*Proceedings of the 2006 Winter Simulation Conference*
*L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.*

# SPECIFYING AND SIMULATING MODERN WARFARE SCENARIOS WITH ITSIMBW

Philipp Hügelmeyer
Timo Steffens
Thomas Zöller

Fraunhofer AIS-ART
Schloss Birlinghoven
53754 Sankt Augustin
GERMANY

## ABSTRACT

The aim of this paper is the presentation of the military multi-agent simulation system ITSimBw. Its decisive features include a strictly agent-based approach to modeling, in which every entity in a simulated environment can potentially become an active element. Technologically, ITSimBw is based on the *Flip-Tick-Architecture*. Moreover, a focus on IT and communication aspects is one of its important characteristics. Additionally, the impact of scaling aspects in the design of scenarios and their support by the simulation system is addressed.

As the utility of simulation strongly depends upon the quality of the employed scenarios, ITSimBw also contains a unique approach to scenario description, termed LAMPS (Language for Agent-based Modeling of Processes and Scenarios). LAMPS is based on high-level Petri-Nets and enables the specification of individual agent behavior as well as complex scenarios in a uniform way.

## 1 INTRODUCTION

In this contribution, we present the military multi-agent simulation environment ITSimBw and its description language LAMPS, which is used for the specification of agent behaviors as well as complete scenarios.

Due to the growing importance of network centric capabilities in military operations, one of the main focus points for the development of ITSimBw is the faithful modeling of IT and communication aspects. Clearly, these issues cannot be considered in isolation, but have to be seen in conjunction with the military mission, command, and report structures, and the influences exerted by the environment.

Consequently, the simulation system must be flexible enough to model any conceivable action or effect for all entities involved. Although military operations generally follow fixed doctrines and rules of engagement, every item in a simulation should be enabled for active participation. ITSimBw addresses this point by strictly adhering to an agent-oriented paradigm which allows for the specification of autonomous, situation-based behavior for all entities. This extended agent concept includes environment, weather, bridges, obstacles, and the like as active elements. This means that all effects and events are handled as actions of agents.

One of the main challenges of simulations is the optimal choice concerning the degree of model detail. Coarse models commonly yield very broad and abstract results with little value for the military practitioner. Overly detailed models on the other hand produce outcomes that are too dependant on minute situational details thus preventing their applicability to other scenarios even if they are highly similar. Therefore, scaling aspects play a major role in the design of simulation tools.

Another important issue for any simulation system is the precise and comprehensive description of the scenarios which are to be examined. To this end, ITSimBw encompasses its own LAMPS description language. Being based on high-level Petri-Nets, it can be represented graphically and by rule-sets. Moreover, due to its generality, it is equally well suited for the description of complete scenarios as for the specification of agent behavior.

In our view, the unique combination of features outlined in the preceding paragraphs distinguishes ITSimBw from other commonly used military simulation tools such as MANA (Stephen, Anderson and Lauren 2002) or Pythagoras (Bitinas 2002).

In the next section, the key concepts of ITSimBw, which have already been touched upon in this introduction, are explained in more detail. Section 3 then gives an exposition of the underlying Flip-Tick-Architecture, followed by a presentation of the main concepts of the LAMPS description language in Section 4. Finally, Section 5 concludes the paper.

## 2 CORE CONCEPTS

### 2.1 Strict agent-based modeling

As has already been stated in the introduction, ITSimBw is designed from the ground up with an agent-oriented paradigm in mind. Consequently, virtually every part of a simulation can be modeled as an autonomous agent. This even includes the landscape, weather phenomena like clouds or rain, and inanimate objects such as bridges and other buildings. Although many of these agents are usually passive, they can be activated at any time if required by the investigated scenario.

The principle of autonomy of agents forbids direct manipulation of internal data structures of other agents. Thus, any influence that one agent exerts on another is mediated via messages. Effects like weapon fire, weather influences, and the like, are communicated via specialized messages which carry the effect type. The concrete impacts of effects, however, are stored in a central data structure, the effect table. Thus, the ITSimBw user has a single reference point for changing the impact of effects that is accessible even to non-programmers.

The extension of all objects belonging to the simulated environment is modeled by a voxel-space. This allows the application of high-performance ray-tracing techniques for determination of visibility, potential weapon influences, and communication abilities.

Additionally, the landscape agent carries a semantic map containing information about terrain types (land, water, swamp, etc.) which can be queried by other agents in order to determine their possibilities for movement. An example view of unit agents situated in a simulated environment is given in Figure 1.

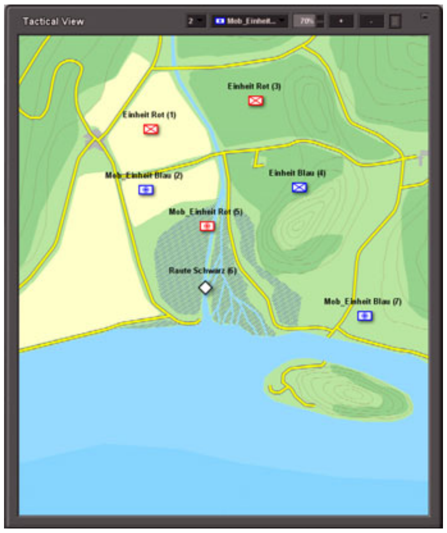

Figure 1: 2D Situation Display

### 2.2 Communication focus

Although ITSimBw does not force its users to model IT and communication equipment as individual agents, this procedure is highly encouraged by the pervasive agent-based modeling paradigm.

The simulation system supports communication modeling essentially by two means:

- A message format for simulated communication acts between agents is provided which allows the detailed specification of communication channel, medium, and range.
- The voxel-space mentioned in the previous subsection is used to determine environmental effects on communication, e.g. mountains causing radio pockets, as well as jamming by opposing forces.

These two features in conjunction allow for a detailed and realistic modeling of communication chains both inside and across different echelons.

### 2.3 Scalability

A multi-agent system for military simulation tasks is typically confronted with three interdependent notions of scale. One concerns the appropriate time resolution at which the simulated events occur. Here, very small time intervals (e.g. for fast moving objects such as fighter jets) as well as time spans of considerable length (e.g. in the unfolding of large scale military operations) have to be addressed.

Moreover, the granularity of the environment model, containing terrain, roads, buildings, and the like, is of major importance. It should, therefore, be adjustable according to the character of the mission. Facilities for the import of spatial data at the required resolution from external sources are also indispensable parts of a useful simulation environment.

Furthermore, scale is also relevant for the simulated units as well as technical systems. Depending on the given scenario, representation at the level of large scale organizational units as divisions or brigades may suffice, whereas other situations might require to model mission critical groups down to the level of the individual soldier. Correspondingly, the technical infrastructure, e.g. IT, communication, and weapon systems, underlie the same considerations concerning the proper level of detail. Clearly, the same holds true for physical effects like weather, contamination by nuclear, biological or chemical agents, or similar events.

Finding the right level of simulation detail is a difficult task due to its strong dependence on the scenario under consideration as well as on the concrete question which is to be examined. Thus, automatic procedures addressing this issue are hardly conceivable. Consequently, our simulation system is based on a few generic principles outlined above, ensuring consistent modelling and operation on all scales, thereby actively supporting the model designer. In particular, ITSimBw imposes no artificial restriction concerning the model granularity. The modeller is thus free to decide upon an appropriate level of detail concerning the participating agents and their behavior as well as the temporal resolution. Moreover, scalability was one of the major design objectives in the development of our scenario description language, which is presented in Section 4.

## 3 THE FLIP-TICK ARCHITECTURE

The ITSimBw simulation environment is based on the Flip-Tick Architecture (FTA) (Richter 1999), which has its roots in the JANUS project developed at the *Gesellschaft für Mathematik und Datenverarbeitung (GMD)* (Beyer and Smieja 1994). At its core, FTA is a design paradigm for scalable distributed systems that exhibit a priorily unknown dynamic characteristics as well as disturbances and inaccuracies which are difficult, if not impossible, to model in a closed-form mathematical approach. As every multi-agent system, FTA is based on the concept of a society of agents (Weiss 1999). It comprises four classes of entities: actors (i.e. agents), assemblies, tags, and tag-boards.

The aforementioned society of agents $A$ is formed by a set of individual agents $a_j$, which are called actors in FTA terminology:

$$A = \{a_1, ..., a_j, ...a_k\}. \tag{1}$$

Each actor $a_j$ is composed of a set of typed attributes $U_j$, whose value assignment determines the agent's state, together with an action function, which entails all operations that can be performed by the actor.

$$a_j = (U_j = (u_{1,j}, ..., u_{m,j}), f_j). \tag{2}$$

Structural information concerning agents, i.e. names and types of attributes, is described via agent types. Formally, we have *type(*$a_j$*)* = $t$, if and only if agent $a_j$ is of type $t$. The system supports agent templates that can be used to store prototypical value assignments. Thus, an individual agent can be created either by instantiation of its type, or by copying from a pre-defined template.

The principle of autonomy of agents forbids the direct manipulation of internal data structures and behaviors of other agents. Consequently, all interactions between agents are handled via messages. Formally, a message $N_v$ is comprised of a number of attributes:

$$N_v = (u_{1,v,...,}u_{l,v}). \tag{3}$$

As agents, messages are typed entities. The set of all message types is given by

$$N = \{N_1, ..., N_h\}. \tag{4}$$

Upon receiving a request, the agent is able to analyze its content and to decide whether it wants to comply. The basic unit of execution is called a *cycle*. During one cycle, the agent reads its messages and triggers the appropriate actions, which might consist of writing messages to other actors.

An assembly is a set of agents sharing a common pace, i.e. all elements of an assembly have the same time resolution *dt*. This in turn implies that their cycles are synchronized and that the assembly switches from cycle to cycle as regularly as the tick of a clock. Hence the term *Tick* in FTA. It is important to note, that different agents do not necessarily share the same time resolution. Instead, the architecture supports individual running speeds for every actor. Moreover, time steps can vary from cycle to cycle. Thus, adaptive control of time increments can be realized (see below). This is particularly valuable for increasing the time resolution in the computation of dynamics equations for fast moving objects.

The messages used for inter-agent communication are called *tags* in FTA. We distinguish between three different categories of tags:

- Message tags are the carriers of communication between agents. Orders are embedded in message tags using XML syntax.
- Position tags contain the current position of agents. They are sent when actors change their position.
- Effect tags are used to mediate effects like weather influences, weapon impact, and the like.

Instead of setting up a direct communication with other actors, agents register with one ore more *tag-boards*, to which they send their messages. Thus, tag-boards serve as the functional units for handling messages in the FTA system. In formal terms, a tag-board forms a medium $M_n$ for message exchange, while a FTA system is capable of supporting multiple media:

$$M = \{M_1, ..., M_m\}. \tag{5}$$

A tag-board consists of two sides. One is write-only and contains all tags sent to the board in time step $t$, whereas the other side is read-only and encompasses all tags written in time-step $t$-1. Analogously to agents, each tag-board has its own time resolution and thus its own cycle time. During a board cycle, the write-only side is flipped over. Thereby, the read-only part mirrors the tag content of the write part from the previous time-step. The write-only side is deleted after flipping. In this way, the lifetime of tags is effectively controlled by the time-scale of the pertaining board. These interrelations are depicted graphically in Figure 2.

With this approach, fully synchronized (all agents and tag-boards share the same time resolution) as well as completely asynchronous systems (every agent and every board has its own time-scale) can be modeled in terms of the FTA.

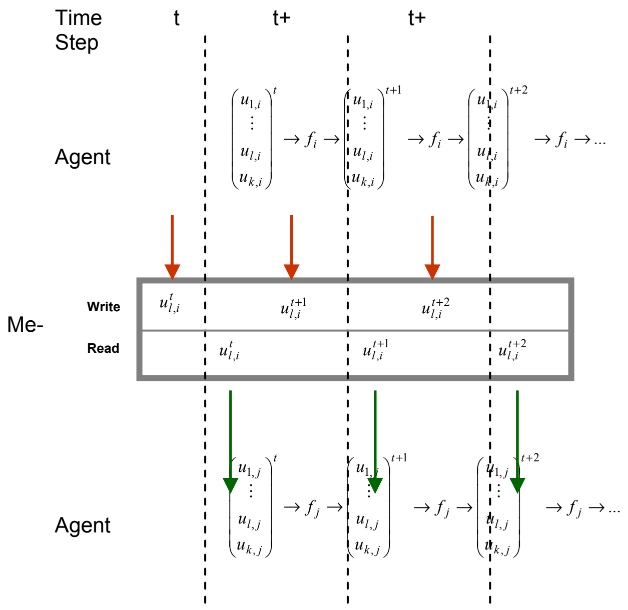

Figure 2: Inter-Agent Communication via Tag-Board

The mathematical model of this agent architecture is a system of flexibly coupled inhomogeneous difference equations

$$D = (D_1, \ldots, D_k), \tag{6}$$

where each agent computes an equation from D. In this setting, the attribute list of an actor corresponds to the variable vector of the pertaining difference equation. Thus, equation $D_i$ of agent $a_i$ computes

$$U_i^{t+dt_i} = f_i(U_i^t), \tag{7}$$

where $dt_i$ denotes the time step size of agent $a_i$. During the iteration of $D$, the society of agents runs through a set of states $Z = \{Z^t\}$, where each of these system states is the union of attribute sets from all participating agents:

$$Z^t = (U_1^t, \ldots, U_i^t, \ldots, U_k^t). \tag{8}$$

Since the $D_i$ are computed using potentially different $dt_i$ , the involved agent attribute sets $U^t_i$ might be undefined for certain $t$. In this case, either the $U_i$ from the last time step of $D_i$ is used, or $U_i$ is explicitly recomputed. We thus have

$$Z = Z^0, \ldots, Z^{t_i}, Z^{t_j}, \ldots, Z^e$$

$$\text{where } t_j - t_i = \underset{h=1}{\overset{k}{Min}}\, dt_k. \tag{9}$$

A graphical visualization of FTA (omitting the discrimination of different tag-board sides) is shown in Figure 3 Arrows denote read() and write() operations. The colors specify types of agents and tags. It is depicted that agents can subscribe to more than one tag-board.

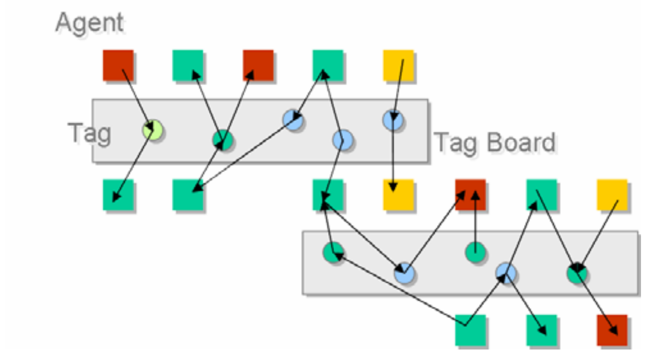

Figure 3: The Flip-Tick Architecture

## 4 LANGUAGE FOR AGENT-BASED MODELING OF PROCESSES AND SCENARIOS (LAMPS)

The utility of modeling and simulation tools for military applications highly depends on the quality of the utilized scenarios. Therefore, the careful design and precise description of scenarios as well as the ability to reliably communicate these assets is of prime importance. Moreover, considering non-deterministic wargaming, one must be able to specify and document alternative courses of action in order to enable a comprehensive analysis as well as comparisons with the optimal procedure.

Therefore, a precise formal specification of complete scenarios constitutes an essential part of modern simulation environments. A complete description of a scenario has to include the environment (landscape, terrain, weather), forces, enemies, and non-combatants. To address this point, we are currently developing a graph based scenario description language. While languages for the specification of simulation models abound, LAMPS is to our knowledge the only one that supports both the description of scenarios and the executable specification of agent behavior for compound agent groups down to individual agents. Existing simulation models are usually extensions of programming languages such as C/C++, e.g. Maisie (Bagrodia and Liao 1994) and the SPaDES environment (Teo, Tay and

Kong 1998), or Java, e.g. SILK (Healy and Kilgore 1997), and the SSJ package (L'Ecuyer and Buist 2005). In contrast, LAMPS is based on high-level Petri nets (Jensen 1992). Thus, LAMPS inherently supports parallel simulation, and is not just an extension of sequential simulation languages like Maisie, Modsim II (Bryan 1989) or SIMSCRIPT III (Rice, Marjanski, Markowitz, and Bailey 2005). Like other modern simulation languages (L'Ecuyer and Buist 2005), LAMPS can be displayed both graphically and as a rule-set.

## 4.1 Basic concepts of LAMPS

According to its underlying formal model of high-level Petri nets, LAMPS consists of the following four basic concepts:

- *Places* hold states, i.e. they contain subsets of the agent attributes. The contents of a place is called token and can be of an arbitrarily complex structured type. Places can contain several tokens of different or identical types. Since the whole simulation (incl. troops, environment, and equipment) is modelled as agents, the state of the simulation is described by the contents of all places.
- *Actions* describe the effects that agents apply to themselves or to other agents.
- *Relations* denote the links between places, actions, and agents. Relations correspond to the arcs in the Petri net model.
- *Agents* correspond to the conditions of Petri nets. An agent in LAMPS observes the set of places that have relations to the agent's actions. Based on these places the agent decides which actions are executed and which parameters should be used.

Figure 4 depicts a simple example using the four basic concepts. The agent *Inf A* observes the place *Enemy spotted*. If the place contains a token, the agent executes action *Combat*, which sends a token into the place *Resistance broken*.

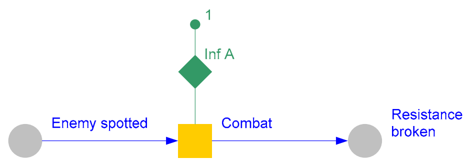

Figure 4: LAMPS Fragment Using the Four Basic Concepts

## 4.2 Generality and scalability

Due to these generic concepts, LAMPS can handle environmental effects like weather, human actions, and physical effects (such as the malfunction of equipment) in the same way. Every action and event is managed by an agent which decides whether and how to execute the action based on its own attributes and other observable attributes. For example, an agent can represent a bridge and decide based on its own status attributes and the locations of other agents whether the bridge should collapse. Note that an agent is usually associated with several actions.

The main difference to flowcharts is that LAMPS is parallel in nature. At any given time more than one place can be filled with tokens and in any place there can be several tokens. This way, actions can be executed concurrently.

LAMPS does not explicitly model time, as it assumes a cycle-based simulation-model. Actions have no duration but are a sequence of short actions that are executed each cycle. Furthermore, LAMPS does not need explicit synchronisation constructs, because ITSimBw uses a cycle-based simulation with global time and synchronisation is achieved via places and conditions. The next cycle is started after all agents have sent their actions.

Furthermore, LAMPS is inherently scalable to different abstraction levels since it is based on high-level Petri nets. Places can be aggregated to form more abstract places. The same holds true for actions. Figure 5 illustrates encapsulating actions. The action is defined as two parallel actions *A1* and *A2* on a more fine-grained level. The abstract action *A* has the same interface as the two combined interfaces of *A1* and *A2*. Note that *Action A* includes on the more fine-grained level an order for Agent *Inf B* to execute *Action A2.*

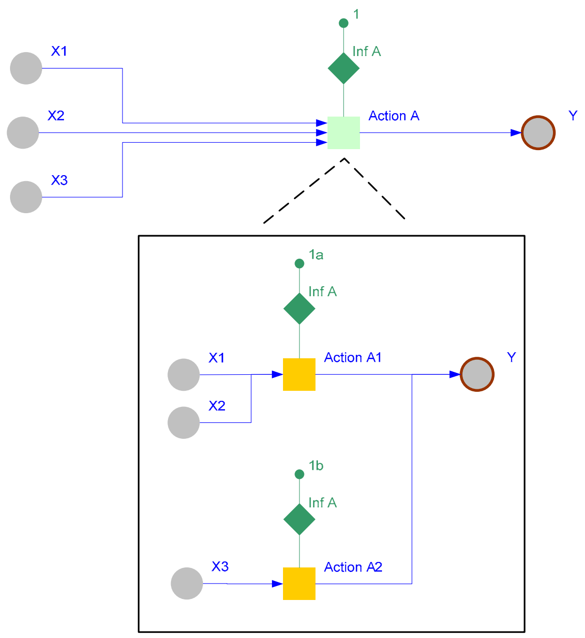

Figure 5: Example for Different Abstraction Levels for an Action *A*

Actions can be executed locally or remotely. This means for example, that an agent can give orders to subordinate agents by sending LAMPS fragments. In the example given in Figure 5, the superior agent *Sup A* orders its subordinate *Inf B* the command to execute action *A2*. This is denoted by the fact that in the LAMPS specification for agent *Sup A*, another agent (*Inf B*) is associated with action *A2*. *Inf B* then sends a result token into place *Y* (where in this example another token is sent by *Sup A* as result of action *A1*). *Sup A* can then access both tokens in place *Y*. Note that this order is not simply a name that references an action that *Inf B* already has to know. Instead, new behaviors can be sent to agents by specifying the action as a LA Modeling of Processes and MPS fragment. The only elements that the agent has to know are the basic action primitives which are coded in C++. The logic of combining these basic actions to complex behaviors is contained in the LAMPS fragment.

LAMPS can be used to express non-deterministic effects. The places resulting from an action are alternative results which can be weighted by adding attributes to the relation. For example, the action *shoot* may result in a place *success* with a probability of 90% and in a place *failed* with a probability of 10%. This can be useful if LAMPS is used for planning. But LAMPS can also be used for documentation of past scenario courses. The result of a non-deterministic action is then used as the only place resulting from the action. In this way, the description of a complete scenario course results in a graph which can be compared to an optimal course of actions via graph-based distance-metrics.

## 5 CONCLUSION

In this paper, we gave an introduction to the ITSimBw simulation environment. Its main design paradigms like strict agent orientation, communication focus, and scalability were illustrated and the underlying technical architecture FTA was explained. Moreover, ITSimBw supports modern XML-based interfaces for cooperation with other systems. Due to the fact that all system components are modeled as autonomous concurrently running agents, the whole paradigm is inherently parallel and thus allows high performance execution on modern PC cluster architectures.

Furthermore, the consequential employment of agent-based thinking lead to the development of LAMPS, a uniform graphical way of describing behavioral programs for individual agents as well as complex scenarios involving a possibly large number of participating entities.

An important area for further research and development is the capability of LAMPS to record events. This feature enables the creation of a scenario data-base containing mission graphs from simulation runs as well as real-world maneuvers or even actual military missions. This data-base can then form the core component of a decision support system for the military commander. Like a chess player comparing a current board position with memorized games to determine the next move, a graph-metric can be used to liken the LAMPS graph of an ongoing mission with those in the data-base. We thus envisage LAMPS to be a core factor for the application of data-mining techniques in mission evaluation.

## ACKNOWLEDGEMENTS

The work on ITSimBw is done under contract for the department A5 of the IT-office of the German armed forces (IT-AmtBw). At this institution, the project is overseen by Captain T. Doll, whose valuable contributions are gratefully acknowledged.

## REFERENCES

Bagrodia, R. L. and W.-T. Liao. 1994. Maisie: A Language for the Design of Efficient Discrete-Event Simulations. *IEEE Transactions on Software Engineering* 20 (4):225--238

Beyer, U. and Smieja, F. 1994. Janus: A society of agents. In *GMD Report # 840*. GMD.

Bitinas, E. 2002. Pythagoras: The newest member of the Project Albert family. *Computing Advances in Military OR – WG 31. 70th Military Operations Research Society Symposium*, Ft. Levenworth, June 18–20.

Bryan, O. F. Jr. 1989. Modsim II - an object-oriented simulation language for sequential and parallel processors. In *Proceedings of the 1989 Winter Simulation Conference*, eds. E. A. MacNair, K. J. Musselman, and P. Heidelberger, pages 122—127. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Healy, K. J. and R. A. Kilgore. 1997. Silk : A Java-based Process Simulation Language. In *Proceedings of the 1997 Winter Simulation Conference*, eds. S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, pages 475—482. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Jensen, K. 1992. *Coloured Petri Nets: Basic concepts*, analysis methods and practical use. Volume 1: Basic Concepts. Monographs in Theoretical Computer Science. Berlin: Springer.

L'Ecuyer, P. and E. Buist. 2005. Simulation in Java with SSJ. In *Proceedings of the 2005 Winter Simulation Conference*, eds. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, pages 611-620. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Rice, S.V., A. Marjanski, H. M. Markowitz, and S. M. Bailey. 2005. The Simscript III Programming Language for Modular Object-Oriented Simulation. In *Proceedings of the 2005 Winter Simulation Conference*, eds.

M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, pp.621-630. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Richter, G. 1999. Flip-tick architecture: A cycle-oriented architecture for distributed problem solving. In *GMD Resport # 19*. GMD.

Stephen, R. T. , Anderson, M. A., and Lauren, M. K. 2002. MANA Map Aware Non-uniform Automata Version 2.0 User's Manual.

Teo, Y., S. Tay, and K. Kong. 1998. Structured Parallel Simulation Modeling and Programming. In *Proceedings of the 31st Annual Simulation Symposium*, pp. 135—142. Boston: IEEE Computer Society Press

Weiss, G., ed. 1999. *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge MA.

## AUTHOR BIOGRAPHIES

**PHILIPP HÜGELMEYER** received his Master Degree in Computational Linguistics and Artificial Intelligence from the University of Osnabrück, Germany, in 2002. After graduation, he worked as a research associate at the University of Osnabrück. In 2005, he joined the department ART at the Fraunhofer Institute for Autonomous Intelligent Systems in Sankt Augustin. He is also a member of the Cognitive Science Doctorate Programme at the University of Osnabrück, where he is working on his PhD thesis about collective choice in multi-agent systems. His research interests include Decision Theory, Multi-Agent Systems, Game Theory and Simulation. His e-mail address is <philipp.huegelmeyer@ais.fraunhofer.de>

**TIMO STEFFENS** graduated from the University of Osnabrück, Germany in Computational Linguistics and Artificial Intelligence in 2002. He is a research assistant at the Fraunhofer Institute Autonomous Intelligent Systems and is currently working towards his PhD in the areas of case-based reasoning and multi-agent systems which are also his main research interests. His e-mail address is <tsteffens@ais.fraunhofer.de> and his web address is <http://www.ais.fraunhofer.de/~tsteffens>.

**THOMAS ZÖLLER** currently works as a research associate at the department ART of the Fraunhofer Institute for Autonomous Intelligent Systems (AIS) in Sankt-Augustin, Germany. He has studied Computer Science together with mathematics as subsidiary subject at the University of Bonn, from which he received Diploma and doctoral grades in Computer Science (MS / PhD equivalents) in 1999 and 2005 respectively. His research interests include Pattern Recognition, Data Analysis, and Multi-Agent Simulation. His e-mail address is <thomas.zoeller@ais.fraunhofer.de>.