Testing in Multi-Agent Systems

Cu D. Nguyen[1], Anna Perini[1], Carole Bernon[2],
Juan Pavón[3], and John Thangarajah[4]

[1] Center for Information Technology FBK-IRST, Trento, Italy
{cunduy,perini}@fbk.eu
[2] IRIT, University of Toulouse, Toulouse, France
bernon@irit.fr
[3] GRASIA, Universidad Complutense de Madrid, Madrid, Spain
jpavon@fdi.ucm.es
[4] RMIT University, Melbourne, Australia
john.thangarajah@rmit.edu.au

Abstract. Testing software agents and Multi-Agent Systems (MAS) needs suitable techniques to evaluate agent's autonomous behaviours as well as distribution, social and deliberative properties, which are particular to these systems. This paper reviews testing methods and techniques with respect to the MAS properties they are able to address. For this purpose, we provide a reference framework that provides a classification of MAS testing levels (such as unit, agent, integration, system, and acceptance) and of testing approaches along the development artefact they exploit (namely, design and code artefacts). Open issues in testing MAS are then discussed providing a basis for a research roadmap.

1 Introduction

Software testing aims at answering questions like *Is the system being built good enough? Does the resulting system fulfil the requirements?* That is, testing is the software development activity, devoted to evaluating product quality and improving it by identifying defects and problems.

The specific nature of software agents, which are designed to be distributed, autonomous, and deliberative makes it difficult to apply existing software testing techniques to them. For instance, agents operate asynchronously and in parallel, which challenges testing and debugging. Agents communicate primarily through message passing instead of method invocation, so traditional testing approaches are not directly applicable. Agents are autonomous and cooperate with other agents, so they may run correctly by themselves but incorrectly in a community or vice-versa. Moreover, agents can be programmed to learn (i.e., change their behaviour); so successive tests with the same test data may give different results. Initial works on evaluating MAS quality focused on the definition of techniques for automating the validation of MAS specifications through formal proofing or model-checking [2, 20], and on the development of debugging techniques and tools to enhance MAS development platforms [15, 3, 31].

M.-P. Gleizes and J.J. Gomez-Sanz (Eds.): AOSE 2009, LNCS 6038, pp. 180–190, 2011.

Structured testing approaches have been proposed more recently, to complement analysis and design methodologies [4, 26, 31, 14]. These approaches rest on the idea that the behaviour of the MAS can be dynamically evaluated providing as input a set of test cases that are derived from analysis and design artefacts.

Differently from these techniques, simulation-based approaches aim at detecting abnormal behaviours while a simplified version of the system (or a model of it) is executed in a virtual environment [13, 35, 6, 18]. These approaches seem to be particularly appropriate to evaluate emerging behaviours in self-organizing systems [7]. Data mining techniques have been applied to analyse simulation logs for large MAS, as it is the case in ACLAnalyser [33, 34], which is a way to cope with scalability of MAS.

In this paper we survey testing methods and techniques, characterizing them along the testing levels they support (such as unit, agent, integration, system, and acceptance). We differentiate them also between simulation-based techniques (called *passive* approaches) and structured testing methodologies (called *active* approaches).

The paper is structured as follows. Section 2 presents a MAS testing framework. State of the art MAS testing approaches are then surveyed in Section 3. Open issues in testing MAS and promising testing techniques are then discussed in Section 4, providing a basis for a research roadmap.

2 Classification Dimensions

In order to provide the reader with a structured view for clarity, we propose to classify existing work on MAS testing following two different aspects: *testing levels* and *testing techniques*. Testing in MAS consists of five levels, as proposed in [23] and [22]: *unit, agent, integration, system*, and *acceptance*. The testing objectives, subjects to test, and activities of each level are described as follows:

- *Unit*. Test all units that make up an agent, including blocks of code, implementation of agent units like goals, plans, knowledge base, reasoning engine, rules specification, and so forth; make sure that they work as designed.
- *Agent*. Test the integration of the different modules inside an agent; test agents' capabilities to fulfil their goals and to sense and effect the environment.
- *Integration* or *Group*. Test the interaction of agents, communication protocol and semantics, interaction of agents with the environment, integration of agents with shared resources, regulations enforcement; Observe emergent properties, collective behaviours; make sure that a group of agents and environmental resources work correctly together.
- *System* or *Society*. Test the MAS as a system running at the target operating environment; test the expected emergent and macroscopic properties of the system as a whole; test the quality properties that the intended system must reach, such as adaptation, openness, fault-tolerance, performance.
- *Acceptance*. Test the MAS in the customer's execution environment and verify that it meets stakeholder goals, with the participation of stakeholders.

We will organise existing work along these levels to see on which levels each work focuses[1].

We can also use testing techniques in terms of their test input perspective: *passive* and *active*, to classify existing work. Some employ the passive perspective to solely observe the output behaviours, test inputs are often predefined configurations or ignored; while others adopt the active perspective seeking extensively for good test inputs and at the same time monitoring the output behaviours of the subjects under test.

It is worthwhile noting that the two chosen dimensions, i.e. testing levels and test techniques, are not intended to be comprehensive. They rather provide a useful framework to systematically organise existing work. The next section will give a survey of the most relevant and active work on MAS testing according to these classification dimensions.

Moreover, the work surveyed are also assessed in terms of their maturity into *usable*, *in progress*, and *concept* as follows: *Usable:* there are published results, there is available material that permits to reproduce the results, and the work has been used in the development of medium or big size projects. *In progress:* there are published results, there is available material that permits to reproduce the results. So far, the work has been used in academic size projects. *Concept:* there are published results, but the applicability of the work is still to be proven.

3 MAS Testing Approaches

The data in Table 1 give an overview of the most recent and active work in this domain. They are organized according to the classification introduced in the previous section. The third dimension, *maturity*, will be tagged to each work. In the following sections, we first summarize the contributions of the works that focus mainly on a particular testing level, e.g. *agent*; hereafter are the works that tackle more than one testing level. In addition, we also summarize some interesting work that do early validation of the agent and MAS design based on simulation techniques and generated agent skeletons. It is non-trivial to organize

Table 1. State-of-the-art work on MAS testing

	Unit	***Agent***	***Integration***	***System***	***Acceptance***
Active	Zhang et al. (2007, 2008, 2009), Tiryaki et al. (2006), Ekinci et al. (2008) , Nguyen et al. (2010)	Núñez et al. (2005), Coelho et al. (2006), Tiryaki et al. (2006), Gómez-Sanz et al. (2009), Nguyen et al. (2008, 2009, 2010)	Gómez-Sanz et al. (2009), Nguyen et al. (2010)	Nguyen et al. (2010)	Nguyen et al. (2010)
Passive		Lam and Barber (2005), Núñez et al. (2005), Gardelli et al. (2005), Fortino et al. (2006), Bernon et al. (2007), Cossentino et al. (2008)	Sierra et al. (2004), Botía et al. (2004), Rodrigues et al. (2005), Serrano and Bota (2009), Serrano et al. (2009), Sudeikat and Renz (2009)	De Wolf et al. (2005)	

[1] These works are organized in chronological order; those published in the same year are ordered alphabetically.

these works following the proposed classification since their subjects under test are agent design, not agent code. However, we decide to include them because they are complementary to testing, and both types contribute to the final goal of ensuring the quality of the agent or MAS under development.

3.1 Unit Level

Zhang et al. [38, 39, 40], *in progress*: This body of work presents a framework for automated unit testing of agent systems. The approach is a model-based testing approach where the models used are the design artefacts of the agent design methodology. The chosen methodology (though it is extensible to other methodologies with similar concepts) is Prometheus [28] and the corresponding tool that incorporates the testing techniques is the Prometheus Design Tool (PDT[2]) [39].

In [38], the basic units for testing are identified as events, plans and beliefs. Events are tested for coverage (does this event get handled) and overlap (are there multiple plans that can handle this event in the same situation). Plans are tested for whether the plan gets triggered in at least some situations (but not all if there is a condition specified to its applicability), does the plan complete[3], and does the plan post the events that it should (as indicated in the design). The testing of beliefs are limited to verifying whether the data fields as indicated in the design are implemented and if there are any events posted by the belief (e.g., if an event is posted when a belief is updated), test if these events do get posted.

The paper details mechanisms for identifying the order in which the units are to be tested, for example, if unit x depends on unit y then y must be tested before x. It also details the process of generating test cases and outlines the overall testing process. All the mechanisms are fully automated, but allow for user input at various stages.

In [40] the details of how variables are to be specified, extracted and assigned values to create test cases are presented. Furthermore, in order to execute the test cases the system environment needs to be setup. For example, there may be interaction with an external program within a plan unit (e.g. a query to an external database of a web service). There may also be interaction with another agent within a testing unit. This work deals with the above by introducing initialization procedures and mock agents to simulate other agents/systems.

Ekinci et al. [10], *in progress*: This work considers agent goals as the smallest testable units in MAS and proposes to test these units by means of *test goals*. Each test goal is conceptually decomposed into three sub-goals: *setup*, *goal under test*, and *assert*. The first and last goals prepare pre-conditions and check post-conditions respectively, while testing the goal under test.

[2] `www.cs.rmit.edu.au/agents/pdt`

[3] Note, here it tests for plan completion not for plan success.

3.2 Agent Level

Lam and Barber [19], *in progress*: This work proposes a semi-automated process for comprehending software agent behaviours. The approach imitates what a human user, can be a tester, does in software comprehension: building and refining a knowledge base about the behaviours of agents, and using it to verify and explain behaviours of agents at runtime.

Núñez et al. [27], *in progress*: This paper introduces a formal framework to specify the behaviour of autonomous e-commerce agents. The desired behaviours of the agents under test are presented by means of a new formalism, called *utility state machine* that embodies users' preferences in its states. Two testing methodologies are proposed to check whether an implementation of a specified agent behaves as expected (i.e., conformance testing); one active, the other passive. In their *active* testing approach, they use for each agent under test a *test* (a special agent) that takes the formal specification of the agent to facilitate it to reach a specific state. The operational trace of the agent is then compared to the specification in order to detect faults. On the other hand, the authors also propose to use *passive* testing, in which the agents under test are observed only, not simulated like in active testing. Invalid traces, if any, are then identified thanks to the formal specifications of the agents.

Coelho et al. [5], *in progress*: Inspired by JUnit [12], this paper proposes a framework for unit testing of MAS based on the use of *Mock Agents*. Their work focuses on testing roles of agents. Mock agents that simulate real agents in communicating with the agent under test were implemented manually; each corresponds to one agent role.

Nguyen et al. [25], *in progress*: This work takes advantage of agent interaction ontologies that define the semantics of agent interactions to: (i) generate test inputs; (ii) guide the exploration of the input space during generation; and, (iii) verify messages exchanged among agents with respect to the defined interaction ontology. A set of generation rules have been defined and using them an automated testing framework can test a particular agent extensively through a large and diverse number of test cases.

Nguyen et al. [24], *in progress*: Agent autonomy makes testing harder: autonomous agents may react in different ways to the same inputs over time, because, for instance they have changeable goals and knowledge. Testing of autonomous agents requires a procedure that caters for a wide range of test case contexts, and that can search for the most demanding of these test cases. Nguyen et al. [24] introduce and evaluate an approach to testing autonomous agents that uses evolutionary optimization to generate demanding test cases. In this work, the authors have proposed a systematic way of evaluating the quality of autonomous agents. First, stakeholder requirements are represented as quality measures, and corresponding thresholds are used as testing criteria. Autonomous agents need to meet these criteria in order to be reliable. Fitness functions that

represent testing objectives are defined accordingly, and guide this evolutionary test generation technique to generate test cases automatically.

3.3 Integration Level

Serrano and Botía [33], **Serrano et al. [34]**, *usable*: One of the issues when testing MAS is their scalability, because of the number of agents and more specifically the huge number of interactions that may arise. This makes it very difficult to apply classical testing techniques. ACLAnalyser addresses these issues by applying data mining techniques on the logs of MAS executions (initially, on the JADE agent platform). This allows discovering emergence patterns at system (social) level. For instance, the creation of communities of agents with strong interaction links can be detected with the clustering facilities of ACLAnalyser.

3.4 Multiple Testing Levels

Tiryaki et al. [37], *in progress*: This work proposes a test-driven MAS development approach that supports iterative and incremental MAS construction. A testing framework called SUnit, which was built on top of JUnit[12] and Seagent [9], was developed to support the approach. The framework allows writing tests for agent behaviours and interactions between agents.

Gómez-Sanz et al. [14], *usable*: This work focuses on agent and interaction testing level. The INGENIAS Development Kit also provides facilities for MAS testing. It relies on the model-driven approach of INGENIAS, which allows the specification of test suites when modelling the MAS. It is possible to specify testing deployments, interaction tests, and agent mental state inspection. MAS models are then transformed into code on the INGENIAS Agent Framework (running on top of JADE agent platform), where tests can be executed and the developer can get information on the progression of organization workflows, interactions, and agents' mental states. Test suites can be automated, and this facilitates model-driven agile development, as it is easy to perform iterations, each one passing a test suite.

Nguyen et al. [26], *in progress*: The paper proposes a comprehensive testing methodology, called Goal-Oriented Software Testing (GOST), that complements and exploits goal-oriented analysis and design to derive test suites at all testing levels, from unit to acceptance. GOST provides a testing process model that brings the connections between goals and test cases explicit and a systematic way of deriving test cases from goal analysis. This can help discovering problems early, avoiding implementing erroneous specifications. In addition, GOST comes with a tool that supports test case generation, specification, and execution.

3.5 Simulation-Based Design Validation

Sierra et al. [35], *usable*: The IDE-eli (Integrated Development Environment for Electronic Institutions) simulation tools support the engineering of MAS as

electronic institutions. The SIMDEI tool, based on Repast, enables to animate and analyze specifications before deploying them, therefore facilitating the location of unexpected behaviours that may jeopardize critical applications. An agent skeleton is automatically generated from its specification, depending on the roles and interactions in which this agent may participate. This skeleton has to be completed by designers with decision-making mechanisms. The simulation process involves different populations of agents with different features that are able to act in a specified institution. The institution designer analyses results and may return to the design stage if results differ from the expected ones.

De Wolf et al.[8], *usable*: This paper proposes an empirical analysis approach combining agent-based simulations and numerical algorithms for analyzing the global behaviour of a self-organizing system. The initial values of macroscopic variables (those related to the properties that are studied) are supplied, a certain number of simulations are then initialized accordingly, simulations are then executed for a predefined duration, the average values of these variables on all the simulations are then given as results to the analysis algorithm, this latter processes these results to compute the next initial values. The algorithm also decides on the configuration of the next simulations (initial conditions, durations) and the cycle is repeated until the algorithm reaches its goal (for instance, converge towards a value, find a steady state).

Gardelli et al. [13], *usable*: Detection of agents having abnormal behaviours in a self-organizing/open MAS. The infrastructure is both based on TuCSoN (agents and artefacts) and principles coming from the human immune system. The behaviour of different scenarios is simulated using SpiM specifications (PI-calculus).

Fortino et al. [11], Cossentino et al. [6], *usable*: A simulation-driven development process is obtained by integrating state-chart-based simulation methodology for MAS with PASSI. Simulation methodology based on three phases: modelling, coding and simulation. Simulation is based on MASSIMO, a Java-based discrete-event simulation framework that enables validation and evaluation of the dynamic behaviours of agents (using execution traces) as well as the performance of the system (using evaluation parameters defined in an ad hoc way).

Bernon et al. [1], *in progress*: This paper proposes a model for a cooperative agent and a simulation tool for helping designers of self-organizing MAS. According to the AMAS (Adaptive MAS) theory, the collective function of these systems emerges from the cooperative interactions between agents. Agents have to always avoid, or detect and then repair situations that are against this cooperative social attitude. The proposed tool enables a designer to build a simplified/prototype of his target AMAS under SeSAM, to run simulations in order to get information about the situations that are non cooperative and not processed in the right way by the cooperative agents. The designer can then observe

the behaviour of these agents in order to verify whether their cooperative attitude is the expected one according to the emergent collective function obtained. He has then the possibility to manually improve behaviours by acting on what and how agents perceive, decide and act.

Sudeikat and Renz [36], *in progress*: The validation procedure consists in making hypotheses on the macroscopic observable MAS behaviour based on the MAS designs. These hypotheses are validated by tailoring simulation settings and analyzing the obtained results. Causal Loop Diagrams are used to express the intended application behaviour. An environment of execution is proposed to measure the correlations between state variables and check the presence of causal relations.

4 Open Issues

The above-described survey allows pointing out open problems, and solutions to some of them that have not been validated empirically, so defining the basis of a research agenda for MAS testing.

First, looking at the five testing levels we may notice that most of the available approaches concern the unit, agent and integration testing levels, leading to the observation that system and acceptance testing need further investigation.

Second, most of the approaches need to be consolidated and evaluated on realistic case studies to provide evidence of their usability.

Specific properties of MAS systems, such as autonomy and adaptivity, and more generally self-* properties[4], are challenging research on engineering and testing such software systems. Similar issues were previously pointed out in the Agentlink's research roadmap [21], quoting it *...techniques are needed to ensure that the system still executes in an acceptable, or safe, manner during the adaptation process, for example using techniques such as dependency analysis or high level contracts and invariants to monitor system correctness before, during and after adaptation.*

While addressing the above mentioned open issues, possible benefits from conventional software engineering testing's techniques are worth considering. For instance, extending to Multi-Agent systems research on metrics for evaluating software qualities and for defining test oracles for systems' properties seem particularly promising. Examples are recent works in the context of the definition of self-organisation and emergence mechanisms for achieving self-* properties, [16] and [32], which study a certain number of metrics for evaluating adaptivity in self-organizing systems. Such metrics could be proved useful when testing and validating self-organising and complex systems. Moreover, Mikhail et al. [30, 29] apply coupling and cohesion metrics to evaluate qualities of service-based systems and intend to apply a similar approach to define metrics for agents within service oriented architectures.

[4] Namely self-managing, self-configuring, self-healing, self-optimizing, and selfprotecting, as defined in the autonomic paradigm [17].

References

[1] Bernon, C., Gleizes, M.P., Picard, G.: Enhancing self-organising emergent systems design with simulation. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 284–299. Springer, Heidelberg (2007)

[2] Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N.R., Treur, J.: DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. Int. J. Cooperative Inf. Syst. 6(1), 67–94 (1997)

[3] Caire, G., Cossentino, M., Negri, A., Poggi, A., Turci, P.: Multiagent Systems Implementation and Testing. In: Proc. of the 4th From Agent Theory to Agent Implementation Symposium, AT2AI-4 (2004)

[4] Chella, A., Cossentino, M., Sabatucci, L., Seidita, V.: Agile passi: An agile process for designing agents. International Journal of Computer Systems Science & Engineering (2006)

[5] Coelho, R., Kulesza, U., von Staa, A., Lucena, C.: Unit testing in multi-agent systems using mock agents and aspects. In: SELMAS 2006: Proceedings of the 2006 International Workshop on Software Engineering for Large-scale Multi-agent Systems, pp. 83–90. ACM Press, New York (2006), http://dx.doi.org/http://doi.acm.org/10.1145/1138063.1138079

[6] Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., Russo, W.: PASSIM: A Simulation-based Process for the Development of Multi-Agent Systems. Int. Journal of Agent-Oriented Software Engineering 2(2), 132–170 (2008)

[7] De Wolf, T.: Analysing and Engineering Self-organising Emergent Applications, PhD thesis, Katholieke Universiteit Leuven (2007)

[8] De Wolf, T., Samaey, G., Holvoet, T.: Engineering self-organising emergent systems with simulation-based scientific analysis. In: Brueckner, S., Serugendo, D.M., Hales, D., Zambonelli, F. (eds.) Third International Workshop on Engineering Self-Organising Application, sUtrech, Netherlands, pp. 146–160 (2005)

[9] Dikenelli, O., Erdur, R.C., Gumus, O.: Seagent: a platform for developing semantic web based multi agent systems. In: AAMAS 2005: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1271–1272. ACM Press, New York (2005), http://dx.doi.org/http://doi.acm.org/10.1145/1082473.1082728

[10] Ekinci, E.E., Tiryaki, A.M., Cetin, O., Dikenelli, O.: Goal-Oriented Agent Testing Revisited. In: Proc. of the 9th Int. Workshop on Agent-Oriented Software Engineering, pp. 85–96 (2008)

[11] Fortino, G., Garro, A., Russo, W., Caico, R., Cossentino, M., Termine, F.: Simulation-Driven Development of Multi-Agent Systems. In: Workshop on Multi-Agent Systems and Simulation, Palermo, Italia (2006)

[12] Gamma, E., Beck, K.: JUnit: A Regression Testing Framework (2000), http://www.junit.org

[13] Gardelli, L., Viroli, M., Omicini, A.: On the Role of Simulations in the Engineering of Self-Organising MAS: The Case of an Intrusion Detection System in TuCSoN. In: 3rd International Workshop Engineering Self-Organising Applications, pp. 161–175 (2005)

[14] Gómez-Sanz, J.J., Botía, J., Serrano, E., Pavón, J.: Testing and debugging of MAS interactions with INGENIAS. In: Luck, M., Gomez-Sanz, J.J. (eds.) AOSE 2008. LNCS, vol. 5386, pp. 199–212. Springer, Heidelberg (2009)

[15] Gutknecht, O., Ferber, J., Michel, F.: Integrating tools and infrastructures for generic multi-agent systems. In: AGENTS 2001: Proceedings of the Fifth International Conference on Autonomous Agents, pp. 441–448. ACM, New York (2001)
[16] Kaddoum, E., Gleizes, M.-P., Georg, J.-P., Picard, G.: Characterizing and evaluating problem solving self-* systems. In: Dini, P., Gentzsch, W., Geraci, P., Lorenz, P., Singh, K. (eds.) Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns (Adaptive 2009), pp. 137–147. IEEE Computer Society, Los Alamitos (2009)
[17] Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
[18] Kidney, J., Denzinger, J.: Testing the limits of emergent behavior in mas using learning of cooperative behavior. In: Proceeding of the 2006 Conference on ECAI 2006, pp. 260–264. IOS Press, Amsterdam (2006)
[19] Lam, D.N., Barber, K.S.: Debugging agent behavior in an implemented agent system. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2004. LNCS (LNAI), vol. 3346, pp. 104–125. Springer, Heidelberg (2005)
[20] Lomuscio, A., Raimondi, F.: MCMAS: A Model Checker for Multi-agent Systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 450–454. Springer, Heidelberg (2006)
[21] Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing), AgentLink (2005)
[22] Moreno, M., Pavón, J., Rosete, A.: Testing in agent oriented methodologies. In: Omatu, S., Rocha, M.P., Bravo, J., Fernández, F., Corchado, E., Bustillo, A., Corchado, J.M. (eds.) IWANN 2009. LNCS, vol. 5518, pp. 138–145. Springer, Heidelberg (2009)
[23] Nguyen, C.D.: Testing Techniques for Software Agents, PhD thesis, International Doctorate School in Information and Communication Technologies - University of Trento (2008)
[24] Nguyen, C.D., Miles, S., Perini, A., Tonella, P., Harman, M., Luck, M.: Evolutionary testing of autonomous software agents. In: The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 521–528. IFAAMAS (2009)
[25] Nguyen, C.D., Perini, A., Tonella, P.: Ontology-based Test Generation for Multi Agent Systems. In: Proc. of the International Conference on Autonomous Agents and Multiagent Systems (2008)
[26] Nguyen, C.D., Perini, A., Tonella, P.: Goal-oriented testing for MASs. Int. J. Agent-Oriented Software Engineering 4(1), 79–109 (2010)
[27] Núñez, M., Rodríguez, I., Rubio, F.: Specification and testing of autonomous agents in e-commerce systems. Software Testing, Verification and Reliability 15(4), 211–233 (2005)
[28] Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons, Chichester (2004)
[29] Perepletchikov, M., Ryan, C., Frampton, K.: Cohesion metrics for predicting maintainability of service-oriented software. In: QSIC 2007: Proceedings of the Seventh International Conference on Quality Software, pp. 328–335. IEEE Computer Society, Washington (2007)
[30] Perepletchikov, M., Ryan, C., Frampton, K., Tari, Z.: Coupling metrics for predicting maintainability in service-oriented designs. In: Australian Software Engineering Conference, pp. 329–340 (2007)

[31] Poutakidis, D., Winikoff†, M., Padgham, L., Zhang, Z.: Debugging and Testing of Multi-Agent Systems using Design Artefacts. In: Multi-Agent Programming, pp. 215–258 (2009)
[32] Raibulet, C., Masciadri, L.: Towards evaluation mechanisms for runtime adaptivity: from case studies to metrics. In: Dini, P., Gentzsch, W., Geraci, P., Lorenz, P., Singh, K. (eds.) Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns (Adaptive 2009), pp. 146–152. IEEE Computer Society, Los Alamitos (2009)
[33] Serrano, E., Botia, J.A.: Infrastructure for forensic analysis of multi-agent systems. In: Hindriks, K.V., Pokahr, A., Sardina, S. (eds.) ProMAS 2008. LNCS, vol. 5442, pp. 168–183. Springer, Heidelberg (2009)
[34] Serrano, E., Gómez-Sanz, J.J., Botía, J.A., Pavón, J.: Intelligent data analysis applied to debug complex software systems. Neurocomputing 72(13-15), 2785–2795 (2009)
[35] Sierra, C., Aguilar, J.A.R., Noriega, P., Esteva, M., Arcos, J.L.: Engineering Multi-Agent Systems as Electronic Institutions. Novatica 170, 33–39 (2004)
[36] Sudeikat, J., Renz, W.: A systemic approach to the validation of self–organizing dynamics within MAS. In: Luck, M., Gomez-Sanz, J.J. (eds.) AOSE 2008. LNCS, vol. 5386, pp. 31–45. Springer, Heidelberg (2009)
[37] Tiryaki, A.M., Öztuna, S., Dikenelli, O., Erdur, R.C.: SUNIT: A unit testing framework for test driven development of multi-agent systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 156–173. Springer, Heidelberg (2007)
[38] Zhang, Z., Thangarajah, J., Padgham, L.: Automated unit testing for agent systems. In: 2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007), Barcelona, Spain, pp. 10–18 (2007)
[39] Zhang, Z., Thangarajah, J., Padgham, L.: Automated unit testing intelligent agents in pdt. In: AAMAS (Demos), pp. 1673–1674 (2008)
[40] Zhang, Z., Thangarajah, J., Padgham, L.: Model based testing for agent systems. In: AAMAS, vol. (2), pp. 1333–1334 (2009)