Performance Measurement of Multiagent Systems: Towards Dependable MAS

Ingo J. Timm, René Schumann
Goethe University Frankfurt am Main
timm|reschu@cs.uni-frankfurt.de

Keywords: Multiagent Systems, Simulation, Performance Measurement

Abstract

In this paper we discuss an inherent problem of multiagent systems (MAS), the validation of desired behavior of an agent as well as a MAS. MAS can adapt their behaviors to a dynamic environment. By the ability of adapting to a variety of changes in the environment, it becomes very hard to ensure that the systems always show an intended behavior on behalf of its owner. We present here a simulation-based approach that allows validating the behavior of MAS. The simulations runs are used to evaluate the MAS in the application domain. By varying simulation settings, changes in the environment can be tested, and thus the dynamic abilities of the MAS are evaluated. If required and enough time is available, different possible sequences of changes can be tested. Thus a confidence value for the trust in the MAS, that it will behave efficient and desired in the real application context, can be defined.

1. INTRODUCTION

It is often argued that multiagent systems (MAS) offer advantages in dynamic environments where flexible software solutions are needed (Kirn, 2006). But it has to be stated that exactly this flexibility is the cause for a significant drawback of MAS. The other side of the coin is that adaptable or even self-organizing systems can hardly be validated. That is to evaluate if the system meets its specifications in a given environment state. If dynamics in the environment is especially regarded in the software system, it has to be validated in all possible environment states. In general this is not possible as the number of possible environment states growth exponentially.

It can be argued that MAS are not dependable or at least bounded towards a minimal level of expected performance, this is one reason for the fact that MAS have not established to come to practical application in the field. One can overstate the current situation as follows:

> "We guarantee that your system will act in all possibly upcoming environments, but we cannot say that it will act as intended in any environment."

Thus we currently can build MAS that can react flexible to a changing environment but we cannot guarantee that the computed solutions remain within acceptable bounds. To improve this situation we argue that it is necessary to combine flexibility and dependability to be able to generate MAS relevant for broader practical application areas. A way to combine those originally contradicting requirements can be the concept of regulated autonomy presented in previous work (Schumann, Lattner, and Timm, 2008).

This paper focuses on the question, how MAS can be evaluated and if possible validated to show desired behavior independent of events that may change the environment. Therefore we advocate that simulations can help to evaluate the system's behavior in different changing environments. In this simulation the environment can be modified in a controlled way. This is a method that has to be classified as a software testing method. And of course, as we all know by the famous statement of Dykstra, testing cannot show the absence of failures, just their presence, it is clear that we cannot do verification for a MAS using this approach. But simulation can be used to investigate the possible environments that are classified by experts as most realistic future developments. This leads, with a comparable low degree of test coverage to a dependable impression of the abilities of the MAS at hand. Thus simulation studies seem to be an appropriate tool for the performance measurement of MAS, respecting the challenges from flexible systems.

The rest of this paper is organized as follows. In the next section we discuss methods of software validation and thereby highlight different testing methods. In section 3 we discuss the use of simulation studies for testing MAS. Based on this more general explanation we present in section 4 a case study how simulations can be used to measure the performance of MAS in concrete scenarios. We then discuss related work (section 5) and finally summarize our findings and discuss future research.

2. VALIDATION OF (DISTRIBUTED) SOFTWARE SYSTEMS

Ensuring that programs do what their programmers and users expect them to do is an important issue, not only in the field of software engineering but in all fields where software is developed. Different techniques have been developed. This section gives a brief overview, a more detailed overview can be found in (Menzies and Pecheur, 2005) with an additional focus on MAS in (Hormann, 2006). According to Menzies and Pecheur (2005) these techniques can be classified in five

groups with increasing strength in their results, as shown in figure 1.

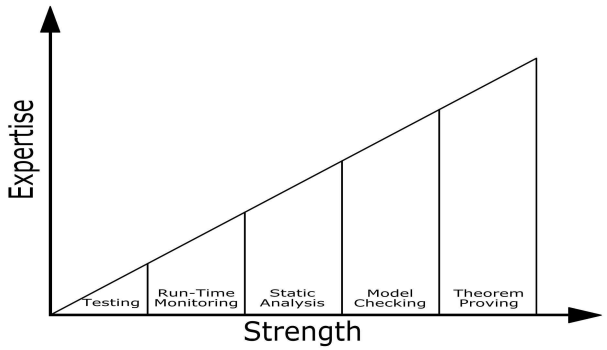


Figure 1. Software testing methods, (Menzies and Pecheur, 2005)

Theorem proving is the most formal method that can be applied to verify a program. A specialized logic has to be designed to reason formally about the correctness of the program. By standard theorem proving technologies it is tried to generate a proof that the program will only act in a specified way. Obviously, this is a very intensive and difficult approach, that can only be applied for safety critical systems, see e.g. (Menzies and Pecheur (2005); Timm, Scholz, and Fürstenau (2006)). In the MAS context actually no explicit approach is known to the authors, expect the METATE language presented by Wooldridge (2002).

Model checking is a formal technique, too. The aim of model checking is to verify that a certain formal property holds in all possible states. This implies that all possible states and the state transitions can be modeled. The advantage of model checking is that complete test coverage can be guaranteed. But it depends on the premises that a complete model can be defined, which is at least hard, if one has an open environment assumption, which is typical for MAS that offer flexible behavior. Wooldridge, Fischer, Huget, and Parsons presented the language MABLE, which allows code transformation into a Promela model which can be checked by the SPIN (Holzmann, 2003) model checker. Although the interaction of agents that is specified in MASE agent development methods can be transformed into Promela models and checked by SPIN (DeLoach, 2004). In a static analysis the source code of a program is analyzed in a static way, without executing it. Thereby control and data flow analysis are in the focus. Ideally this process can be completely automated, as it can be integrated in the compiler, for example. Here the special challenges arise for any distributed systems. Those systems consist of a number of potentially independent developed subsystems. Thus a simple analysis of the source code is not sufficient as the interaction between different agents is critical, as they can change the executed path, and thus lead to situations, an internal state of an agent, that cannot be accessed without interaction with other agents. Thus for distributed systems the interaction pattern, which can be very complex, has to be integrated into the analysis. It is doubtable if such integration is feasible at all. The idea of run-time monitoring is to supervise the behavior of a program during its execution. Most prominent are assertions, for instance, that are integrated in the Java runtime environment (since Java 1.4)[1]. The programmer can ensure that a specified Boolean expression is true at a given point during program execution by formulating such an assertion. If the assertion is violated, an exception is thrown. From a more abstract point of view the system to be monitored has to be wrapped and the input and output behavior have to be monitored. For a distributed system it is only reasonable to monitor all subsystems separately. Otherwise one cannot assume which input is known to which agent, and thus cannot evaluate if an observed reaction is acceptable with the given information. Moreover it is necessary to specify recovery actions that have to be taken if the system does show an unspecified actions or leave the interval of acceptable performance, as the system is actually operating. Two options are possible. The system can re-delegate the problem to a human operator, or a fallback strategy has to be defined, that is appropriate for the given situation. Runtime monitoring is a pre-requisite for the concepts of self-healing, self-optimizing systems. And although the constant monitoring of the system by itself and the reasoning about this observation is used in the concept of regulated autonomy (Schumann et al., 2008), as well. Approaches for run-time monitoring within MAS was presented e.g. by Timm (2004). In the classical software engineering the idea of software test is already well established in research and practice. A first overview of software testing in software engineering can be found in (Goldsmith and Graham, 2002) and is just sketched here very roughly:

- Unit tests: Within a unit test, a specific unit is tested, in terms of correct input / output behavior.
- Integration tests: The goal of integration tests is to evaluate and check the interaction of different units. Thereby sequences of interactions are of particular interest.
- System tests: In a system test, the entire system is tested to evaluate if all functional and non-functional requirements are met by the system.
- User acceptance tests: This test is, in contrast to the test previous mentioned ones, not done by the software developer, but by the user. It is commonly done with real data and under realistic environments.

A general problem of software testing is that only a very limited number of possible execution paths are tested. Thus

[1] See e.g. `http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html`

the test coverage is very low, which leads only to a weak confidence level. Actually there exist no systematically testing approaches for MAS, except (Hormann, 2006), to the best knowledge of the authors. Hormann (2006) has presented a method for comparable to unit testing for the JADE[2] agent development platform.

3. SIMULATION STUDIES FOR VALIDATING MAS

Simulation techniques are also used for modeling and analyzing dynamic, complex systems. MAS are a specific form of dynamic systems, i.e. MAS are software systems consisting of various autonomous agents and each of the agents uses a hidden internal state. Normally simulation of MAS is used to evaluate the quality of the computed solutions, to compare them with results of other techniques or actual methods in practical use for a given problem. Here we discuss the value of simulation for validating MAS. First of all it has to be stated that simulation can be seen as a form of system or acceptance test that allows evaluating the behavior of a MAS for a set of possible environments. Thus it is common knowledge that results from simulation studies cannot be used to argue that the system has no errors or always will act on the behalf of the user. If one uses a white-box or grey-box approach, such that one can look inside the MAS and use internal information, it can be possible to monitor single agents of the entire MAS, what allows a form of runtime monitoring. This combination of runtime monitoring within a simulation can be very fruitful for evaluating and debugging a MAS. If one has no insights of the MAS, runtime monitoring is not possible, and the data collected during the simulation has to be analyzed to evaluate the behavior of the MAS. In contrast to structured testing the test coverage of all possible environments that can be reached by simulation studies is comparable low. Nevertheless an acceptable confidence of the intended user can be reached by simulation studies. The reason for this is that simulations can be seen as knowledge guided exploration of the state space of possible environments of the MAS. If different changes in the environment can be classified with a probability, one can start with an emulation of the current situation and then evaluate possible developments. Even if the number of possible states that are investigated is relatively small, each simulation run is very time consuming. Therefore it is necessary to enforce the automation of the entire validation process. The possible design of such a simulation tool is shown in figure 2. The components of this tool are sketched in the following. They have to automate the process as far as possible to keep the approach feasible. The basic scenario and possible events with different probabilities should be specified by the expert. The quality of this estimation is, of course, critical to obtain the desired confidence. Based on this data the specification for the environments can be generated in a semi-automated way, at least. An example for the automated test scenarios generation is presented in (Maiden, 1998). The simulation tool can be a standard commercial of the shelf tool, like for instance Plant Simulation[3]. The MAS has to be integrated into this simulation tool, for instance this was done for the evaluation of IntaPS (Lorenzen, Woelk, Denkena, Scholz, Timm, and Herzog, 2006). It can happen that the results of a simulation run are not complete or that the simulator does not compute results in a timely fashion, because it may crash. The correct execution of a simulation has to be observed, and results have to be checked for completeness and plausibility. This is necessary to ensure that only valid results are used in the further processing steps. If a simulation run did not generate proper data, it has to be repeated. This is done by the simulationcontroller. Moreover this component could be used for the adaptation phase of the MAS to the given scenario, if the MAS need a setup phase.

The interpretation of the results can be done using again an existing tool for statistical analysis, e.g. the R project[4]. To control the entire process another controller, called simulationstudycontroller, is needed. This control has some important functions. First it has to calibrate the model that is that the simulation computes results that can be observed in the real system as well. This situation is the starting point for the knowledge guided exploration of possible worlds. Second the controller has to control the scenario generator based on the information given by the user and the interpretation of the computed results. For example, it might be useful to use results from the interpretation module to guide further simulation runs in ”interesting” areas or it can enforce more replications if statistical relevance cannot be proven by existing data.

4. VALIDATION BY SIMULATION: A CASE STUDY

In our research on intelligent agent architectures we are working on an innovative goal selection mechanism in BDI agents. Our primary research objective is to develop an algorithm which could consider conflicts of interests between desires explicitly. Our primary research objective is met as the resulting algorithm, cobac (conflict-based agent control), was modeled and specified formally. However, a proof-of-concept in form of a prototypical implementation is required. Like various research in MAS, our primary research question does not address an improvement or optimization of existing approaches but the invention of an algorithm which handles domain specific aspects which are not considered in the cur-

[2]Java Agent DEvelopment Framework (http://jade.tilab.com/)

[3]http://www.ugsplm.de/produkte/tecnomatix/plant_design/em_plant.shtml

[4]http://www.r-project.org/

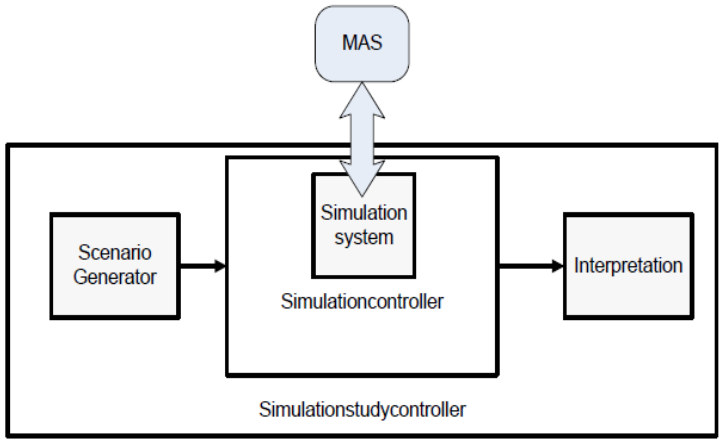


Figure 2. Components of a simulation tool for validating MAS

rent state-of-the-art. Consequently, it is often claimed that a comparison or a performance measurement is not necessarily within the scope of evaluation or validation. In our context, the application of the MAS is planned for the production manufacturing domain. With respect to an interdisciplinary research approach, we have to explain the domain experts the benefits as well as the functionality of the system within the application domain. Therefore, we are using domain scenarios within a simulation-based testing procedure to evaluate and validate the agent architecture and the goal selection mechanism. In the following we will introduce the algorithm briefly, present our evaluation approach, and interpret the results.

In order to balance various desires in the manufacturing domain e.g., produce or maintain, agents need to implement a sophisticated conflict management. In the Discourse Agent architecture, which is based on BDI, conflict management is integrated explicitly by the cobac-algorithm. Doing so, the process of goal generation and option filtering is substituted. In the first step of the cobac algorithm options are generated - in analogy to conventional BDI approaches - on basis of accessible desires and current intentions. A first extension leads to the creation and allocation of plans for any accessible desire. Accessible desires and active intentions are handled as options; however, on the formal basis (multi-modal logic), there is no efficient way to decide on accessibility. In BDI, the powerset of accessible desires and intentions is computed (reduced to consistent elements) and a non-deterministic selection of one element of this set as an intention is performed. In cobac, the powerset is not computed and each option is assessed by an evaluation function considering chance of pursuing this option, risk of ignoring this option, as well as user preference. In the next step the options are filtered. The filtering uses conflict assessment and resolution, i.e., for each pair of options, a synergy as well as a conflict value is calculated. Two options receive a high synergy value if they are pursuing similar desires and the plans are not contradictory, e.g. the post condition of plan A is not prohibiting the pre condition of plan B. A conflict and synergy potential is calculated as the sum of each conflict and synergy value and is used as a performance indicator within the process of conflict resolution. The conflict classification and resolution is motivated from the field of inter-personal conflict studies and following a conflict taxonomy which is introduced in (Timm, 2004), where each pair of options is classified as a leaf in this taxonomy. For each type of leaf, there is a resolution strategy taking the cooperation or conflict potential into account. For instance if two objectives are very similar, they can be merged in a cooperative setting and two objectives pursuing conflicting post conditions can be removed.

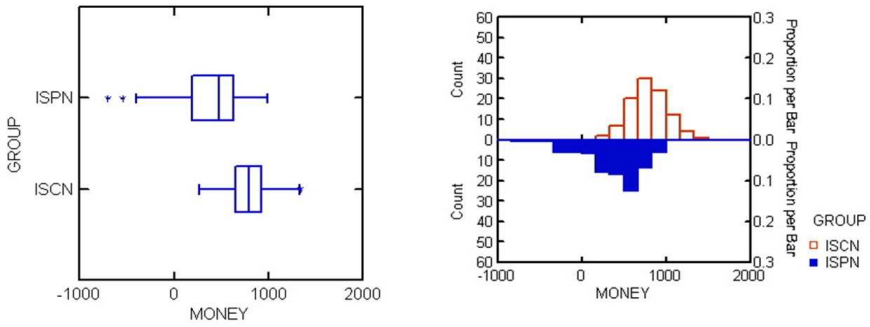
The evaluation of our approach is two-fold. On the one hand, the prototypical implementation is used for proof-of-concept to support the primary research question. On the other hand, the prototype should be used for evaluation and performance measurement with respect to a representative model of the application domain. The main challenge for the proof-of-concept of cobac lies in the implementation of the (theoretically infinite) accessibility relation, which is based on the possible worlds semantics in BDI. We suggest explicit restriction to the length of the temporal structure and inherit five different types of accessibility consideration in the agent deliberation process. In this paper we present results of the variation where no accessibility relation is computed, i.e., any intention and desire is assumed to be accessible. With these modifications a prototypical system in the production engineering domain was built. However, the second step of the evaluation process, performance measurement, is not handled by this first evaluation.

After proofing the concept, it is necessary to evaluate, whether the approach is adequate with respect to the desired application domain. In this phase, it has to be shown, that the MAS is solving problems of the application domain meeting the requirements. However, a further step in the evaluation process includes the comparison of the solution to available optimal, state-of-the-art, or best practice solutions for this domain. Consequently, for evaluation purposes, at least one simulation model of the desired application domain is required. The design of the simulation system is based on the abstract model in figure 2. The modular design enables testing of multiple MAS and consequently for the comparison of MAS. One of the main problems of evaluating MAS is that MAS are implemented specifically for tasks with a high degree of autonomy and flexible interaction. Thus, it is difficult to compare a MAS to another in a specific domain as it could require to implement both MAS. Considering our case study, it is important to define scenarios which are representing standard and extreme situations of the desired application. For evaluating cobac we specified a shop floor with resources and manufacturing orders. The representation of resources includes error probability and maintenance models and various as well as partially conflicting desires. Orders and resources should be controlled by intelligent agents - however, the resource agents are implemented by Discourse Agents with cobac algorithm. As a contrast to this, we implemented the Discourse Agents with and optionally without cobac option filtering, such that as many concepts as possible are shared between these approaches. As we also have internal knowledge about the simulation model, we explicitly use this to develop a system which is nearly optimal in the concrete scenario. In our case, especially the error and maintenance probability is used for creating a specific algorithm. This specific algorithm cannot be used as a standard solution for the practice, but it can be used to determine a ”reference point”. On this basis we can analyze the system behavior and measure the ”performance” with respect to the application.

For simulation-based testing and the following measurement and interpretation of the results, we specified 17 different simulation configurations; each of them has been computed 100 times to reach statistical significance in the results. The different configurations cover different sizes of the desire set, different accessibility relation models, and increased and decreased error probability. The length of each simulation run is set to 96 deliberation steps which cover one shift in the shop floor. The cobac algorithm as well as the priority option filtering algorithm are each tested with a set of generic desires (five desires) and an extended, set of partially composed desires (seven desires), as well as five different approaches to accessibility computation. Our system automatically generates programs for computing explorative and test statistics with Systat. The results of the explorative analysis are automatically transformed into a pdf-document for future reference. Even with this level of automation, the interpretation of the results is extremely complex and it is not feasible, that it is performed by a human expert by himself. E.g., if only one key performance indicator is used for consideration of the differences of our 17 configurations lead to 153 tests; even if only tests within a group with comparable parameters is performed, there are around 40 tests to perform. The challenge in the interpretation is to find the interesting and representative results within the simulation runs. In figure 3 results of the experiments with a generic set of desires and no computation of accessibility relation are plotted for cobac (ISCN) and priority-based selection (ISPN). It seems to be obvious, that cobac is superior to priority-based selection. In this context we introduced the reference point experiments, which use internal knowledge of the simulation model design. The results of these experiments are used as a reference (as 100%); interpreting our results in this context, it can be assumed that cobac reaches between 70% and 80% of an optimal solution with respect to money balance. In contrast to this, priority-based selection reaches ca. 35% of this reference solution. For further details on the evaluation, please refer to Timm (2004).

5. RELATED WORK

The focus of this work is on the validation of the behavior of MAS. As automated verification, done either by theorem proving or model checking can only be done to assert, formulated as a logic formula, that a given statement holds during the systems execution or not, it becomes hard to investigate the entire specification of the given MAS, because for each assertion that results from the specification the entire verification process has to be done. Thus we do not discuss such approaches here in more detail. The field of validating distributed software systems cannot be surveyed here exhaustively. More detailed discussions of existing approaches can be found in (Sudeikat, Braubach, Pokahr, Lammersdorf, and Renz, 2006; Fuentes, Gómez-Sanz, and Pavón, 2004). Fortino, Garro, and Russo (2005) present in their work an integrated development and validation framework. For their validation they use a simulation-based approach, as well. In their article Fortino et al. present an development approach that bases on the Gaia method presented by Wooldridge, Jennings, and Kinny (2000). To simulate a MAS, the designer has to build a simulator program, which embeds the agents in the presented simulation framework. At the core of this framework an event-driven simulator is used, based on the idea of statecharts. The simulator is responsible for the interleaved execution of the event processing of the agents and for the passing of events among the agents. Thus it emulates an agent platform. This implies that the agents have to be adapted after the simulation phase to the desired productive platform.



a) b)

Figure 3. Results of the cobac-evaluation

This implies that the system is changed after its validation phase. What can be problematic is that after the system is validated by simulation, it is modified by hand to adapt it to its intended agent platform. This, of course, can be an erroneous process step and compromise all results from the validation phase. Sudeikat et al. (2006); Sudeikat and Renz (2006) and present in their work techniques for validation of belief-desire-intention (BDI) agents. In their work they survey existing methods and testbeds for BDI agents and discuss different technologies for validating and testing BDI-based MAS. Runtime monitoring, namely assertions, and techniques for static analysis were presented to validate single agents (Sudeikat et al., 2006) and a technique to monitor a MAS consisting of goal directed agents (Sudeikat and Renz, 2006). A case study is implemented using the Jadex[5] framework, where the assertion techniques have been added. For validating special aspects of single agents or interaction protocols different approaches have been proposed, e.g. (Poutakidis, Padgham, and Winikoff) or (Liedekerke and Avouris, 1995). As we are interested here in validation the system behavior and not the subsystem or interactions, we will not go into detail here. In (Sudeikat et al., 2006) existing approaches are presented. Another approach to generate more reliable multiagent systems can be achieved using the principles of model-driven-architectures (MDA). Following the MDA approach the MAS is generated automatically. The designer specifies the system on an abstract level and the executable system is generated automatically. If those transformation steps are validated, it can be ensured that a resulting MAS implementation will behave as specified. Of course, it has to be shown that the defined concepts in the MDA approach are sufficient to model the requirements of the application at hand. If, for example, the needed system behavior cannot be expressed, for instance, real-time behavior, then the MDA approach can ease the development process, but do not offer a benefit for validating the system. Approaches to design MAS using MDA techniques are discussed in (Amor, Fuentes, and Vallecillo, 2004; Gómez-Sanz and Fuentes, 2003; Hahn, 2008).

[5]see `http://vsis-www.informatik.uni-hamburg.de/projects/jadex/`

6. CONCLUSION

For the last century, agent technology is claimed to be next generation's paradigm for modeling, designing, and implementing large scale, adaptive, and intelligent software. The natural distribution of MAS as well as the complex interaction skills is used as standard arguments, "proofing" the benefit. In opposite to these expectations, MAS is not a standard approach for building large scale, flexible software. One main bottleneck for the application of agent technology is, that proving, proofing, validating, or even evaluating MAS' behavior is difficult and standard testing and assessing are currently subject of research and not established, yet. This is an especially challenging field as one face a dilemma designing MAS. On the one hand, one has to design a system that can adapt its behavior flexible to its environment. On the other hand, this flexibility is the main obstacle in validating the system. Therefore, in this paper, we introduced an approach for testing and simulating MAS on the basis of individual agents. Thus, we are proposing a framework for controlled testing, which allows simulating comparable and repeatable environments. In the case study we analyzed our approach of testing and validating a MAS for the manufacturing domain. Even if we have a high degree of automation, the interpretation effort is so high, that it is not feasible to perform it manually. The resulting challenges can be found in automatic execution and variation of simulation models and automatic analysis of statistical exploration and test results. These challenges are object of our current research: we are developing a knowledge-based approach to support configuration, calibration, and statistical interpretation of simulation models. In a first step, this approach is applied to simulation techniques in general. After

positive results, we plan to transfer these results to the automated validation of MAS as outlined in this paper.

REFERENCES

Amor, Mercedes, Lidia Fuentes, and Antonio Vallecillo. 2004. Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. In *Agent-Oriented Software Engineering V, 5th International Workshop, AOSE 2004*, ed. James Odell, Paolo Giorgini, and Jörg P. Müller, vol. 3382 of *Lecture Notes in Computer Science*, 93–108. New York, NY, USA: Springer.

DeLoach, Scott A. 2004. The mase methodology. In *Methodologies and software engineering for agent systems*, ed. Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, 107–125. Multiagent Systems, Artificial societies, and simulated organizations, Bosten, Dordrecht, London: Kluwer Academic Publishers.

Fortino, Giancarlo, Alfred Garro, and Wilma Russo. 2005. An integrated approach for the development and validation of multi-agent systems. *Computer Systems Science and Engineering* 4:259–271.

Fuentes, Rubén, Jorge J. Gómez-Sanz, and Juan Pavón. 2004. Verification and validation techniques for multi-agent systems. *Upgrade, Council of European Informatics Societies* 5(4):15–19.

Gómez-Sanz, Jorge J., and Rubén Fuentes. 2003. Agent oriented software engineering with INGENIAS. In *Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)*, ed. Vladimír Marík, Jörg P. Müller, and Michal Pechoucek, vol. 2691 of *LNCS*, 394–403. Prague, Czech Republic: Springer.

Goldsmith, Robin F., and Dorothy Graham. 2002. The forgotten phase. *Software Development* 45–47.

Hahn, Christian. 2008. A domain specific modeling language for multiagent systems. In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, ed. Lin Padgham, David C. Parkes, Jörg P. Müller, and Simon Parsons, 233–240. Estoril, Portugal.

Holzmann, Gerard J. 2003. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional.

Hormann, Arne. 2006. Testbasierte spezifikation von agenteninteraktionsverhalten. Diploma thesis, University Bremen,.

Kirn, Stefan. 2006. Flexibility of multiagent systems. In *Multiagent engineering*, ed. Stefan Kirn, Otthein Herzog, Peter Lockemann, and Otto Spaniol, 53 – 69. Berlin: Springer.

Liedekerke, Marc H van, and Nicholas M Avouris. 1995. Debugging multi-agent systems. *Information and Software Technology* 37:102–112.

Lorenzen, Leif-Erik, Peer-Oliver Woelk, Berend Denkena, Thorsten Scholz, Ingo J. Timm, and Otthein Herzog. 2006. Integrated process planning and production control. In *Multiagent engineering: Theory and applications in enterprises*, ed. Stefan Kirn, Otthein Herzog, Peter Lockemann, and Otto Spaniol, 91–113. International Handbook on Information Systems, Heidelberg: Springer.

Maiden, N. A. M. 1998. CREWS-SAVRE: Scenarios for Aquiring and Validating Requirements. *Automated Software Engineering* 5(4):419–446.

Menzies, Tim, and Charles Pecheur. 2005. Verification and validation and artificial intelligence. *Advances in Computers* 65.

Poutakidis, David, Lin Padgham, and Michael Winikoff. Debugging multi-agent systems using design artifacts: the case of interaction protocols. In *Proceedings of the first international joint conference on autonomous agents and multiagent systems (aamas '02:)*.

Schumann, René, Andreas D. Lattner, and Ingo J. Timm. 2008. Regulated autonomy: A case study. In *Intelligente systeme zur entscheidungsunterstützung,teilkonferenz der multikonferenz wirtschaftsinformatik*, ed. Lars Mönch and Giselher Pankratz, 83–98. München: SCS Publishing House.

Sudeikat, Jan, Lars Braubach, Alexander Pokahr, Winfried Lammersdorf, and Wolfgang Renz. 2006. On the validation of belief-desire-intention agents. In *4th International Workshop, ProMAS 2006*, 185–200. LNCS, Hakodate, Japan.

Sudeikat, Jan, and Wolfgang Renz. 2006. Monitoring group behavior in goal-directed agents using co-efficient plan observation. In *7th International Workshop on Agent-Oriented Software Engineering (AOSE 2006)*, ed. Lin Padgham and F. Zambonelli.

Timm, Ingo J. 2004. Dynamisches Konfliktmanagment als Verhaltenssteuerung intelligenter Agenten. Ph.D. thesis, University Bremen.

Timm, Ingo J., Thorsten Scholz, and Hendrik Fürstenau. 2006. From testing to theorem proving. In *Multiagent engineering: Theory and applications in enterprises*, ed. Stefan Kirn, Otthein Herzog, Peter Lockemann, and Otto Spaniol, 531 – 554. International Handbook on Information Systems, Heidelberg: Springer.

Wooldridge, Michael. 2002. *An introduction to multi agent systems*. John Wiley & Sons Ltd.

Wooldridge, Michael, Michael Fischer, Marc-Philippe Huget, and Simon Parsons. Model checking multi-agent systems with mable. In *Proceedings of the first international joint conference on autonomous agents and multiagent systems (aamas '02:)*.

Wooldridge, Michael, Nick R. Jennings, and David Kinny. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems* 3(3):285–312.