Integration of a Hierarchical Decision Making Framework into the Multi-Agent Simulation System LAMPSys

Christoph Mies and Robert Mertens

Fraunhofer Institute Intelligent Analysis and Information Systems IAIS
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany
{christoph.mies,robert.mertens}@iais.fraunhofer.de

Abstract. LAMPSys is a Petri net based platform for simulating multi-agent scenarios. Throughout the last years, it has been used to successfully simulate complex scenarios in a number of different domains. However, in earlier versions of LAMPSys, agent behaviors had to be hard-coded. This fact makes modeling scenarios that involve complex agent behavior a rather tedious endeavor. This paper introduces an approach that integrates the concept of leader agents with LAMPSys' modeling language. Leader agents encapsulate decision making for a group of related agents in one central agent - thus modeling a decision structure that is often found in hierarchical organizations such as most business companies or the military. This strategy does reduce modeling effort by decomposing the overall task into less complex subtasks. It also models many decision structures in a highly realistic manner.

1 Introduction

In this paper we extend the existing agent-based simulation system LAMPSys with a framework for hierarchical decision making. LAMPSys is designed to simulate scenarios for decision support. The system itself is highly customizable and able to simulate scenarios from arbitrary domains. In order to be applied to different application areas, domain-dependent so-called functional agents have to be modeled. These agents are building blocks for the scenarios. In a traffic scenario functional agents would be motor-bikes, cars, pedestrians etc. as well as infrastructure elements such as road signs or traffic lights. Thus, a society of agents defines a given scenario.

The primary goal of LAMPSys is to work as decision support system. This goal is reached by simulating a number of candidate decisions in different scenarios. During these simulations, a lot of additional decisions have to be made autonomously. The simulation results help the user to select the decisions that on average performed best during the simulations. The required enormous number of simulation runs is usually performed on a cluster of computers. However, autonomous decision making is vital for a significant number of simulations.

The introduced framework is based on the concept of leader agents. These agents make decisions for a group of assigned subordinate agents. We follow

the general definitions of agents provided in [1, chapter 2] and [2, chapter 1]. A negotiation with agents on the same hierarchical level is not yet integrated[1]. Since most business companies and the military are structured hierarchically, this purely hierarchical approach does not only reduce the modeling effort but also models the decision structures of many organizations in a realistic way.

The remainder of this paper is organized as follows: section 2 introduces the agent-based simulation system LAMPSys as well as its graph-based modeling language LAMPS. The next section gives an insight on how scenarios can be modeled with LAMPSys. Section 4 introduces the concept of leader agents and their command hierarchy. This structure naturally enables a decomposition of overall tasks into less complex tasks. At each level, the number of available actions for the leader agent decreases and a more efficient decision making becomes possible. Then the intended modeling according to the sense-think-act approach is discussed. Section 5 concludes the paper with a final discussion and gives an outlook on future work.

2 LAMPSys

In LAMPSys, processes and agents are modeled with a special modeling language called LAMPS (Language for the Agent-based Modeling of Processes and Scenarios). It extends the concept of colored and hierarchical Petri nets [4, 5]. The former introduced the concept of typed tokens, the latter structured so-called places (states) and actions hierarchically. Additionally, LAMPS introduces the concept of agents. The behavior of agents is modeled with LAMPS fragments. Each is inherently parallel due to its core of high-level Petri nets. Agents live in parallel, and each agent can execute several parallel actions. All actions whose conditions are true in a given cycle are executed. This is also the main difference to flow-charts, because there can be several token per place, and several places can be filled simultaneously. All basic constructs can be recursively encapsulated. For example, places can be combined to so-called *super-places*. An action can be recursively defined as a LAMPS fragment, as long as the interface (i. e. the incoming and outgoing places) of the action and the fragment are identical. This way, a process can be modeled and viewed on different levels of detail. Agents can also be recursively encapsulated. A group of agents can be aggregated into one agent (say, a group of computers into a cluster). LAMPSys is a simulation system that is able to execute LAMPS graphs. It is based on the Flip-Tick-Architecture [6] and uses the concept of a society of agents. The whole environment is modeled in terms of agents and the whole interaction is based on effects of actions executed by agents. 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. The basic unit of execution is called a cycle. During one cycle, the agent reads its messages and triggers the appropriate actions, which might

[1] According to [2, chapter 2] and [3], negotiation is not required for multi-agent environments.

consist of writing messages to other agent. This is called the *tick* of an agent. During each tick a certain amount of time dt is simulated. Actions that last longer than dt are triggered during several consecutive ticks and perform only a part of the overall action during each tick. Thus, the time resolution of the simulation can vary from cycle to cycle. This is particularly valuable for increasing the time resolution in the computation of dynamic equations for fast moving objects. The whole simulation is performed in cycles. Each agent manipulates data during its *ticking*. After all agents have been ticked, the system *flips* them. During this *flipping*, the agents publish their data. Thus, at cycle t the agents read data produced in cycle $t-1$. This process allows the parallel execution of agent actions without the danger of racing conditions. Please refer to [7, 8] for more details about LAMPSys.

3 Modeling Scenarios With LAMPSys

In LAMPSys everything that can change in a scenario is modeled in terms of agents, i. e. a society of agents defines a scenario. Since LAMPSys provides the framework only, all domain specific, i. e. functional, agents have to be modeled before interesting tasks can be simulated. Currently, there exist several domains where LAMPSys is used for simulation [9, 10]. In this paper, the domain of medical evacuation (see [7]) is analyzed in terms of demands of the simulation system. The domain as well as some requirements for the agents involved are explained in section 3.1. A taxonomy of different types of agents is given in section 3.2. The section introduces basic, auxiliary and decision maker agents. Afterward the approach for functional, i. e. domain dependent, communication, which maintains the interoperability of LAMPSys, is discussed. Finally the sense-think-act modeling approach of basic agents is introduced.

3.1 Domain of Medical Evacuation

The domain of medical evacuation is a logistics environment. At the beginning of the simulation some agents representing injured people are placed at random locations. A dispatcher agent can command some emergency transporters. Its task is to let these transporters rescue the injured by transporting them into hospitals. The transporters can either be helicopters or vans. All of them can have different average speeds and transport capacities. The most important difference between helicopters and vans is that the latter move on roads whereas the former ones can fly the direct way. These two types of transporters additionally have different activation and repairing times. The injured have different degrees of injuries, ranging from light to severely wounded. Additionally, they have a certain amount of health points that are reduced over time until the agent perishes. The number of health points as well as the reduction rate depends on the degree of the injury. When the injured have been delivered to the hospital they can be healed, i. e. all health points are restored. The dispatcher's primary target is the survival of all injured. The involved agents have to be capable of

moving along roads (e. g. the vans), moving directly to target locations (e. g. the helicopters), communicating and causing effects (e. g. the hospitals must be able to heal the wounded). Additionally, they must be able to make intelligent decisions (dispatcher). The dispatcher agent can be considered as a leader agent (see section 4) since it commands other agents in order to reach its goal. In its basic version the domain of medical evacuation is fully-observable, deterministic and nearly markovian. The only violation of the markov assumption [1, chapter 15] is the stochastic occurrence of wounded. This can easily be solved by generating new decisions when new wounded occur. Additionally, no neutral or opponent agents exist. The wounded are considered as allied agents. Nevertheless, this domain can be extended such that it becomes partial-observable (e. g. if the wounded have to be actively searched for) or non-deterministic (e. g. if inter-agent communication is disrupted).

3.2 Types of Agents

A taxonomy of different types of agents is useful in order to give names to different kinds of agents. The following agent types exist:

- *Basic agents*: These are the most important agents from the functional point of view, because most parts of the domain are modeled by basic agents. They are able to execute orders (see section 3.3) and can perform non-trivial tasks like moving on roads or transporting wounded. Examples are helicopters, vans, hospitals etc.
- *Auxiliary agents*: They are also needed to model a domain. The difference to basic agents is that they perform very simple tasks that merely require decision making and cannot execute orders. Thus, they are relatively simple to model. In the medical evacuation domain, radio-equipment agents or channel-agents (see section 3.4) are auxiliary agents.
- *Decision maker agents*: These agents make complex decisions and their reasoning-ability is very important for the significance of the whole simulation. An example is the dispatcher agent which commands other agents. This agent type is extended to a framework in section 4. Thus, decision maker agents are substituted by leader agents.

This informal taxonomy of agent types simplifies the discussion in the remainder of this paper. Note that LAMPSys is not aware of that taxonomy and treats all agents in the same way.

3.3 Functional Communication

In order to decouple functional (domain-specific messages) and technical (data) communication, all functional communication is encoded in a domain-specific functional language. For the medical evacuation domain, BML [11] is used. BML is based on a formal grammar. It is designed such that it can easily be parsed by computers as well as been understood by humans. BML comprises three

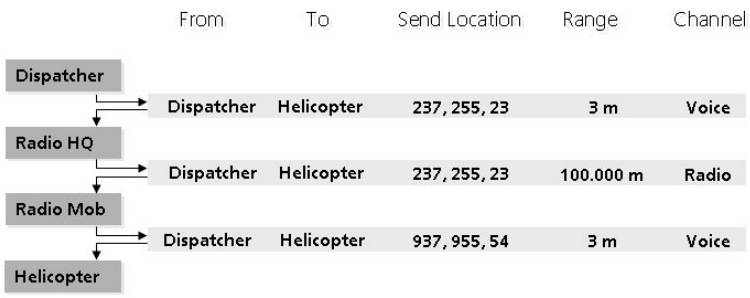

Fig. 1. Exemplary communication chain: The dispatcher wants to send a message to the helicopter which cannot hear it directly due to the small range of the voice-channel. But the dispatcher's radio equipment senses the message and sends it to the radio-channel. The helicopter's mobile radio equipment receives the message from there and sends it back to the voice-channel. Finally, the helicopter agent can receive it from there.

types of statements. It is able to express orders for other agents and reports about interesting events. Additionally it is also able to express effects. The BML statements are then compiled to conceptual graphs [12]. The main advantage is the interoperability of conceptual graphs. Many languages can be compiled to conceptual graphs and vice versa. Thus, the conceptual graph serves as main knowledge representation for many different domains that can all be simulated with LAMPSys. The signatures of services or procedures do not need to be changed. Additionally, it is straight-forward to perform reasoning on conceptual graphs [12]. The following section shows how simple reports being in an agent's short-term-memory can be extended by other reports in order to enrich the contained knowledge.

3.4 Sense, Think, Act for Basic Agents

The basic agents are modeled according to the sense-think-act approach. This sequential model requires that each agent reads and evaluates its sensors at the beginning of each cycle. Afterward, each agent makes its decisions and finally uses its effectors to execute actions in the environment. Many agent architectures are proposed in the literature, e. g. [1, ch. 2.4] and [2, ch. 1.3]. The basic agents can sense sounds, optical input and effects that are caused by other agents. The sound sensor is modeled very detailed in a physical manner. An auxiliary agent represents the voice-channel, from which the agent can sense the words that are spoken in its direct proximity. The radio equipments are also represented by auxiliary agents, that scan the voice sensor and transmit everything they hear to the radio-channel that is used for radio communication. This channel is again scanned by all radio equipments and then transmitted to the voice-channel again. Thus, the radio equipments work as translators between the voice- and the radio-channel. An example of a communication chain is depicted in figure 1. Arbitrary other communication channels, e. g. IT systems, can also be modeled using the

same concept. Due to the autonomy of agents, every agent decides for itself how it reacts to a message. The same principle is pursued by the sensing of effects. All caused effects are communicated via a certain auxiliary agent, the effect-channel agent. Every basic agent scans this channel and decides whether it wants to take the effect or not. This is necessary since no agent is allowed to change the state of other agents. Thus, if one agent heals a wounded, the wounded decides if it becomes healthy or not. Taking this approach, causing and taking effects is modeled as communication. Arbitrary chains of effects can be built similarly to the case of the communication chain. If, for example, a helicopter's engine breaks down one effect could be that the helicopter crashes which again could cause the pilot to die. Thus, the primary effect of a broken engine triggers the secondary effect, i. e. the crashing helicopter. This event causes the pilot to die, the third effect. A slightly different approach is taken for the optical sensors, the visibility. Due to performance reasons, the visibility is calculated at a central agent. The technical visibility agent holds a voxel representation of the whole scenario. All basic agents participate in a publisher-subscriber communication. They publish all data that is needed for the visibility calculation, especially their voxel-shape as well as their position, and subscribe for the visibility messages calculated by the visibility agent. Basically, every agent receives a location-message for each other agent it can currently see. The current decision making performance of basic agents is quite poor. After reading the sensory input and fusing the new knowledge with the current one. This knowledge is composed of orders and reports. The second step concerns the execution of BML statements that are included in the agents' world model. Note that all basic agents are able to execute BML orders (see section 3.2). As mentioned above, conceptual graphs are used to represent the basic knowledge. An example for knowledge is the BML order `move on road dispatcher van to (100.0 200.0 0.0) reply with id1`. Its meaning is that the dispatcher-agent `dispatcher` orders the medical transporter `van` to move to the coordinates `(100.0, 200.0, 0.0)` and to send a reply-message with the label `id1` when the order's execution is finished. Let the answer of the agent `van` be `reply van dispatcher id1 successful` meaning that agent `van` has fulfilled its order. Then the agent `dispatcher` would merge the answer with its own order resulting in a larger conceptual graph. With this simple merge concept, arbitrary graphs with a huge amount of contextual information can be built. Thus, each graph represents information that belongs to the same context. All important knowledge concerning this context can be extracted from one single graph. In order to model the restricted mental capacity of basic agents only a certain number of graphs can be contained in the STM at the same time. If the maximal number of graphs in the STM is exceeded the oldest ones are discarded, i. e. forgotten. Additionally, all BML statements that have not been updated for more than a certain time are removed. Note that each merge with other graphs causes the graph to get a new time stamp and count as new graph. One advantage of the merged conceptual graphs can be seen here. After receiving `van`'s reply message, `dispatcher` automatically refreshes the knowledge about the given order. Thus, the whole context of the reply message is

refreshed. The second phase of the current decision making of basic agents is the execution of BML orders. This execution can be compared with reactive skills of a layered robot control architecture as defined in [13, chapter 8]. Currently, any agent executes any order that it receives. If several contradicting orders occur, the last received is executed. Note that by now there exists no model for the identification of contradicting BML orders. Executing a BML order works as follows. A certain LAMPS graph `execute_bml_order` reads the orders. According to the type of order, e. g. `move on road`, the graph activates other LAMPS graphs by changing the corresponding attributes of the agent. This newly activated LAMPS graph, e. g. `move_on_road`, performs the action itself. Additionally `execute_bml_order` monitors the execution of the activated graph, e. g. `move_on_road`. If this LAMPS graphs stops, `execute_bml_order` generates a reply message, e. g. `reply van dispatcher id1 successful`. The activated LAMPS graph is modeled by hand, i. e. it is hard-coded. This is tractable for "simple" graphs like moving on a road, but gets very tedious and complicated if complex graphs like the dispatching of transporters are to be modeled. Afterward, the act phase begins. Here, the desired effects are caused. Basically, there are two kinds of effects of actions. On the one hand, effects may only cause the change of internal attributes. For example, if an agent is moving, it basically changes the internal attribute position. Note that a position change is published to the visibility agent as mentioned above. But in principle, a change of an internal attribute is not observable for other agents. On the other hand, an effect may involve communication. This approach is taken by speaking some orders for other agents into the voice-channel agent or by causing effects (e. g. healing) via the effect-channel agent.

4 Leader Agents

A *leader agent* commands some subordinate agents in order to reach its goal. The logistic dispatchers can be considered as leader agents although they do not yet work as described in this section. Basically, a leader agent calculates a plan for all its subordinates and monitors the plan's execution. Since a leader agent always simulates the "intelligence" of an agent, it must be plugged into a basic agent (see section 3.2). Metaphorically speaking, the logical leader agent, which models the intelligence, must be hosted in a basic agent that models the physics. Section 4.1 introduces the command hierarchy. This hierarchy defines the relation between leader agents and subordinate agents. The following three sections describe the sense-think-act modeling approach for leader agents. Especially section 4.3 is important since decision making is the core process of a leader agent.

4.1 Command Hierarchy

A precondition for leader agents is a command hierarchy which models which agent is commanded by which leader. Note that a recursive assignment is possible, i. e. a leader agent can command subordinate leader agents. This chain

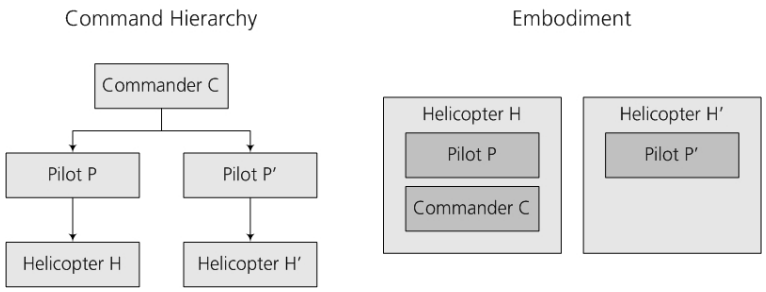

Fig. 2. The left part shows the command hierarchy: The leader agent C commands the elementary leader agents P and P' which again command the basic agents H and H', respectively. The embodiment of the leader agents is shown on the right: H embodies P as well as C and H' embodies P'.

ends with the *elementary leader agents* that command exactly one basic agent. The hierarchy must be modeled such that every agent is assigned to at most one leader agent in order to avoid competing orders. Note that it is not acceptable to simply execute the last given orders in case of conflicts if they come from different leader agents. Consider a leader agent C that commands two helicopters H and H'. Each of these helicopters consists of the helicopter itself (represented as basic agent) as well as the corresponding elementary leader agents P and P', respectively. Then each helicopter agent is commanded by its corresponding elementary leader agent which is itself commanded by C. Thus, C cannot command the basic helicopter agents directly. Note that C has to be embodied in a physical basic agent. It can be embodied in one of the helicopter agents denoted by H. In this case H embodies two leader agents (one commanding the other one) but is directly commanded by one, only. Figure 2 shows this exemplary situation. The task of the elementary leader agent is to command its embodying basic agent. The main advantage of sourcing out the basic agent's decision making to the elementary leader is a more modular modeling. The intelligence is encapsulated in the leader whereas the capabilities of, for example, a helicopter are modeled in the basic agent. Thus, the intelligence can be changed by simply changing the leader. It is also important that the command hierarchy is flexible during the simulation. Thus, it is possible that one agent moves from the command of one leader to the command of another one. This is especially important, because a basic agent can vanish during the simulation, e. g. a helicopter can crash and the leader agent C from the above example can die. Then a substitution must command the remaining helicopters. Functionally speaking, the commander changes, but technically speaking the command hierarchy changes and all assigned agents must be assigned to another leader agent. By this hierarchical approach, the overall complex task can easily be decomposed into partial tasks. These are then less complex and can again be decomposed into smaller ones. Finally, the elementary leader agent gives orders that are simple enough to be executable by its basic agent directly. Concerning the medical evacuation do-

main, the overall task is to evacuate all wounded. If n hospitals exists and each of them has its own vans and helicopters, the overall leader agent can command n leader agents that again command the elementary leader agents of the transporters. The embodiment of leader agents simplifies the modeling complexity since these agents serve as a "intelligence plug-in" for basic agents. The latter ones carry out tasks that require less tactical requirements whereas the leader agents' behavior has to be tuned in order to get suitable results. Thus, the basic agents need not to be adapted for every scenario.

4.2 Sense: World Model

The world model is the basis for all decisions that a leader agent can make. In each cycle, the world model is updated by sensory input. Currently, the sensory input consists of BML statements for facts. Additionally, the ¨felt¨ world can be represented by potential fields. Basically, friendly and hostile areas can be assigned peaks and troughs, respectively, in a potential field. This knowledge can, for example, be used for routing along roads. Another important point is the representation of the leader agents' target. This basically consists of the formulation of one primary target that has to be reached under certain constraints (secondary targets). In the domain of medical evacuation the target could be to rescue all wounded under the constraints of using as less fuel as possible, or of being as fast as possible. Planning requires models of the environment. In LAMPSys all non-static entities are represented as agents. Thus, only models about other agents are needed. From the leader agent's point of view the assigned agents can execute certain BML statements. Thus, they can be assigned *roles* describing exactly one capability. For example, an agent that can move along streets is able to execute the move-on-road-order and thus filling the move-on-road-role. Agents can fill different roles at the same time as long as these are not mutual exclusive. A van can transport a wounded while moving on roads. But it cannot load a wounded while driving. Two kinds of agents have to be distinguished: Models for subordinate and non-subordinate agents. The former ones are assumed to behave exactly as they are ordered by the leader agent itself. Thus, it knows which actions the assigned agents will execute in the future. The actions executed by the assigned agents can be considered as actions executed by the leader agent itself. The second case is more complex since the leader agent cannot know how these agents behave. The non-subordinate agents can again be clustered into three groups: Allied, neutral and opponent agents. The former ones are assumed to behave cooperatively. It could even be possible that these agents communicate their desired behavior in order to simplify the collaboration of the alliance. The neutral ones will not communicate their goals and desires. But they will not try to actively disturb the leader agents task. Finally, the opponent agents will try to harm the leader agent as well as its subordinates and actively try to prevent them from reaching their goals.

4.3 Think: Decision Making

The core of the leader agent concept is the decision making framework. The proposed framework should not be restricted to certain kinds of methods. Thus, a SOA approach might be useful. Currently, two types of dispatchers are available for the domain of medical evacuation. One is hard-coded and follows a greedy approach. The other one is optimized by a genetic algorithm (see [7]). However, the hierarchical task decomposition opens possibilities for integrating more sophisticated decision making methods. The fact that on each level of the hierarchy leader agents can only choose from a limited set of actions prunes search space considerably. This makes it possible to simulate leader agents' decision making with computationally expensive methods. One such method is automated planning. Planning is a suitable method, because the wide variety of available techniques ranging from domain-independent to heuristically guided systems opens many possibilities. Additionally, the supported range of target domains is wide. Classical planning techniques are available for restricted domains and there exist techniques that are able to cope with very demanding (e. g. highly uncertain) domains. An important question that arises when planning is applied concerns plan monitoring [14]. Its task is to decide when the currently executed plan should be stopped and a new one should be generated. If the plan is stopped because of execution flaws (e. g. due to unforeseen changes in the environment resulting from wrong or imprecise models) another question arises: Re-plan from scratch or repair the current plan? Although it has been proved by [15] that repairing is at least as hard as re-planning in general, many systems (e.g. a variant of LPG [16]) state that in certain domains repairing pays off in terms of computation time and plan similarity. The latter is also important if the agents have already committed themselves to certain tasks since repaired plans are often similar to the original ones. The monitoring can result in three outcomes: The first one is *Re-plan*, which initiates a re-plan from scratch. Secondly, a *Plan-repair* can be initiated. Another possible outcome is *Continue* denoting that the current plan is further executed. Further execution means that potentially new orders are given. Note that after a *Re-plan* or *Plan-repair* new orders might also be given to subordinate agents. The plan monitoring can easily be integrated into LAMPSys since the monitoring procedure is automatically called in each simulation cycle as part of executing the agent's behavior. In the current version, a *Re-plan* is initiated in case of execution flaws. With this approach, complex decision making is performed deliberatively and the same LAMPS graphs can be used in several domains. The planning domain as well as the planning problem generation have to be adapted for each application area. Note that plan monitoring has to consider incoming orders since these might require re-planning. The decision making of basic agents as described in section 3.4 remains unaffected except that only orders given by its elementary leader agent are executed.

4.4 Act: Give Orders

After a plan has been generated, the assigned subordinate agents have to execute it. Therefore, the plan has to be converted to BML commands for each assigned

agent. These have to be transmitted to the corresponding agent. If a leader agent L gives orders to a subordinate agent S three cases exist. Firstly, S can be L's embodying basic agent. Then S can simply execute the orders. Secondly, S can be a leader agent embodied on the same basic agent as L is. Consequently, S can take the orders and execute them. Finally, S can be an agent that is not embodied on the same basic agent as L is. Then L's embodying agent has to communicate the order to S's embodying agent. In all three cases the BML commands are delivered from L to S using the appropriate communication mechanism (direct or via intermediate agents).

5 Conclusion and Future Work

In this paper, we introduced the concept of leader agents. We have integrated leader agents as a framework for automated decision making into the agent-based simulation system LAMPSys. The framework's most important characteristic is the hierarchical structure in which orders are given and executed. It supports a hierarchical task decomposition reducing the number of available actions at each level. Thus, a more efficient decision making is possible. Note that leader agents are not necessarily used to model human decision makers. It is our primary goal to provide decision support by performing many simulations with (nearly) optimal decisions. Currently, it is not intended to model social aspects of human communities. The concept of basic agents is still vital for LAMPSys, because it reduces the modeling effort in comparison to e. g. a stochastic action outcome generator. The reason therefore is that many events can occur during the simulation. Having modeled the agents' responses to these events, they can be used in all types of scenarios. This is an advantage, because having n stochastic components for n types of scenarios would require to adapt each component to all events. For example, weather can influence the speed of helicopters. This influence might be restricted locally. Thus, every helicopter has to calculate the weather's impact on its speed, which can perfectly be modeled with basic agents.

Since only the framework has been introduced, there remains a considerable amount of future work. First of all, we want to couple LAMPSys with several existing planning systems: A metric PDDL planning system [17], a decision theoretic (DT) planner, a hierarchical task network (HTN) planning system and a control knowledge based system. PDDL is interesting, because it is domain-independent and there exist many planning systems. DT planning systems are designed to cope with uncertainty and non-determinism. The last two planning approaches are domain-dependent and can be guided heuristically by the user. Most proposed techniques of [18] are represented by these planning approaches. It is important to evaluate the strengths and weaknesses of the systems. Then we can try to find rules or heuristics that identify situations in which certain systems are superior to others. Another important method is the automatic generation of domain and problem definitions (as defined in [18] for all approaches mentioned above) from the agent's knowledge. Thus, the planning system of each leader agent works on a context-dependent abstract world model. We expect

that this automatic search space pruning leads to a significant improvement of the planning times. Since planning time is the limiting factor, we expect better plans in terms of solution quality to be achieved. A first approach for developing this meta-reasoning system might be to pursue the idea of [19]. Additionally, we want to adapt the idea of the Pandemonium [20]. Basically it should be possible to apply several planning systems at the same time and decide from situation to situation which system generates the plan.

References

1. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach (2nd Edition). Series in Artificial Intelligence. Prentice Hall (2002)
2. Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press, Berlin (1999)
3. Klügl, F.: Multiagentensimulation. Informatik Spektrum **29**(6) (2006) 421–417
4. Jensen, K.: Coloured Petri Nets: Basic concepts, analysis methods and practical use. Volume 1 of Monographs in Theoretical Computer Science. Springer (1992)
5. Vojnar, T.: Various kinds of petri nets in simulation and modelling. In Stefan, J., ed.: Proceedings of MOSIS '97. Volume 1. (1997) 227–232
6. Richter, G.: Flip-tick architecture: A cycle-oriented architecture for distributed problem solving. Technical report, GMD, Sankt Augustin, Germany (1999)
7. Mies, C., Wolters, B., Steffens, T.: LAMPSys - An experiment-platform for the optimization of processes and agent-behaviors. KI Zeitschrift (2) (2009) 42–45
8. Steffens, T., Zoeller, T., Huegelmeyer, P.: Agent-based modeling of processes and scenarios with high-level petri nets. Technical report, Fraunhofer IAIS (July 2006)
9. Huegelmeyer, P., Steffens, T., Zoeller, T.: Specifying and simulating modern warfare scenarios with ITSimBw. In Perrone, L.F., Wieland, F.P., Liu, J., Nicol, B.G.L.D.M., Fujimoto, R.M., eds.: Proceedings of WinterSim 2006, ACM (2006)
10. Usov, A., Beyel, C.: Simulating interdependent critical infrastructures with simcip. European CIIP Newsletter **4**(3) (2008)
11. Huegelmeyer, P., Schade, U., Zoeller, T.: Application of BML to Inter-Agent Communication in the ITSimBw Simulation Environment. In Biller, B., Hsieh, M.H., Shortle, J., Tew, J.D., Barton, R.R., eds.: Proceedings of WinterSim 2007. (2008)
12. Sowa, J.F.: Conceptual graphs. In: Handbook of Knowledge Representation, Elsevier (2008) 213–237
13. Siciliano, B., Khatib, O.: Springer Handbook of Robotics. Springer (2008)
14. Micalizio, R.: On-line monitoring of plan execution: A distributed approach. PhD thesis, Universit di Torino, Torino, Italy (2007)
15. Nebel, B., Koehler, J.: Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis. Technical Report RR-93-33, DFKI GmbH (1993)
16. Fox, M., Gerevini, A., Long, D., Serina, I.: Plan Stability: Replanning versus Plan Repair. In: ICAPS 2006, AAAI Press (2006)
17. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Artificial Intelligence Research **20** (2003) 51–124
18. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Praxis. Morgan Kaufmann Publishers, Boston, MA, USA (May 2004)
19. Hartanto, R., Hertzberg, J.: Fusing DL Reasoning with HTN Planning. In: KI 2008: Advances in Artificial Intelligence. LNCS, Springer (September 2008) 62–69
20. Selfridge, O.: Pandemonium: A paradigm for learning. In: Mechanisation of Thought Processes, Her Majesty's Stationary Office (December 1959) 511–531