*Proceedings of the 2012 Winter Simulation Conference*
*C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.*

# HOW TO DESIGN AGENT-BASED SIMULATION MODELS USING AGENT LEARNING

Robert Junges
Franziska Klügl

Örebro University
70182 Örebro, SWEDEN

## ABSTRACT

The question of what is the best way to develop an agent-based simulation model becomes more important as this paradigm is more and more used. Clearly, general model development processes can be used, but these do not solve the major problems of actually deciding about the agents' structure and behavior. In this contribution we introduce the MABLe methodology for analyzing and designing agent simulation models that relies on adaptive agents, where the agent helps the modeler by proposing a suitable behavior program. We test our methodology in a pedestrian evacuation scenario. Results demonstrate the agents can learn and report back to the modeler a behavior that is interestingly better than a hand-made model.

## 1 MOTIVATION

In this contribution, we address the problem of systematically designing an agent-based simulation model. Due to its "generative" nature, the design of an agent-based simulation model may impose a lot of challenges on the modeler. Actions and interactions on the agent-level *produce* the overall outcome on the system during simulation. Unless the modeler is very experienced, a trial and error procedure for finding the best model on the agent-level has to be applied. This is not desirable as only a systematic process guarantees that the final model satisfies quality criteria such as validity, reliability or clarity of the model design, which are essential for modeling and simulation. Also, such a process is not easy to convey to students or collegiate who want to adopt this new attractive modeling and simulation paradigm. Thus, a concise and generally applicable methodology is needed to guide the developer when analyzing the options of agent and model design.

During the last years, different methodologies have been proposed and single process elements discussed. In contrast to these, which mainly are based on classical simulation and software engineering approaches, we propose an approach that uses the agent idea in a special way for system analysis and model design: instead of being equipped with a full behavior program, our agents determine the appropriate behavior themselves using agent learning techniques.

In the following, we first give the background and related work for our endeavor in Section 2, discussing existing methodologies for agent-based simulation model development, as well as related work in agent learning. Then, in Sections 3 and 4, we present our methodology in a quite general way. After that we present the implementation (Section 5) and results (Section 6) from a first application in an pedestrian evacuation scenario. We end the paper with a discussion of the conditions under which the approach may work, in Section 7, and the future work, in Section 8.

## 2 BACKGROUND AND RELATED WORK

Phases of a systematic development of agent-based simulation models have been suggested by several researchers (Gilbert and Troitzsch 2005, Drogoul, Vanbergue, and Meurisse 2003, Richiardi, Leombruni, Saam, and Sonnessa 2006). While focusing on integration of data or different roles of participants in the study, their suggestions for procedures are quite similar to guidelines for general simulation study development

given by Law 2007, Balci 1994, Shannon 1998. Basic phases are initial works such as fixing the objective, making a profound analysis of the system or gathering necessary data and information. This is followed by making a model concept and elaborating it - this is basically model design. Model implementation and calibration are additional phases that are accompanied by analysis and validation phases. When the model is ready, the simulation study is completed by deployment runs, documentation and interpretation of the results. Clearly, bugs and deficiencies discovered in one phase may require re-working in previous phases.

Such overall process models frame our endeavor focusing on the design phase which requires a lot of creativity of the modeler. Model design for agent-based simulations is particularly difficult due to the previously mentioned generative character of such models: the outcome of the simulation is produced during a simulation run; the connection between actual model which is formulated on the agent level and the resulting simulation behavior often remains unclear. Thus, despite of the intuitiveness of its metaphor – actors in the real world correspond to agents in the simulation – it is not trivial to determine the appropriate "rules of behavior". Yet, surprisingly few suggestions have been made so far for systematic approaching the model design.

In the following we discuss about related work on solving the agent (system) design problem for agent-based systems and, as our approach is based on agent learning, on using learning techniques for generating agent behavior controllers.

### 2.1 Methodologies For Agent-Based Software Engineering

Similarly to other branches of software engineering (e.g., Object-Oriented Software Engineering), the focus of Agent-Oriented Software Engineering (AOSE) lies on systematically producing software that exhibits flexibility, scalability, robustness, ease-of-use, quality, etc. Some well-known early, yet influential AOSE methodologies are MaSE (DeLoach, Wood, and Sparkman 2001), Gaia (Wooldridge, Jennings, and Kinny 2000) and Prometheus (Padgham and Winikoff 2003).

For example, Gaia proposes systematic analysis, elaborating an organizational view in a somehow top-down way. The actual agent model is developed quite late in the overall process, derived from role models and made for providing the identified services. That means agent design is derived from organization design which is appropriate for the devised application area of Gaia: Multiagent Systems consisting of only a few, cooperating (heterogeneous) agents. For open multiagent systems with and without adaptive agents, a bottom-up approach starting from the agents, their behavior and their interactions seems to be more appropriate. Examples for methodologies that were specially developed to fill that gap are ADELFE (Bernon, Gleizes, Peyruqueou, and Picard 2003) or PASSI (Cossentino, Burrafato, Lombardo, and Sabatucci 2003). Both methodologies use simulation of the agent system for testing the design in an iterative manner - the starting point for agent design in ADELFE is the AMAS theory which is more focusing on repairing misleading behavior and interactions for reaching the intended behavior than trying to define the suitable behavior from the beginning. It has also been adapted for debugging in agent-based simulation (Klügl and Bernon 2011). PASSI uses patterns for agent structures and behavior as a first inspiration for agent (system) design.

Recent proposals in the area of agent-oriented software engineering propose the usage for model-driven design in which models of different phases are elaborated and translated step by step from concept to implementation (Aguero, Rebollo, Carrascosa, and Julian 2009).

### 2.2 From Software to Simulation Engineering

Pattern-oriented procedures have been also proposed in the area of simulation by Klügl and Karlsson (2009). However, these authors never realized their suggestions for agent design. So called pattern-oriented development of agent-based simulations are usually associated with the work of Grimm and Railsback (2005), who focus on the identification of data-patterns that have to be observable in both, real data and simulation-produced data, but give no advice for systematic model design. Klügl (2009) gave an

overview over different approaches for achieving a suitable model design; the work here basically extends the environment-based approach proposed there. More recently, Kubera, Mathieu, and Picault (2011) suggested an interaction-based methodology for reactive agents where the design of the agents is based on the proper identification of interactions and behavior primitives that implement this interactions.

### 2.3 Role of Agent-Learning In Simulation Development

Our idea is to use agent-level machine learning for supporting the model design. In previous work we have addressed the search for a machine learning that would be suited for this purpose. In Junges and Klügl (2010) we have tested Reinforcement Learning, Learning Classifier Systems and a particular setting of Neural Networks in how far such learning techniques are suitable for generating initial behavior programs for simulated agents. In Junges and Klügl (2011) we extended the investigation to include Genetic Programming. Even though all techniques performed well in our evaluation scenarios, Reinforcement Learning and Genetic Programming proved to be more adequate for modeling support: They provide more readable results in the form of behavioral rules or decision trees. Nevertheless, (even model-free) reinforcement learning gives us more information from the agent learning process: it generates a number of positive and negative situation-action pairs – which can be seen as behavioral rules – that can later be abstracted to other forms of representation.

There are a number of related works focusing on particular learning techniques in combination with abstraction and rule extraction techniques. Examples are: Hester and Stone (2009), using decision trees for improving the performance of reinforcement learning; or Jacobsson (2005), with sophisticated interpretation techniques. However, as this contribution is about a design methodology, a detailed discussion of alternative agent learning setups are omitted.

## 3 THE MABLe METHODOLOGY

The **M**odeling **A**gent **B**ehavior by **Le**arning methodology is based on the following idea: an appropriate conceptual model of the overall system can be developed by setting up a simulation model of the environment, determining agent sensors and effectors, and giving a function to evaluate agent validity as "performance". Put into in the simulated environment, agents are then able to learn an appropriate behavior program that fulfills the objective expressed by the performance function. The learned behavior is then providing the modeler with a draft agent design that can be used in later phases of model development.

The steps of the MABLe method are in particular:

1. *Separate agents and environment* and set which are the entities whose design shall be determined;
2. *Identify relevant aspects* (global status, global dynamics, local entities) of the part of the overall model that represents the environment for the agents in focus;
3. *Determine the effector commands or primitive actions* of the agent in this environment and how the environment changes in reaction to these actions;
4. *Determine what information from the environment* is given to an agent via its sensors;
5. *Decide on a learning architecture* that is apt to connect perceptions and actions of the agent appropriately for actually learning the agents' behavior. In the same step, the components describing the internal agent status are settled;
6. *Determine the performance function for providing feedback to the agents.* This reward is a representation of the distance between the valid behavior and what the agent actually performed. The actually structure of this performance function is clearly depending on the particular intended model, however mapping performance to such a feedback function, gives quite some flexibility for expressing "good" behavior. So it also supports a prescriptive style of modeling stating that the "right" behavior is the one that performs best, yet does not need to be necessarily optimizing something like efficiency of problem solution. It is possible to use a descriptive measure, e.g., for reproducing something like a failure rate. The performance function may be arbitrarily complex,

combining individual and system level evaluations, yet it needs to be defined for all possible situations and needs to fit to the selected learning mechanism.
7. *Implement the environmental model* including the performance function using a simulation platform, that provides all necessary simulation and learning functionality;
8. *Specify and implement the agents* basic behavior components and agent interfaces in combination with the chosen learning mechanism;
9. *Simulate, make the agents learn and test the overall outcome.* Execute simulation runs including learning agents, and analyze the results carefully for identifying potential artifacts that may come from an improper environmental model, performance function or weak agent interfaces;
10. *Analyze the resulting behavior model* for getting inspiration for the actual design should look like. This step may involve post-processing of the directly learned behavior. Analysis can be done by comparing the agents' learned behavior structures with some hypothesis or an existing solution that is not properly working.

Clearly, this is an iterative process in which steps have to be repeated if results turn out to be not good enough. Finally, the outcome should be a model of the agents that produces behavior optimizing the validity expressed in the performance function in the given environmental setup. This function should express the agent's goal with regards to the observed valid aggregated behavior of the system, so we also achieve a valid macro level behavior. That means, we define what is valid on the individual agent level that is then used to generate the system level behavior. It is critical however to integrate the overall and local validity description into the agent performance evaluation.

We focus on a reactive agent architecture, aiming at a simple representation of the agent behavior. The agents adapt to the environment continuously during simulation, they interact with other agents and they develop a knowledge base of the world, represented by the situation-action pairs.

There are a number of critical steps in this overall process: the actual learning mechanism used as well as the definition of the performance function. One could argue that the difficulties in defining agent behavior are just transferred to the definition of the performance function. This is discussed in Section 7. As indicated above, we already did a number of tests for determining a good learning mechanism.

Not all agent-learning architectures are equally apt for usage in the intended context. The learning technique must be able to cope with the level of complexity that is required for handling the environmental information and for learning a sufficiently complex behavior model. This is a quite standard requirement. For the aim of the methodology the user perspective is essential: the mechanism should produce behavior models that can be understood and interpreted by a human agent designer. Also, how the learning architecture actually works, shall be explainable to and by the human for generating justifiable outcomes.

## 4 LEARNING ARCHITECTURE IN MABLe

The chosen learning architecture for this contribution combines Reinforcement Learning with Decision Tree learning.

Reinforcement Learning is a well-known class of techniques. We use Q-learning (Watkins and Dayan 1992), which focuses on developing a policy that gives the expected utility of taking a specific action in a specific state. Q-learning is the simplest model-free reinforcement learning technique, directly producing situation-action pairs. The agents keep track of the experienced situation-action pairs by managing the so-called Q-table. Instead of directly learning an abstracted representation, the original experience data (Q-table) is kept and can be read and edited when interpreting the situation-action pairs as behavioral rules – which can be more interesting for our modeling support purpose than a black-box model.

However, even for small scenarios, it can be hard to oversee the actually learned behavior when looking at the entire Q-table. If the situation is captured using $n$ binary elements, the number of possible situation instances is $2^n$. Including situations that the agent rarely encounters. In this factored domain representation many times only one piece of information is relevant. For tackling this general readability

problem we use a decision tree representation of the implicit behavior of these situation-action pairs. Decision trees form and appropriate representation for decision-making processes having in mind human readability (Huysmans, Dejaeger, Mues, Vanthienen, and Baesens 2011). In this contribution we selected the C4.5 algorithm (Quinlan 1993) to generate decision trees, using the situation-action pairs generated by Q-learning as the input. The main purposes of generating this abstracted representation are: a) provide a more compact, abstract representation of the situation-action pairs set; and b) identify the relevant features from the state-space description.

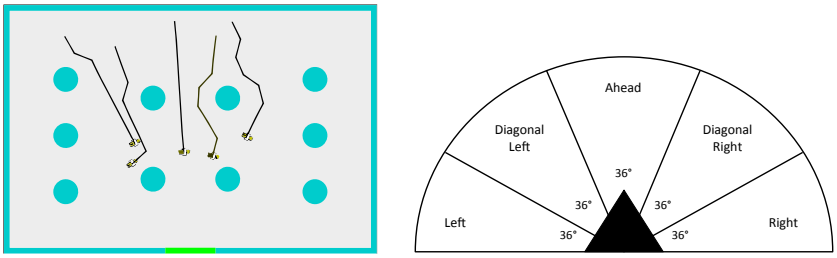
The decision tree is returned at the end of the simulation process for a given agent. It is generated by selecting the best situation-action pairs from the Q-table and learning an abstracted model from them. In C4.5, we map the individual components of the situation description as the attributes of the instances, and the action as the correct classification.

The best situation-action pairs are taken by first excluding those pairs that haven't been tested enough, as the confidence on their expected utility is lower. We count the "experience" of the situation-action pairs by recording how many times the agents use them. Then, we select for each situation the action with the highest Q-value, considering only those with non-zero, positive Q-values. The generated decision tree basically accomplishes the lacking abstraction that makes the resulting behavior description transparent for the designer – if the resulting tree sufficiently resembles the best pairs.

## 5 CASE STUDY

As a case study for illustrating how the design methodology work, we use an evacuation scenario. This test scenario can be coined as a shared-environment-actor model (Klügl 2009). Agents are not directly interacting for example to coordinate their actions, but indirectly. For achieving coordinated behavior or for maximizing their individual utility, agents have to predict what the other agents might do in the future.

1. *Separate agents and environment*: This is obvious in the case of the evacuation scenario. The only active entities are the pedestrians that are supposed to leave the room in an calm and ordered way. The environment consists of a hall with columns and an exit – the layout of the built environment is usually given in evacuation simulations.
2. *Identify relevant aspects of the environment*: There are two important elements in the environment: the obstacles that the agents should not collide with and the exit they should reach as fast as possible. Also, the other agents form the individual environment of an agent who shall not collide with its fellow agents as well. The basic time advance shall be $1/10$ sec.
3. *Determine the effector commands or primitive actions*: The only actions that the agents can take are movement actions. In the case study there are in principle two options: a) the action set consists of a move action with that the agent moves a unit according to its current orientation and a number of turn actions with given discrete angles. or b) have a set of move actions such as move-to-exit, move-slightly-right... always keeping a basic orientation towards the exit. We decided for the second option as it makes the learning process easier as the agents just need to learn a collision avoidance behavior and do not need to perform an additional navigation task.
4. *Determine the information from the environment*: Again there are a number of options to choose, ranging from full information on all positions and orientations of all other agents and obstacles, to a reduced, discretized perception that just covers the area that needs to be tested for the next step. Given that the first option would not be realistic – even in our smoke-free, bounded area –, we selected a discretization with a limited perception distance challenging the learning mechanism using a very restricted sample. Figure 1 illustrates the test scenario and the perception of the agents.
5. *Decide on a learning architecture*: We selected the above introduced combination of reinforcement learning and decision tree learning. As mentioned above, we also tested other mechanisms in this or similar scenarios.




(a) Scenario with 5 agents (b) Perception sectors

Figure 1: Scenario example and agent perception sectors.

6. *Determine the performance function for providing feedback to the agents.* We assumed that the performance function can be seen clearly as a reward function optimizing the task performance, not taking into account panic behavior or other human factors such as realistic reaction times, etc. Thus, the performance is measured and a reward is given to each agent individually after each step containing the following components: (1) *Exit Reward* indicating whether the exit was reached; (2) *Collision Reward* punishing collisions; and (3) *Distance Reward* indicating whether the agent came closer to the exit. More details can be found in Junges and Klügl (2012b).
7. *Implement the environmental model*: We implemented the setup in the fast prototyping, visual environment SeSAm (www.simsesam.de).
8. *Specify and implement the agents*: The agents and the reinforcement mechanism were also implemented using SeSAm.
9. *Simulate, make the agents learn and test the overall outcome.* and *Analyze the resulting behavior model* We give a glance on the results in the next section illustrating what form of agent behavior design can be achieved with this overall process.

## 6 A GLANCE ON RESULTS

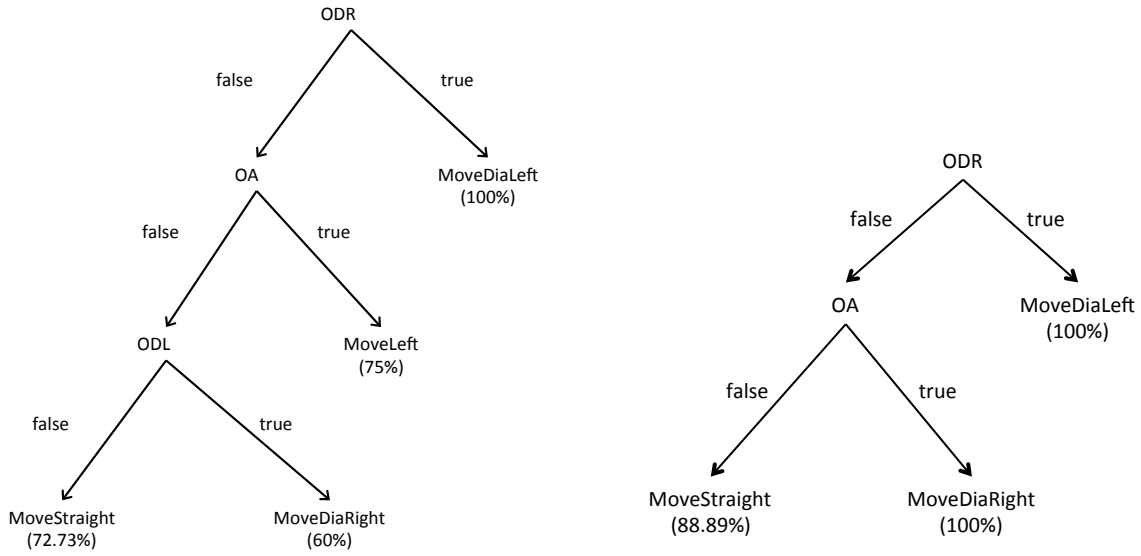
In general, our tests show that the agents indeed learn a behavior program that interestingly is better than a hand-made behavior model as the agents avoid other agents even if they cannot clearly distinguish between other agents and static obstacles.

Prior to formulating the methodology presented in this paper, we tested our design strategy with regards to the learning mechanisms: how sensitive the outcome produced is to **a)** changes in the settings of the learning algorithm and **b)** in the weights of the reward function. Our focus in the present contribution is on the methodological aspects of agent behavior engineering, while in the below-mentioned previous work we focused on the learning aspects of our implementation.

**a)** In Junges and Klügl (2012b), we tested learning performance and characteristics of the generated decision trees considering different setups of the learning mechanisms. We presented a series of experiments where we changed parameters like the learning rate functions and the explore-exploit trade-off. It turned out that, if there is no convergence in the learning done by the Q-learning, the generation of decision trees for producing a more readable behavior program must fail as there is no action clearly assigned to a perceived situation. We also could see that the final behavior program may appear to be similar although very different situation-action pairs were used for its generation.

**b)** We also tested the effect of the reward function guiding the reinforcement learning on the final decision tree outcome (Junges and Klügl 2012a). We ran experiments similar as to Junges and Klügl (2012b), but this time varying the reward configuration, instead of the learning configuration. It turned out that there are interesting differences in how elaborate the decision trees are, which cannot be seen in the mere performance evaluation of the reinforcement learning. The results in general were uncritical: with a higher weight for the *Distance Reward*, the agents tend to develop actions that lead to shorter paths, at the same time as they try to avoid collisions. A shorter path means the selection of movements that direct

the agent to passing obstacles in smaller distance as the agent might want to risk colliding with another agents. This is different in the case with higher *Collision Reward*: the decision tree points to a selection of perceptions and actions that lead to the development of a wider collision-avoidance path. This comes from the fact that is hard to predict other agents' movements if agents cannot distinguish between pedestrians and columns. If the weight on the collision avoidance is higher, the agents learn to be more "cautious". That means finally those trees with higher collision reward were more elaborated than the others. Figure 2 shows example trees generated from the experience of one of the agents in each simulation setting presented above. The codes in the nodes correspond to different perceptions: *O* for Obstacle, *D* for Diagonal, *A* for Ahead, *L* for Left and *R* for Right. This shows that the behavior program is not really complete, but very illustrative. Further tests against artifacts have to be performed.




(a) Collisions

(b) Evacuation

Figure 2: Behavior trees for different objective function details putting more focus on collision avoidance or fast evacuation.

Each agent learns individually, it has no shared memory with other agents. In our case all agents have the same goal, they use the same objective function. However, at the end of the simulation process each one develops its own controller, or behavior program. This happens because they have different experiences during the simulation, given by the situations they face. As we randomly reposition the agents every time a new trial begins – a trial being the process of exiting the room – they learn different paths to the exit and experience different situations. Also, it is not trivial to predict the actions of other adaptive agents. The same situation may have different best actions for different agents and the same action may have also different evaluations for different agents. Moreover, analyzing a number of learned models may help the developer to find issues to correct in the interfaces or environmental model when agents are not agreeing on a specific behavior. This means that at the end of the simulation we end up with a number of behavior programs, which the developer needs to evaluate for its semantics and performance – that's when other scenario learning metrics come into place, like the number of collisions. These learned models, or the best one, serve as an inspiration to the human modeler to further develop the agent behavior model.

## 7 DISCUSSION

Considering the case study, the question of applicability of the proposed methodology has to be discussed. We identified the following requirements for the applicability of the proposed methodology:

- There must be a precise idea of how the environment works, so the proper interfaces between an agent and its environment can be set. Also, the definition of a proper performance function is essential for determining the behavior. This function is also highly dependent on the environment and may not be defined easily. This is true especially in the case of the simulation of emergent phenomena. In those cases it is not obvious how the individual agent actions and interaction contribute to the overall phenomenon.
- For the used learning architecture, it should be possible to discretize agents perceptions (factored domain) and actions. So that it becomes clear how the reward is assigned to the their combination. It is important to notice, nevertheless, that single-agent reinforcement learning is already a challenging task in multiagent systems when the Markov properties are violated by the dynamic nature of adaptive agents (Busoniu, Babuska, and De Schutter 2006). Learning convergence becomes a problem when abstracting the behavior learned (Junges and Klügl 2012b): actions may have similar evaluations for the same situation, mainly due to the partial-observability of the environment.

Even if the learned agent behavior is not so remarkable, the analysis done with the application of this methodology is useful also just for validating the environment or for identifying implicit assumptions about the environment: Under what circumstances a specific behavior – emergence – is valid? How the environment can be shaped to influence a particular aggregated behavior?

We use the approach for generating inspiration for the agent behavior model. The most abstract way of describing behavior is a set of rules — which are generated by the learning mechanism. If the behavior description is directly implemented without further elaboration, the agent architecture may be characterized as reactive.

## 8 CONCLUSION AND FUTURE WORK

In this contribution we have presented MABLe, a methodology for agent behavior design in multiagent simulation. It is fundamental to provide a systematic way of bridging the micro-macro gap existing between the agent's local behavior and the generated system phenomenon. Without it, the process is dependent on the experience of the modeler and may result in a series of random trial and error rounds. We have presented the overall structure of the methodology, the learning architecture and an initial application to a pedestrian evacuation scenario.

Future steps include refining the learning architecture: Other forms of reinforcement learning have to be tested, mainly with regards to model-based approaches. About model-free approaches we need to investigate other possibilities for the post-processing step, which should improve the generalization aspects of the learned model: Multi-label classification (Tsoumakas and Katakis 2007) offers an alternative to the post-processing step with decision trees. It aims at overcoming the problem when different actions (label classes) may be possible, but there is a lack of information to explicitly choose one of them. This lack of information in our case comes form the lack of convergence in individual reinforcement learning in dynamic MAS environments. Also, the negative experienced situation-action pairs could be used as counterexamples in the decision tree building process (Nicolas Cebron and Lienhart 2012). This way, we gain information from almost every example situation-action pair experienced.

Finally, the methodology needs to be further validated. In this sense we also aim at experimenting with more complex scenarios: increasing the number of agents; broadening the perception range of the agents, to include more perception variables; adding more elements to the objective evaluation; and also by including heterogeneous agents, that are required to perform different roles and are subject to different

objective functions. Moreover, organizational structures or normative elements, explicit communication and explicit teamwork should be considered.

## REFERENCES


Aguero, J., M. Rebollo, C. Carrascosa, and V. Julian. 2009. "Agent Design Using Model Driven Development". In *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, edited by Y. Demazeau, J. Pavon, J. Corchado, and J. Bajo, Volume 55 of *Advances in Intelligent and Soft Computing*, 60–69. Springer Berlin / Heidelberg.

Balci, O. 1994, December. "Validation, verification, and testing techniques throughout the life cycle of a simulation study". In *Proceedings of the 1994 Winter Simulation Conference*, edited by J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 215–220. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bernon, C., M.-P. Gleizes, S. Peyruqueou, and G. Picard. 2003. "ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering". In *Engineering Societies in the Agents World III*, edited by P. Petta, R. Tolksdorf, and F. Zambonelli, Volume 2577 of *LNCS*, 70–81. Springer Berlin.

Busoniu, L., R. Babuska, and B. De Schutter. 2006. "Multi-Agent Reinforcement Learning: A Survey". *9th Int. Conf. on Control Automation Robotics and Vision*:1–6.

Cossentino, M., P. Burrafato, S. Lombardo, and L. Sabatucci. 2003. "Introducing pattern reuse in the design of multi-agent systems". In *Proceedings of the NODe 2002 agent-related conference on Agent technologies, infrastructures, tools, and applications for E-services*, edited by R. Kowalczyk, J. P. Müller, H. Tianfield, and R. Unland, NODe'02, 107–120. Berlin, Heidelberg: Springer-Verlag.

DeLoach, S. A., M. F. Wood, and C. H. Sparkman. 2001. "Multiagent Systems Engineering". *International Journal of Software Engineering and Knowledge Engineering* 11 (3): 231–258.

Drogoul, A., D. Vanbergue, and T. Meurisse. 2003. "Multi-agent based simulation: where are the agents?". In *Proceedings of the 3rd international conference on Multi-agent-based simulation II*, edited by J. S. ao Sichman, F. Bousquet, and P. Davidsson, MABS'02, 1–15. Berlin, Heidelberg: Springer-Verlag.

Gilbert, N., and K. G. Troitzsch. 2005. *Simulation for the Social Scientist, 2nd edition*. Buckingham: Open University Press.

Grimm, V., and S. F. Railsback. 2005. *Individual-Based Modeling and Ecology*. Princeton University Press.

Hester, T., and P. Stone. 2009, May. "Generalized Model Learning for Reinforcement Learning in Factored Domains". In *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, edited by K. S. Decker, J. S. Sichman, C. Sierra, and C. Castelfranchi, 717–724. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Huysmans, J., K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. 2011. "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models". *Decision Support Systems* 51:141 – 154.

Jacobsson, H. 2005. "Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review". *Neural Comput.* 17:1223–1263.

Junges, R., and F. Klügl. 2010. "Evaluation of Techniques for a Learning-Driven Modeling Methodology in Multiagent Simulation". In *Multiagent System Technologies*, edited by J. Dix and C. Witteveen, Volume 6251 of *Lecture Notes in Computer Science*, 185–196. Springer Berlin / Heidelberg.

Junges, R., and F. Klügl. 2011. "Evolution for modeling: a genetic programming framework for sesam". In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, edited by N. Krasnogor and P. L. Lanzi, GECCO '11, 551–558. New York, NY, USA: ACM.

Junges, R., and F. Klügl. 2012a. "Behavior Modeling from Learning Agents: Sensitivity to Objective Function Details". In *The Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, edited by W. van der Hoek, L. Padgham, V. Conitzer, and M. Winikoff, 1335–1336.

Junges, R., and F. Klügl. 2012b. "Programming Agent Behavior by Learning in Simulation Models". *Applied Artificial Intelligence* 26 (4): 349–375.

F. Klügl 2009. "Agent-based Simulation Engineering". Habilitation Thesis, University of Würzburg.

Klügl, F. 2009. "Multiagent Simulation Model Design Strategies". In *MAS&S Workshop at MALLOW 2009, Turin, Italy, Sept. 2009*, edited by G. Fortino, M. Cossentino, J. Pavn, and M.-P. Gleizes, Volume 494 of *CEUR Workshop Proceedings*, 361–368: CEUR-WS.org.

Klügl, F., and C. Bernon. 2011. "Self-Adaptive Agents for Debugging Multi-Agent Simulations". In *Proceedings of ADAPTIVE 2011, Rome, September 2011*, edited by J. Fox and A. Rausch, 79–84.

Klügl, F., and L. Karlsson. 2009. "Towards Pattern-Oriented Design of Agent-Based Simulation Models". In *Multiagent System Technologies, 7th German Conference, MATES 2009, Hamburg, Germany, September 9-11, 2009. Proceedings*, edited by L. Braubach, W. van der Hoek, P. Petta, and A. Pokahr, 41–53.

Kubera, Y., P. Mathieu, and S. Picault. 2011. "IODA: an interaction-oriented approach for multi-agent based simulations". *Autonomous Agents and Multi-Agent Systems* 23 (3): 303–343.

Law, A. M. 2007. *Simulation Modeling and Analysis*. 4th ed. McGraw-Hill Higher Education.

Nicolas Cebron, F. R., and R. Lienhart. 2012. "Decision Tree Induction from Counterexamples". In *Int Conf. on Pattern Recognition Applications and Methods (ICPRAM)*, edited by P. L. Carmona, J. S. Sánchez, and A. L. N. Fred, 525–528.

Padgham, L., and M. Winikoff. 2003. "Prometheus: a methodology for developing intelligent agents". In *Proceedings of the 3rd international conference on Agent-oriented software engineering III*, edited by C. Castelfranchi and W. L. Johnson, AOSE'02, 174–185. Berlin, Heidelberg: Springer-Verlag.

Quinlan, J. R. 1993, January. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. 1 ed. Morgan Kaufmann.

Richiardi, M., R. Leombruni, N. J. Saam, and M. Sonnessa. 2006. "A Common Protocol for Agent-Based Social Simulation". *Journal of Artificial Societies and Social Simulation* 9 (1): 15.

Shannon, R. E. 1998, December. "Introduction to the art and science of simulation". In *Proceedings of the 1998 Winter Simulation Conference*, edited by D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, 7–14. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Tsoumakas, G., and I. Katakis. 2007. "Multi-label classification: An overview". *Int J Data Warehousing and Mining* 3:1–13.

Watkins, C. J. C. H., and P. Dayan. 1992. "Q-learning". *Machine Learning* 8 (3): 279–292.

Wooldridge, M., N. R. Jennings, and D. Kinny. 2000. "The Gaia Methodology for Agent-Oriented Analysis and Design". *Journal of Autonomous Agents and Multi-Agent Systems* 3 (3): 285–312.

## AUTHOR BIOGRAPHIES

**ROBERT JUNGES** is a PhD Student at the Modeling and Simulation Research Center of the Örebro University (Sweden). His PhD thesis is on how to support modeling using machine learning. He holds a Master in Computer Science from UFRGS, Brazil. Email address: robert.junges@oru.se.

**FRANZISKA KLÜGL** is a Full Professor in Information Technology at the Örebro University (Sweden), where she heads the Multiagent Simulation Group in the Modeling and Simulation Research Center. She holds an Habilitation and PhD in Computer Science from the University of Würzburg, Germany. She is interested in questions on model and simulation engineering especially for multi-agent systems. Her email address is fkluegl@acm.org.