Towards Pattern-Oriented Design of Agent-based Simulation Models

Franziska Klügl and Lars Karlsson

School of Science and Technology
Örebro University, Örebro, Sweden
{*franziska.klugl, lars.karlsson*}*@oru.se*

Abstract. The formalization and use of experiences in good model design would make an important contribution to increasing the efficiency of modeling as well as to supporting the knowledge transfer from experienced modelers to modeling novices. We propose to address this problem by providing a set of model design patterns inspired by patterns in Software Engineering for capturing the reusable essence of a solution to specific partial modeling problem. This contribution provides a first step formulating the vision and indicating how patterns and which types of patterns can play a role in agent-based model design.

1 Introduction

Since the seminal studies of Willemain [1] on how modeling experts actually build models, it is known that modeling and simulation processes are mostly guided by expert intuition.

Despite of its obviously intuitive way of modeling, one must admit that the development of a concept model and the design of an agent-based simulation from that is a very challenging task – especially for starting modelers. Finding the right abstractions, focussing on only the relevant relations and modeling on the right level of detail are issues that hardly an experienced modeler can handle well without following a painful trial and error procedure. For novices in modeling and simulation – with or without (software) engineering education –, this phase is particularly difficult and frustrating as they find no guidelines and heuristics for these early, yet decisive phases of a simulation study. Engineering processes advising the different steps in a systematic simulation study are not helpful in these early phases of modeling. Unfortunately, the greatest advantage of multi-agent simulation - its high degree of freedom in model design - turns out to be a curse as there are no constraints and restrictions on what can be formulated in a model. Therefore, especially novices may feel lost in the wide field of alternatives.

The problem of guiding the development of a model concept can hardly be solved from a general methodological point of view as it involves too much domain-specific knowledge. However, we can formulate iterative strategies for model development in general and support the model design on a more technical

level. This contribution is part of larger vision of advancing the methodological status of agent-based simulation towards systematic engineering starting from a model concept to the analysis of the simulation results.

Our idea for supporting technical model design bases on the well-known concept of software design patterns that – since their first introduction by Gamma et al., [2] – became a success story in object-oriented software design. Design patterns are meanwhile used in undergraduate courses in computer science for teaching proper software design. They are acknowledged as an efficient vehicle for transporting knowledge and experiences about how to solve particular problems. Therefore the idea of transferring the concept to modeling and simulation seems to be promising - especially for the case of agent-based simulation. While patterns address good simulation model design, components for modeling as proposed in [3] are provided as implemented, configurable building blocks. Both concepts address model reuse, but they are clearly different.

The next section gives the state of art in pattern-oriented design of agent-based software, but also examines whether there are similar concepts of patterns in simulation and modeling. This is followed by a section on patterns for agent-based simulation tackling general representation and different categories and examples. The contribution ends with a short conclusion and indication of future work.

2 Pattern-Oriented Modeling

About ten years ago, software patterns became popular [2]. The basic idea behind those was to capture experience about good design for particular problems in a uniform and schematic way for supporting its reuse. A pattern can be seen therefore as a recurrent problem and its solution on conceptual, architectural or design level. Thus, the design of software should be ideally systemized to analysis and identification of basic problems item to solve, followed by the selection of appropriate patterns and instantiation and adaption to the particular case.

2.1 Patterns in the Agent World

Due to the obvious usefulness of patterns for software design, it is not surprising that soon patterns for agent-based software – not simulation – have been discussed and defined. Weiss [4] gives a good introduction to how to develop and use patterns in the development of agent-based software. Oluyomi and others [5] survey and analyze existing patterns for agent-oriented software and classify many of them. This work is accompanied by second publication from the same group [6]. There, the authors systematically deal with description templates appropriate for multi-agent systems with the aim of making different specifications of similar or equal patterns more comparable. In contrast to this, Sauvage [7] tackles more a meta-level view on patterns introducing the categories of anti pattern, metaphors and meta pattern.

Kendall and others [8] give patterns for several layers of an agent society such as sensory layer, action layer, collaboration and mobility layers. The patterns range from *The Layered Agent Pattern* to patterns for different types of agents with different abilities with respect to agent capabilities as sensory-level patterns; *intention pattern* or *prioritizer pattern* as action-level patterns; agent system patterns ranging from *conversation pattern* to *facilitator* and *proxy pattern* and finally *clone* or *remote configurator* as examples for mobility patterns. They hereby use a uniform schema consisting of problem, forces, solution, variations and known uses.

Aridor and Lange [9] deal with patterns for mobile agents classified as task pattern, like *Master-Slave*, as interaction pattern, like *Meeting* or *Messenger* and as travel pattern, e.g. *itinerary* for agent routing. More patterns for itineraries can be found in the work of Chacon et al., [10] – whereas they mean more behavioral patterns – like the *Sentinel Agent Behavior Pattern* for persistent monitoring, than a routing pattern as in the previous case. Social patterns that help to design interactions between agents can be found in a publication by Do et al., [11]. They categorize social patterns into peer patterns such as the *booking pattern* or the *call-for-proposal* pattern and into mediation patterns such as *monitor pattern* or *broker pattern*. Only the *booking pattern* is fully elaborated until code generation in this publication. A list of similar mediation patterns is given in a paper by Hayden et al., [12]. More recently, one can find patterns for specific application domains, mostly in the area of information agents. A very extensive list of patterns can be found in the PhD thesis of Oluyomi [13].

For agent-based simulation, these patterns have to carefully evaluated and potentially revised. The main reason lies in the general differences between agent-based software and agent-based simulation: The constraints for the design of the simulated multi-agent system are much higher as it has to credibly correspond to an original system. Whereas functional and technical requirements of a software allow more alternatives in appropriate design, the main directive for simulation model design is its validity. Hereby, structural validity refers to the correspondence between the design of the model and the real-world system [14]. Thus, technical patterns, such as yellow pages pattern are not relevant for multi-agent simulations, as long as they don't need to be part of the model due to the objective of the simulation study. Additional patterns are relevant due to the relevance of the environment and of particular feedback loops hopefully leading to some given macroscopic behavior - for example in terms of population dynamics. This is the most difficult aspect in model design and is therefore the most attractive for formalizing experiences in terms of design pattern.

2.2 "Patterns" in the Simulation World

Grimm and Railsback [15] introduced so called pattern-oriented modeling for individual- respectively agent-based simulation in ecology. They thereby refer to patterns in given data that are to be reproduced by the simulation. Based on data patterns, a systematic model development procedure is defined. This is a different form of pattern than we use it here where we are discussing design

patterns that should support the production of a well-structured model that resembles all required data patterns or stylized facts.

Similar pattern-like building blocks also have been established in mathematical, equation-based modeling, especially in biological simulation. The researcher analyzes what kind of relationship may be assumed between two or more variables and then selects the appropriate mathematical function from to a catalogue of well known formula. Finally, she or he fits the parameters of the functions to available data. These functions can be seen as basic model patterns. Examples are the exponential growth function or saturation curves. In his book on biological modeling, Haefner [16] lists a complete toolbox of functions for modeling different relationships that are usually applied in modeling and simulation in the area of population biology. Also the basic Lotka-Volterra equations of a predator prey system form elements of such a model schemata library. In [17] a pattern language for spatial process models is given; the examples are more like full model description. Unfortunately, such function patterns cannot be transferred to agent-based simulation as – in contrast to these models – agent-based models *generate* macroscopic dynamics from the agents behavior whereas the macroscopic models *describe* the relations and dynamics on the aggregate level. Nevertheless, it would be a most interesting – and most challenging – form of design support to find agent population patterns that produce the corresponding dynamics on the macroscopic level.

Also, in object-oriented simulation, pattern-based methods for developing simulators were proposed [18]. Patterns are hereby used for putting together objects or components that form the building blocks for a model. Using code templates associated with a pattern, the code for a complete simulator could be instantiated. Such a system was developed for the domain of building simulation.

3 Agent-based Simulation Model Design Patterns

Considering agent-based modeling and simulation, the actual level of patterns may not always be obvious: the technical, detailed implementation-near level and a more conceptual design level. An example for the former is the *configuration interface pattern* describing how to aggregate all start values into one interface object. Examples for the latter are patterns of *limited exponential growth* or a *food web pattern*. One may further divide model design patterns into agent architecture patterns that describe appropriate ways of conceptualizing an agent itself, patterns for agent models generating certain population dynamics, and interaction patterns for capturing dynamics among agents and between agents and their environment. Also, meta-level patterns are possible that describe how different design pattern could be combined. As with software engineering patterns, all of these may seem trivial for the experienced multi-agent simulation developer, but may be highly valuable for starters, but also for general characterization of possible agent models.

In the following we will discuss several pattern categories and patterns in more or less detail; this is not meant as a complete list but more like an indication

of what can be possible. First, we will set the general frame for approaching such a pattern by giving a schema.

3.1 Pattern Schema

Many experienced modeler implicitly use patterns of agent behavior. In an analogous way to software design patterns, model design patterns may be made explicit using some predefined scheme. Thus, before going into the details of particular model design patterns, we have to discuss a proper scheme for making the experiences explicit and thus reusable. As mentioned above, there are many suggestions for schemes. The following is clearly inspired by the original pattern language [2].

Name: Each pattern needs an memorable name for referring to the pattern in a short yet descriptive way.

Intent: What is the problem that this pattern aims at solving? What kind of pattern/relations should be reproduced by it?

Scenario: In what situations or basic model configurations, does it make sense to apply this pattern?

Description: Short description of the functionality produced by the pattern.

Dependencies: Does the pattern only make sense in combination with other patterns? Do we need a specific form of data for reasonably handling the structures?

Agent System Structure: This is the actual content of the pattern. What agents classes are involved? How do they interact? What environmental structures are part of the pattern?

Agent Behavior: Exact specification of the agent behavior pattern. This corresponds to the original "code" attribute of a pattern scheme.

Technical Issues: Sometimes additional low-level technical aspects are important. This serves to prevent artifacts coming from poor implementation.

Example: If necessary, an additional example can be given for clarifying the usage of the pattern. In what models was the pattern successfully applied?

Configuration: For evaluating the properties of the pattern in the simulation context, the existence and effects of parameters have to be discussed.

Consequence: Using this pattern may have consequences for further design decisions. Information about this should be given here. This item is different from the original one: the original "consequences" were split up into configuration and consequences.

Related Patterns: Pointer to other patterns that may compete with this pattern or have similar properties.

This is a first attempt for finding a systematics for documenting agent-based simulation model design. In the following a number of patterns are sketched and example patterns are discussed in more detail using the above introduced pattern description scheme. We have identified the following pattern categories: agent architecture patterns, agent population patterns, interaction patterns and environmental patterns.

3.2 Agent Architecture Patterns

The first category that comes to mind is agent architectures. They form basic patterns for how to organize different modules and processes within an agent. A model design pattern on the level of an agent architecture does not necessarily tackle full agent architectures, but also certain component interactions.

Agent architectures were in the focus of research from the beginning of Distributed Artificial Intelligence. Thus, one can already find a wealth of architecture suggestions in the literature with different complexities, intended applications, etc. During the last years the discussion about appropriate agent architectures has calmed down a bit. For agent-based simulation, the selection/design of an agent architecture can be difficult as the used generic architecture must be explicitly treated as a basic assumption that also has a correspondence to the original system or its influence on the results should be credibly justified.

Formulating agent architectures as patterns is quite popular in agent-oriented software design (see Section 2.1). As one can find psychological theories for some architectures, especially layered and BDI architectures, it is absolutely justified to add these full architectures to a list of agent model design patterns. But also smaller parts or architectural details can be interesting to be formulated as pattern. In the following we give an example of an agent architecture pattern that turned out useful in several applications: *Perception Memory Pattern*. It combines a particular form of knowledge representation with specific reasoning. Instead of providing a complete architecture, it may also be combined with other patterns.

Name: *Perception Memory Pattern*
Intent: The pattern shows how perception can be dealt with separately from interpretation. This increases efficiency as perception is appropriately buffered instead of allowing environmental scans whenever information is needed.
Scenario: This pattern makes sense when the agent needs information about the local environment more than once in its behavior specification. The general background is that memory space is cheaper than scanning the environment.
Description: The agent status contains a set of variables for memorizing perceptions. The first step in each update cycle consists in the agent scanning its environment and memorizing all noticed objects in the variable. A cascade of filters provides information necessary in the current situation based on the initial perception. The agent may always access the initially perceived information, instead of scanning the environment again. The pre-processed information is then used for decision making or for determining the agent's behavior.
Dependencies: This is a basic pattern that may be used in a variety of models with other diverse patterns built upon it.
Agent System Structure: The agent needs at least one variable or memory location for saving the memorized information about its surroundings: PerceptionMemory. Additional data structures contain preprocessed information
Agent Behavior: The behavior or an agent contains the following partial steps:

1. Initial Scan → PerceptionMemory
2. Filter PerceptionMemory appropriately → ProcessedPerception
3. Follow behavioral rules, use ProcessedPerception as if it would come from direct access in the environment if necessary go back to the second step.

Technical Issues: It must be secured that the basic scan happens at the appropriate point of time in an update cycle and sufficiently often.
This separation between basic environmental scan with memorizing all information that might be useful in the later context may also be used for implementing virtual parallelism where all agents sense the environment and in a second step process the sensed information.

Example: We applied this patter in all pedestrian simulation models. The agents scanned their complete environment within their range of perception and saved all objects in a $PerceptionMemory$ variable. In the next step all objects that are recognized as obstacles are filtered and stored separately for being used in the collision avoidance rules. A second independent preprocessing step allows the agent to sort out whether it has already reached its goal or not.

Configuration: The most basic parameter is the range of initial scan (a distance in metric space or a number of hops in a graph environment). This initial scan must be done with sufficient distance as the environment is only scanned once and all eventualities have to be foreseen.

Consequence: -

Related Patterns: This pattern might be used in combination with different memory structure patterns, such as the *Mental Map Pattern*, etc.

Although it takes quite some space to describe the *Perception Memory Pattern*, it is basically trivial. However our experience showed that especially beginners again and again access the environment in time intervals where no change can happen just because they want to save memory. However depending on the particular implementation of perception, accessing the environment often is the most expensive operation.

Beside other, already mentioned patterns, we might indeed find it useful to specify a BDI architecture in terms of a pattern. A pattern formalizing how to best design an adaptive discrete choice in term of a *Discrete Choice With Feedback Pattern* may be useful when agents have to choose from a set of options.

3.3 Agent Population Patterns

Especially interesting are design patterns that capture agent control aspects for producing some specific overall behavior on the aggregate level, such as a given population dynamics. Similar to the set of useful functions in the book of Haefner [16], one may identify several agent-level behavior modules that may lead to specific relations on the macro level. Such patterns are useful for all forms of systematic design – bottom-up or top-down. It is clear that for a final model these mostly isolated patterns have to be integrated into environmental dynamics, other behavioral feedback mechanisms, etc. One can image a lot of different population dynamics generated by different behavior and interaction behaviors on the agent level. The following *Exponential Growth Pattern* can be seen as the one of the simplest forms of such an agent population pattern.

Name: *Exponential Growth Pattern*

Intent: A population of agents should exhibit basic exponential growth in population numbers. This is basically the purest agent-based implementation of an exponential growth function.

Scenario: Useful for different scenarios where exponential growth of an agent population is necessary.
Description: Agent duplicate themselves. Duplication is triggered with a given individual probability.
Dependencies: none
Agent System Structure: There is just one agent class with one attribute. An additional global container may store references to all agents and serve as a bookkeeping device for the number of agents.
Agent Behavior: see figure 1

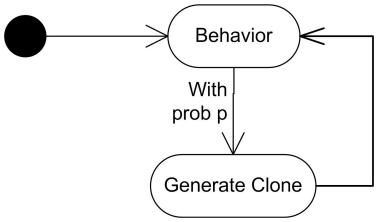

Fig. 1. Behavior specification for the agent behavior producing exponential growth.

Configuration: There is only one parameter per agent, namely the duplication probability. The basic question is how to set this local agent parameter for producing the corresponding macro behavior. Basically the macro rate should equal the probability for duplication. However, this is not so simple as illustrated in figure 2.
Even without parameter variations, the outcome of a short simulation generated with the same initial conditions, varies a lot. This is due to the high degree of randomness in this model formulation.
A technical issue concerns the integration of new agents into the update schedule of the simulation. One has to pay attention, whether there are delays originating from the inclusion of new agents into the overall update sequence.
Example: This pure agent system pattern is completely unrealistic in reality as there is no unconstraint, unbounded growth. However, an exponential growth model might set the frame for a more complex one, modifying the reproduction probability.
Consequence: Its exponential growth: depending on the parameter the population growth tremendously fast, sufficient computational resources are necessary. The system easily gets out of reasonable population sizes. Due to the relevance of the random component with direct influence on population sizes, it is hard to control.
Extensions: There are many obvious extensions addressing the decision for duplication: the probability may be replaced by a counter for some regular, deterministic reproduction. Both probability and counter-based reproduction can be modified by resource or energy bounds. Pattern-like structures can also be formulated for sexual reproduction, local density dependent duplication, etc.
Related Patterns: Pattern that remove agents from the simulated environment, such as age-dependent death, starvation or predation.

This *Exponential Growth Pattern* seems to be trivial, but it is also something that many modelers have used in their models without really reflecting about

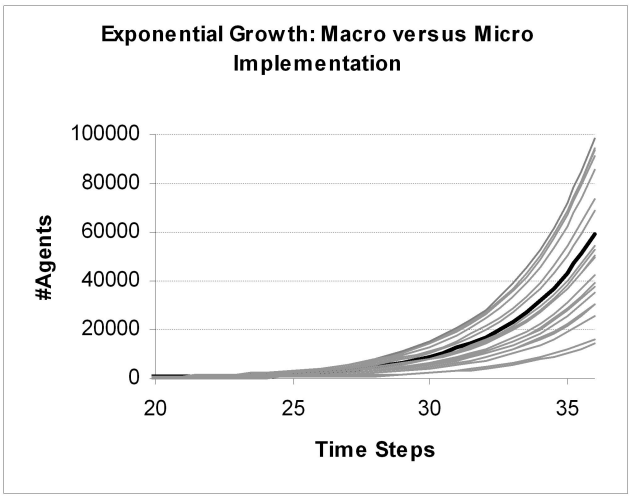

Fig. 2. Different micro runs do only in average resemble the macro model (black).

that this could be pattern – a special building block for a model that could be used to document best practice. As mentioned in the extensions section, there a set of potential variations. For bounded growth in interaction with other types of agents something like *n-species food web pattern* could be specified that tackles interacting populations of agents, but defined from an agent-perspective. In contrast to the following category the patterns here aim at reproducing certain macro-level population dynamics; the following focus on interaction between agents for coordination or for (self-)organization.

3.4 Interaction Patterns

In a similar way, one can specify good solutions to standard problems concerning negotiation among and organization of agents. Here, the intersection with agent-oriented software engineering patterns should be high - considering bidding, call-for-proposals, or even mediation pattern (see Section 2.1). As with the agent architecture pattern, the degrees of freedom in design are constrained by the required correspondence between original and model. Additional patterns can be interesting besides the patterns suggested by the agent software community, such as the *Tag-based Interaction Pattern* or *Memory-based Interaction* that can be often found in models of iterated games. In biological as well as social science models a pattern can be found describing how some form of interaction leaves marks in the beliefs/memory of an agent that influences the selection of future interaction partners. This pattern may be denoted *Emergent Organization Pattern*. Interesting patterns may also be found for specific organization

or relation networks – a pattern describing the initialization of a small world network is definitely useful.

3.5 Environment Patterns

Explicit space often plays an important role in agent-based simulation models. In most cases one can observe that space representation and consequently, environment-agent interactions are handled in similar ways. Patterns can be formulated like the *Background Cellular Automata Pattern* describing discrete cells that carry (dynamic) information that is perceived and modified not only by the agents, but also by the cells themselves potentially in relation to the values of their neighbor cells. Application examples can be found in certain pedestrian simulations like in the work of Bandini and co-workers [19] where a cellular automata is used to store the potential gradient for guiding the agents movement or in the well-known Sugarscape [20] model to represent the heterogenous renewable resources. Such a pattern would document how a cellular automata and its particular update could be used as the agents environment. Hereby a cell forms the position of an agent that accesses the state variable of that cell. A similar pattern for the continuous case may be something like a *Potential Field Pattern*.

Also for agent movement in space, patterns can be usefully applied. Just think about, how often a random walk has to be designed and how often the modeler starts anew thinking about whether a completely random change in direction from time to time or a slight random change in direction before every step is more appropriate. A *Random Walk Pattern* is a good way to formulate the experiences and guide future random walk designs. Similar considerations may result in a *Pheromone-based interaction* as a specific recurrent form of stigmergic interaction. Also more elaborate walking patterns can be useful, e.g. a *Path in Continuous Space Pattern* where the problem of obstacle avoidance in a continuous space without additional spatial structure is tackled.

4 Future Challenges: Formalization and Evaluation

The next steps in actually making these ideas viable would be a thorough examination of as many available models as possible for candidates of patterns. Those patterns then must be fully and precisely described and more clearly classified than we did in this paper. However, this raises the question what languages should be used for precisely characterizing the patterns, respectively the different elements of their description. Original UML [21] and its current versions or a selection from the many agent-oriented extensions of UML (such as MESSAGE/UML [22]) can be seen as good candidates at least for partial aspects of the overall description.

One can see that the number of potentially useful agent-based model design patterns for supporting the design of the a multi-agent model can be quite long. We gave a few examples and indicated a number of others whose usefulness is immediately clear to somebody who was confronted with such a problem.

Nevertheless the patterns in this list have to be evaluated, e.g. its usefulness and useability have to be tested.

What constitutes a good design pattern is ultimately an empirical question, and can only be answered by investigating what the outcome is when people actually use it. The usefulness of design patterns in software engineering have been empirically evaluated in a number of studies. The first study of that kind was conducted by Prechelt et al [23], who experimentally compared software maintenance with and without design patterns. They came to the conclusion that "pattern-relevant maintenance tasks were completed faster or with fewer errors if redundant design pattern information was provided." A similar study by Vokác [24] found that certain design patterns were easier to use than others, and resulted in considerable lower defect rates.

5 Conclusion

The process from conceptual to implemented model is the hardest part in modeling and simulation – especially when considering multi-agent simulations with their unlimited freedom of design. Therefore it is important to provide less experienced modelers with some guideline for a good model design. Following the success of patterns in object-oriented software engineering and consequently in agent-oriented software engineering, we discussed the application of patterns for agent-based simulation models. We gave some examples that illustrated the particular situation in agent-based modeling showing that actual patterns in agent-based simulation can be different from the ones suggested for standard agent-based software.

One may observe that all tackled patterns in this contribution are on a very technical level. Yet, the actual modeling problem often starts before the technical level. Nevertheless, a list of well-defined, evaluated patterns on the level we discussed here, may show the technical options that a modeler has for realizing a model concept, especially when the pattern are connected to a particular modeling and simulation platform that includes some facility for code generation.

References

1. Willemain, T.: Insights on modeling from a dozen experts. Operations Research **42**(2) (1994) 213–222
2. Gamma, E., Helm, R., Vlissides, R.J.J.: Design Patterns: Elements of reusable object-oriented software. Addison Wesley, Boston (1995)
3. Triebig, C., Klügl, F.: Designing components for multiagent simulation. In: Agent-Based Modeling & Simulation Symposium at EMCSR, Wien, April 2006. (2006)
4. Weiss, M.: Pattern-driven design of agent systems: Approach and case study. In Eder, J., Missikoff, M., eds.: Advanced Information Systems Engineering, 15th Int. Conf. CAiSE 2003, Klagenfurt, Austria. (2003) 711–723
5. Oluyomi, A., Karunasekera, S., Sterling, L.: A comprehensive view of agent-oriented patterns. Autonomous Agents and Multi-Agent Systems **15**(3) (2007) 337–377

6. Oluyomi, A., Karunasekera, S., Sterling, L.: Description templates for agent-oriented patterns. Journal of Systems and Software **81**(1) (2008) 20–36
7. Sauvage, S.: Agent oriented design patterns: A case study. In: AAMAS '04: Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society (2004) 1496–1497
8. Kendall, E.A., Krishna, P.V.M., Pathak, C.V., Suresh, C.B.: Patterns of intelligent and mobile agents. In: AGENTS '98: Proc. of the 2nd Int. Conf. on Autonomous agents, New York, NY, USA, ACM Press (1998) 92–99
9. Aridor, Y., Lange, D.B.: Agent design patterns: elements of agent application design. In: AGENTS '98: Proc. of the 2nd Int. Conf. on Autonomous agents, New York, NY, USA, ACM Press (1998) 108–115
10. Chacon, D., McCormick, J., McGrath, S., Stoneking, C.: Rapid application development using agent itinerary patterns. Technical Report 01-01, Lochheed Martin Advanced Technology Laboratories (2000)
11. Do, T.T., Kolp, M., Pirotte, A.: Social patterns for designing multiagent systems. In: Proc. of the 15th Int. Conf. on Software Engineering & Knowledge Engineering (SEKE'2003), San Franciso, USA, July 2003. (2003) 103–110
12. Hayden, S.C., Carrick, C., Yang, Q.: Architectural design patterns for multiagent coordination. In: In Proc. of the 3rd Int. Conf. on Autonomous Agents. (1999)
13. Oluyomi, A.: Pattern and Protocols for Agent-Oriented Software Engineering. PhD thesis, Department of Computer Science and Software Engineering, University of Melbourne, Australia (2006)
14. Klügl, F.: A validation methodology for agent-based simulations. In: SAC Symposium, Advances in Computer Simulation Track, March 2008, Ceara, BR. (2008)
15. Grimm, V., Railsback, S.F.: Individual-Based Modeling and Ecology. Princeton University Press (2005)
16. Haefner, J.W.: Modeling Biological Systems – Principles and Applications. 2nd edn. Springer, New York (2005)
17. Koenig, R., Bauriedel, C.: Modular system of simulation patterns for a spatial-processes laboratory. In: Proc. of the ICA Workshop on Geospatial Analysis and Modeling, Vienna, July 2006. (2006)
18. Schütze, M., Riegel, J.P., Zimmermann, G.: A pattern-based application generator for building simulation. In: Proceedings of the ESEC 1997, Zurich (CH). (1997)
19. Bandini, S., Manzoni, S., Vizzari, G.: Towards a methodology for situated cellular agent based crowd simulations. In Dikenelli, O., Gleizes, M.P., Ricci, A., eds.: Post-Proc. of the 6th Int. Workshop Engineering Societies in the Agents World (ESAW 2005). LNAI 3963, Springer (2006) 203–220
20. Epstein, J.M., Axtell, R.: Growing Artificial Societies. Social Science from the Bottom Up. Random House Uk Ltd (1996)
21. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison Wesley (1999)
22. Caire, G., et al.: Agent oriented analysis using Message/UML. In: AOSE '01: Revised Papers and Invited Contributions from the 2nd Int. Workshop on Agent-Oriented Software Engineering II, London, UK, Springer-Verlag (2002) 119–135
23. Prechelt, L., Unger, B., Tichy, W.: Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. IEEE Transactions on Software Engineering **28**(6) (2002) 595–606
24. Vokác, M.: Defect frequency and design patterns: an empirical study of industrial code. IEEE Transactions on Software Engineering **30**(12) (2004) 904–917