions, the reason for a problem can be found solely on the micro-level. Thus, the original seven NVS of AMAS can also be identified in the context of multi-agent simulations:

1) INCOMPREHENSION. An agent receives an information but has no interpretation for this information. For example, an agent bumps into something, is not able to recognize it as an obstacle and cannot avoid it, getting blocked by it.
2) AMBIGUITY. The perception can be interpreted in more than one way. The behavior model is not refined enough with respect to the environmental complexity or the messages sent by other agents. For example, an agent which is roaming in an environment is not able to distinguish an obstacle from a source of energy, it will therefore be unable to know how it has to behave (avoiding the obstacle or recharging its batteries).
3) INCOMPETENCE. The agent possesses no rule for dealing with a particular internal status. That means the state variables of an agent possess values that are beyond the values that can be dealt with by its reasoning. For example, an agent which wants to recharge and finds a source of energy, finds a source which has a failure, it is therefore incompetent to recharge.
4) UNPRODUCTIVENESS. Given the current information situation, the reasoning is not capable of producing an output. For example, an agent receives a message that is not meant for it, it is not able to do something by using this message.
5) CONCURRENCE. When an agent attempts to execute the selected action, another agent executes the same action resulting in redundancy when this action has just to be done once. If there are these inefficiencies in the original system, they have to be reproduced in the simulation model for developing a realistic reproduction of the original system. Nevertheless there are cases when this redundancy may point to a problem of inappropriate action selection. For example, two agents roaming in an environment want to move in the same place at the same simulated time whereas there is a clear coordination in the real world.
6) CONFLICT. Two or more agents possess a conflict over resources. That means, if one agent executes its actions, the others cannot reach their goals. As with CONCURRENCE, this situation may actually occur,

but real-world agents may have invented mechanisms to cope or solve the conflict. For example, when two simulated pedestrians want to concurrently pass through a narrow door, they are not blocked for a longer time, but one of the agents will proceed while the more polite one will give way. Thus, the task of the adaptive component may not be just to modify the simulated agent for avoiding the critical situation from the beginning, but to add appropriate, timely rules about how agents can decide in the case of conflict.

7) USELESSNESS. The action of an agent has no effect. In the simulation case, it might happen that the action has no impact neither on the environment nor on the internal status of the agent. For example, a car agent that travels between two locations in a town environment is blocked because of a traffic jam at a unrealistic position. Being obliged to stay still is useless for the agents since this situation does prevent it (and may be others that are blocked behind it) from reaching its goal.

3) Adaptation: For every identified problem there might be a number of critical situation descriptions for explaining the identified problem. With a critical situation serving as explanation, there should be an assigned adaptation or "repair" plan – a sequence or skeletal plan – stating what can and shall be modified for tackling the critical situation. The selection of the modifying actions is not necessarily deterministic. Optimality and convergence can only be determined for a particular model.

One can identify the following action repertoire usable in such a plan:

1) *Move*(<*destination*>) modifies the agents embedding into its environment – move changes its local context by moving the agent to another location. The movement has to be seen figuratively: movement not just in metric space, but also within networks and organization, adapting addressees of information, communication partners, etc.
2) *AdaptParameter*(*Parameter,* [<*direction*>]) modifies an individual parameter of the modeled agents. For example, *AdaptParameter*(*DesiredSpeed,* $[Increase]$) increases the parameter that determines the desired speed of an agent. This might lead to more heterogeneity in the agent population as every agent modifies its parameters individually. However, if the parameter of all agents have to be adapted, this individual-based adjustment may not be efficient.
3) *InsertRule*(<*condition*>$\rightarrow$<*action*>) means structurally modifying the agent behavior by adding rules For discovering the appropriate model modifications, agents may use learning techniques.
4) *Generate*(<*agentType*> [*at* <*time*>*]*) generates new agents to the current situation at a given time. This time may be "now" or any point during the simulation or the start situation. For reproducibility, this generation must be integrated into the model description – therefore changing the model input adding a new external event from a local perspective. Since the simulated environment is treated as an explicit entity in the simulation model, it may also be equipped with an adaptive component changing the input behavior from a global perspective.
5) *Delegate*(<*situation*>$\rightarrow$<*list of agents*>) notifies a set of other agents about the occurrence of a specific critical situation. Basically, the agent has detected that there is a particular critical situation that cannot be solved by itself but by other agents involved in the situation.
6) *Delegate*(*ResponsibleModeler*). The adaptive component of the model entity has detected that there is a specific situation, but has no information or plan for coping with the situation. Thus, it notifies the modeler, or the domain expert, about the occurrence of the situation and asks for modifications by the human involved. Such a "repair" plan element has similarity to some break points in traditional programming languages.

These actions for modifying the simulated agents nominal behavior can be configured and used to set up a repair plan. This has to be done for each identified NVS. The identified NVS gives the particular addressee (particular parameter, network position, etc.) of the modifications. However setting up such a repair plan may not be simple. As indicated in the last possible action, a solution might be delegating the problem solution to an involved human expert.

After modification, whether the modified agents are now able to produce the intended overall behavior has to be tested. Depending on the particular simulation endeavor, the simulation run has to be completely restarted or can be resumed from the simulated time where it was stopped due to the identification of the problem. When blocking situations are involved, resuming might be sufficient to see whether the USELESSNESS or CONFLICT situation resolves. On the other side, when population dynamics are involved, restart may not be avoidable.

IV. RELATED WORKS

Due to their main features – distribution, heterogeneity, parallelism, scalability, autonomous behaviors of agents, emergent collective behavior, etc. – multi-agent systems are difficult to test and debug. Agent-oriented methodologies are becoming mature enough to study how phases of testing and validation can be integrated into their process development [7]. However, very few are specifically interested in guiding the development of agent-based simulations and less aim at helping the modeler by providing him with tools to facilitate the implementation of the model, to observe how it behaves and to improve this behavior. In *Ingenias*, for instance, a graphical modeling language helps modelers to design a model and code generation is then enabled toward some simulation platforms (Mason, Repast) [8]. In *Adelfe*, a fast prototyping ability is offered to see how the agents behave, however this tool is based on state machines and does not offer a visual aid that would enable detection of wrong behaviors by observing agents moving in a simulated environment [5]. Also in [9] agent self-modification using is suggested for adapting to a dynamic environment.

Debugging in multi-agent systems is often provided by observation and visualization tools, see [10] for a suite of visualization tools devoted to debugging. However, displaying a huge quantity of information to the designer, most of the time about interactions between agents, is not always helpful and automatic correlation or/and analysis of these data (e.g., using graph as in [11]) or automatic detection of wrong behaviors is highly desirable. In Prometheus, this help is brought by a tool, which observes conversations between agents, tests if they conform to the specified interaction protocols, and notifies violations to the developer [12]. This kind of debugging can only validate specifications initially made during the design and cannot discover ways of communicating that could enable a better behavior at the micro or macro-levels. Since agent-based systems can be considered from different perspectives, the approach proposed in [13], for notably testing simulations, is to study problems (mainly deadlocks, bottlenecks and performance problems) that can occur at the agent level before considering errors at the interaction level for eventually testing problems at the system one. To avoid a deterministic approach, tests are made in a stochastic way and repeated several times. Here also, testing at an agent level compares its behavior with the way it was specified and designed; may be the result of testing may lead to notify the designer of problems but nothing is done to enable discovering a behavior better adapted to some constraints or environment.

As discussed in the beginning, the situation for multi-agent simulation is difficult as testing and debugging for implementation bugs is only secondary. Due to their generative nature and often restricted data availability, “guessing” about the agent level behavior and interaction is based on experience and creativity. Explicit handling of information on the original system is central. Most suggested tests for simulations are therefore based on humans evaluating model structure, behavior and outcome [14]. Our suggestion is a realistic intermediate step between full human-done modeling, implementing and debugging and approaches such as [15] that try to transfer the agent modeling problem to a learning problem.

V. Discussion

There are aspects in the proposed approach to debugging in multi-agent simulation that need to be discussed. The most important issues are the availability of explicit meta-knowledge, dependencies between adaptations, the convergence of adaptation and the simplicity and understandability of the outcome.

The approach described so far can only work when criteria for validity can be formulated and tested automatically by the agents, and changes be given on how to react on the perceived deficiencies. The former is especially depending on the particular model. For many multi-agent simulations only statistical data are available on the macro level, population numbers, average turnovers, road link load over a certain time, etc. However, for the described approach, the validity criteria have to be broken down to individual values, behavior paths or the outcomes of interactions. Whether such information is formulate-able is depending on the particular model. However, often a modeler, domain expert or stakeholder can, based on their experience with the original system, identify situations and features that do not “look” realistic although they might not be able to explicitly describe what are the reasons for this invalid observations. When having identified that there is a problem, they can often tell how the situation should “look” like or what should happen. Thus, based on an appropriate user interface, human intelligence should be used when automatic detection and repair is not possible. Actually, this would be a fall back to the previous way of debugging, it is now embedded into systematics.

There is a second critical aspect: one can find application scenarios where model modifications cannot be done for agents independently from modifications in other agents or entities. For example, the metabolism of an agent has a connection to available resources in the environment when intending to produce some sustainable agent population growth. Interdependent modifications in different model elements cannot be formulated based on adaptive components added to individual agents. Higher level entities can be created that do not have a role, or just may have a passive role in the original model. These entities or the explicitly modeled environment can be augmented with the meta-level module for testing and adaptation.

A priori, we cannot guarantee that the proposed approach has any noticeable impact on model quality properties such as simplicity or understandability. These are highly depending on the way the modeler formulated the initial nominal and the adaptive behaviors. However, the distinction between nominal and adaptive behavior and their explicit handling may support understandability, as the modeler is forced to explicitly treat and add meta-level knowledge about the model. We would also expect that the explicitness of such information is also of great relevance for the reusability and maintainability of the model.

VI. Conclusion

We proposed a concept and architecture for self-debugging agents for multi-agent simulation. A modeler can start with a model prototype, augment it with additional explicit knowledge on how the outcome looks like. Then, systematically non-valid situations are identified and repaired. This forms an iterative, yet consequent debugging and modification process towards a sufficiently valid representation of the original system. Our framework relies on the ability of a human modeler to identify and formulate critical situations. Yet, yhe modeler does not need to provide direct solutions about how to adapt the agents' behavior for coping with or avoiding such situations, but just needs to indicate what the agents might have to change for improving future simulation runs. Thus, the process is different from trial and error debugging but supports systematic design. From the beginning, the modeler has to think about potential critical situations. He explicitly classifies what may be adapted and what is sacrosanct. This is not only a deliberate treatment of potential non valid situations but also a clear and concise elaboration of what are the parts of a model that are fixed (e.g., because the data are clear or the underlying theory does not allow adapting these model elements) or which ones can be varied to what extent.

Clearly there are some weaknesses and potential perils that have to be analyzed and tackled in future research: in the current approach, there are only local adaptation actions suggested, initiated by agents with local view on NVS. A solution might be introducing adaptive agents on higher aggregation levels. The most critical issue is whether the modeler actually can provide sufficient meta-level information for specifying the adaptive part of the agent behavior, the NVS including the repair plans. If he just can specify the NVS but has to delegate all repair actions to a manual modification, the overall process is reduced to defining constraints and invariants and automatically testing them, notifying the modeler in cases that they are violated. This is nevertheless a valuable modeling support. Our future research will therefore be directed to the application of the self-modeling agents in a variety of application domains gathering experience about forms of accessible and explainable meta-knowledge that can be used for systematically defining critical situations and repair plans. Also, we will test data mining methods for supporting the elicitation of model meta-knowledge that can be used for guiding the modifications towards better multi-agent models.

References

[1] J. M. Epstein, "Agent-based computational models and generative social science," *Complexity*, vol. 4, no. 5, pp. 41–60, 1999.

[2] L. R. Izquierdo and J. G. Polhill, "Is your model susceptible to floating-point errors?" *Journal of Artificial Societies and Social Simulation*, vol. 9, no. 4, p. 4, 2006.

[3] J.-P. Georgé, G. Picard, M.-P. Gleizes, and P. Glize, "Living design for open computational systems," in *Int. Workshop on Theory And Practice of Open Computational Systems (TAPOCS at WETICE 2003), Linz, June, 2003*. IEEE Computer Society, 2003, pp. 389–394.

[4] R. Boero and F. Squazzoni, "Does empirical embeddedness matter? methodological issues on agent-based models for analytical social science," *Journal of Artificial Societies and Social Simulation*, vol. 8, no. 4, p. 6, 2005.

[5] C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard, "Engineering adaptive multi-agent systems: the Adelfe methodology," in *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, Eds. Idea Group Pub, June 2005, pp. 172–202.

[6] M.-P. Gleizes, V. Camps, and P. Glize, "A theory of emergent computation based on cooperative self-organization for adaptive artificial systems," in *4th European Congress of Systems Science*, L. Ferrer Figueras, Ed. Spanish Society of System Science, 1999.

[7] B. Henderson-Sellers and P. Giorgini, *Agent-Oriented Methodologies*. Idea Group Pub, June 2005.

[8] C. Sansores and J. Pavón, "Agent-Based Modeling of Social Complex Systems ," in *Current Topics in Artificial Intelligence (CAEPIA 2005)*, ser. LNAI, R. Marín, E. Onaindía, A. Bugarín, and J. Santos, Eds., vol. 4177. Springer, 2006, pp. 99–102.

[9] F. Brazier and N. Wijngaards, "Designing self-modifying agents," in *Proc. of the Creative Design Workshop, Dec. 2001*, 2001.

[10] D. T. Ndumu, H. S. Nwana, L. C. Lee, and J. C. Collis, "Visualising and debugging distributed multi-agent systems," in *Proc. of the 3rd Conf on Autonomous Agents*. New York, NY, USA: ACM, 1999, pp. 326–333.

[11] E. Serrano, J. J. Gómez-Sanz, J. A. Botía, and J. Pavón, "Intelligent data analysis applied to debug complex software systems," *Neurocomputing*, vol. 72, no. 13-15, pp. 2785–2795, 2009.

[12] L. Padgham, M. Winikoff, and D. Poutakidis, "Adding debugging support to the prometheus methodology," *Engineering Applications of Artificial Intelligence*, vol. 18, pp. 173–190, 2005.

[13] T. Salamon, "A three-layer approach to testing of multi-agent systems," in *Information Systems Development*, G. A. Papadopoulos, W. Wojtkowski, G. Wojtkowski, S. Wrycza, and J. Zupancic, Eds. Springer US, 2010, pp. 393–401.

[14] O. Balci, "Validation, verification and testing techniques troughout the life cycle of a simulation study," *Annals of Operations Research*, vol. 53, pp. 121–173, 1994.

[15] R. Junges and F. Klügl, "Evaluation of techniques for a learning-driven modeling methodology in multiagent simulation," in *Proc. of the 6th Int. Conf. MATES, Leipzig*, 2010.