Toward an Embedded Multi-Agent System Methodology and Positioning on Testing

Camille Barnier, Oum-El-Kheir Aktouf, Annabelle Mercier and Jean-Paul Jamont
Univ. Grenoble Alpes, Grenoble INP, LCIS, F-26000 Valence, France

Email: camille.barnier@etu.isae-ensma.fr and {oum-el-kheir.aktouf,annabelle.mercier,jean-paul.jamont}@lcis.grenoble-inp.fr

***Abstract*—Like all systems, multi-agent systems need to be verified during the design cycle. But the test of multi-agent systems is more difficult than the test of classic software. Indeed, it implies to test more than agent functionalities: individual agent behavior, interaction between agents and global system need to be tested. Several testing methods compensate these difficulties, but do not cover all the aspects of multi-agent systems. In this paper, we compare software testing and multi-agent systems testing and particularly in embedded context. Then major multi-agent system testing techniques are analyzed, with the AEIO facets for multi-agent systems. Finally, we propose a strategy to conduct the testing activity for embedded MAS.**

***Keywords*—MAS Testing, Embedded Multi-Agent Testing, Interaction Protocol Testing**

I. INTRODUCTION

Agents are autonomous entities, which manage by themselves their own state and behavior. Agents can communicate with other agents and users; they also can interact with their environment, through interaction protocols. A **Multi-Agent System (MAS)**, is a society of autonomous agents, which evolve in cooperation in their environment to reach collectively a global objective. On the embedded systems field, the wireless techniques appropriation allowed these systems to be increasingly physically distributed and able to make decentralized decision. Decentralization is a hot topic in MAS research, and has allowed the emergence of the concept of **embedded MAS (EMAS)**. The embedded application of MAS can be found in various fields like distributed control [30], and collective robotics [31]. Agent-oriented software engineering (AOSE) proposes numerous methods to design MAS or EMAS [1], such as the PASSI method [2] and the DIAMOND design cycle [3]. Multi-agent research essentially considers design and analysis of MAS. The implementation and deployment phases are sparsely mentioned, so testing methods are sparsely developed [1].

In this paper, we will focus on the proposition of an embedded MAS testing strategy to guide the testing activity.

To present this strategy, the basic concept of MAS (section II), and the test in the context of MAS (section III) are presented. The difficulties of MAS testing will be pointed out (section V). A discussion concerning major testing methods used for MAS concludes this paper (section VI).

II. BASIC CONCEPTS OF MAS AND EMAS

In this section, we will focus on basic concepts of MAS and EMAS. To do that, we will present the agent and MAS concepts. Finally, the particularities of EMAS will be presented.

A. Agent

An agent is software or hardware entity. The more important aspect in an agent is the autonomy. This autonomy implies the agents to have the control on their behavior without intervention of another agent or an external user. Agents are flexible, which implies additional properties, like:

Reactivity: Reactivity implies that agents have continuously a link with their environment and respond at changes that occur.

Proactiveness: Proactive systems generate and reach their goals. Their behavior is not only event driven.

Social Interaction: Social systems can interact or cooperate with other systems.

B. Multi-Agent System

A MAS is a set of agents that share the same environment, and collectively respond to a global objective. MAS are decentralized systems, which can be described following the vowels decomposition AEIO [4], each vowel corresponding to a concept used to build the MAS:

Agent (A): gathers all elements for defining and constructing these entities. It concerns the agent's knowledge, its model and its architecture.

Environment (E): deals with elements necessary for the multi-agent system realization such as the perception of the environment and actions the multi-agent system can do on it.

Interaction (I): includes all elements needed for structuring the external interactions among the agents like agent communication language and interaction protocols.

Organization (O): allows managing agent groups in organizations determined according to their roles [5].

This facets decomposition (figure 1) shows that a MAS is composed by a society of agents. These agents can interact with each other, and act on their environment.

978-1-5386-2387-9/17 $31.00 © 2017 IEEE
DOI 10.1109/ISSREW.2017.57

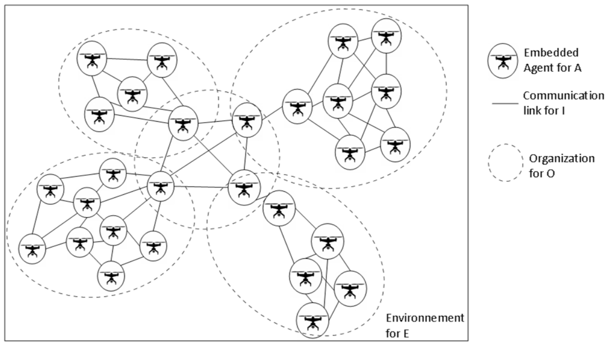

Figure 1: AEIO representation for MAS

The interactions between agents can lead to organization inside the MAS. The more important concept in MAS is the collective production of the agent society [1], [6], [29] as this is often the main goal of using MAS. This collective production comes from the fact that agents are aware of their environment and have knowledge of their neighboring agents. This definition of MAS is important in our work to prioritize the aspects of interactions in the MAS. In the following, we will focus on the main features of embedded MAS.

C. *Embedded MAS*

An embedded MAS is an embedded system, where agents are software and hardware mixed entities. Interactions are physical communications or physical actions and perception on real world environment.

An EMAS has to meet some properties coming from embedded systems [1]:

Reactivity: Embedded systems must react continuously to their environment.

Timeliness: Embedded systems need to keep time control, and their reactions need to be done in time.

Liveness: Embedded computing may not accept premature termination of programs. Deadlock must be avoided. The system must be able to adapt its behavior even by giving a not totally satisfying answer or a partial answer taking timeliness into account.

Power autonomy: Embedded software is often deployed on mobile small-size devices for which energy supply is not simple. To remain operational, embedded software must manage the tasks according to their priorities and available energy.

Safety management: As they are strongly related to physical devices, embedded systems can perform dangerous actions for humans or their environment. So, safety is a critical issue.

These properties must be satisfied by agents. But with the new expectations, raised by both the industry and the general users, applications often need a network of embedded systems [1]. In addition to embedded systems features presented above, a network of embedded systems implies other parameters, which are:

Complexity: Networked embedded systems can become very sophisticated, and building a global model is often impossible for these systems.

Concurrency: Embedded systems interact with a lot of physical processes, controlling several actuators and interacting with humans. They must simultaneously react to stimulus from a variety of sensors.

Heterogeneous aggregation: Heterogeneity is an intrinsic part of computation in embedded systems since embedded systems are a mixture of hardware and software. Other kind of heterogeneity can be found in embedded systems such as continuous/discrete time control techniques or synchronous/asynchronous event handling.

Interface integration: Embedded systems must integrate concurrent systems and devices. Because of the interoperability requirements, frameworks to ease the components coordination should be available.

Self-organization: Networked Embedded Systems need to frequently adapt their behavior according to their interactions and local tasks to achieve the global goal of the system.

Mobility: In case of wireless communication, the mobility of nodes in a Networked Embedded System, needs to be considered.

Integrity: Maintaining integrity of Networked Embedded System essentially lies in ensuring functional integrity through fault tolerance mechanisms.

The design aspect of this kind of systems has been developed [3], but not the strategy to conduct the testing activity. For testing activity, important properties in relation both with MAS and embedded systems influence the testing strategy. For example, autonomy of MAS may lead to non-deterministic behavior. Thus, the same test cases may not produce the same test results. In addition, an unexpected collective production may be generated and is difficult to detect. Mobility, energy supplies and reactivity considerations imply additional testing difficulties.

III. Analysis of embedded systems testing and MAS testing

The purpose of this section is to present embedded systems testing and MAS testing, to compare them and highlight the particularities of MAS testing and the particularities of embedded systems testing. The ultimate objective is to develop an embedded multi-agent system testing strategy.

A. *Embedded Systems Testing*

Testing activity for embedded systems is like software testing, with the same detail levels. The differences are on these detail levels. Section II above presented the major features of embedded systems that lead to intrinsic parameters within these systems. Considering such parameters increases the testing difficulty. As an example, the multi-thread programming design that is mostly used in implementing embedded systems needs time consideration, with scheduling validation and response time analysis.

B. *MAS Testing*

Fundamentally MAS testing [7] is not different from software testing, but it is more complicated and subtle than software testing. The complexity of MAS testing comes from the intrinsic characteristics of MAS. These characteristics are: (i) the *autonomy of agents* (ii) opening property of the system (iii) the *quantity of generated data* due to multiple sources of data

(iv) the *communication way by message passing* instead of method invocation.

Consequently, there is a need to conduct differently the testing activity, especially by introducing more detail levels than in software testing. These levels are: (i) Unit Testing (ii) Agent Testing (iii) Society Testing (iv) Acceptance Testing.

Unit Testing is the same as in software testing, and the Acceptance Testing corresponds to the System testing in software testing. The difference is in the integration testing which is cut in two distinct detail levels, described below.

1) Agent Testing

The Agent testing can be considered as the smallest testing unit on MAS. It can be considered like a module testing in software testing. This kind of test targets the internal functionalities of the agent, and the functionalities made available for the community of the agent. It can detect errors related to the agent's behavior and the integrity of the interaction between the tested agent and its environment. The main issue encountered at this testing level concerns the agent autonomy (and intelligence) that may lead to two tricky situations:

- a non-reproducible result for the same test cases;
- the refusal of the test request by the agent.

2) Society Testing

The society testing is a kind of integration test adapted for MAS. This kind of test should verify if the agent society works in cooperation like expected in the specification. This test may detect errors concerning incorrect communication, deadlock, content of agent's message, *etc*.

As a conclusion of section III, we can say that embedded systems and MAS have specific difficulties related to their testing activities. Consequently, a testing methodology for EMAS has to tackle all these difficulties.

We present in the next section the strategy to lead the testing activity for EMAS.

IV. EMBEDDED MULTI-AGENT SYSTEM TESTING

In this section, a strategy to conduct the testing activity for embedded MAS will be developed. This methodology (figure 2) focuses on 3 axes, which are:

- **Agent Testing:** activity to test an agent
- **Collective Features Testing:** testing a set or a subset of the agent's society
- **Acceptance Testing:** verifying the global conformity of the system

Two basic concepts must be defined before presenting the details of the proposed methodology: the concepts of society and organization.

Society: A society in MAS, is a population of agents and contains organizations.

Organization: An organization is a group of agents, which interact with each other's. An organization can be described like a communicating epicenter.

The main contribution of our work is to consider specific parameters related to EMAS, thus developing an adequate testing strategy for EMAS. In the following we will give details on the EMAS testing strategy (figure 2). To do that, the presentation of each axis will be done by detailing the several aspects which could be handled to cover these axes.

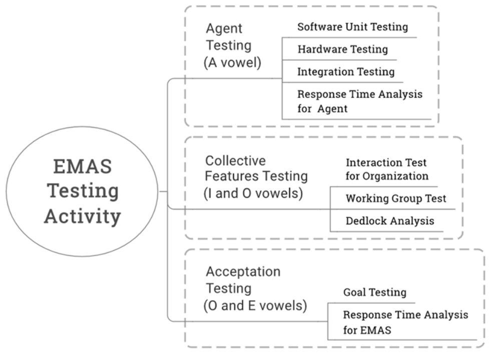
EMAS Testing Activity
Agent Testing (A vowel)
Software Unit Testing
Hardware Testing
Integration Testing
Response Time Analysis for Agent
Collective Features Testing (I and O vowels)
Interaction Test for Organization
Working Group Test
Dedlock Analysis
Acceptation Testing (O and E vowels)
Goal Testing
Response Time Analysis for EMAS

Figure 2: EMAS Testing Strategy

A. Agent Testing

Even in embedded MAS, agents are the smallest testable units. The objective of this testing level is to validate the agent as an individual entity. To achieve this goal, the software and hardware aspects of an agent are tested. This testing level focuses on 4 major items:

Software Unit Testing: This item will validate the software part of the agent by testing the functionality of the software and its structure. It can be conducted like the testing activity of classic software.

Hardware Testing: This item will validate the hardware part of the agent by testing the robustness of the hardware. It can be conducted like hardware testing.

Integration Testing: This item will focus on the integration of software part and hardware part in one entity, the agent. The behavior of the agent will be tested in this level, and we will check if the agent manages correctly received messages.

Response Time Analysis for the agent: This item focuses on the reactivity of the agent, and the estimation of the worst-case response time given a request.

When all these items are validated or evaluated, the agent is considered as validated. Currently, all the items presented above have methods to realize them. Additional objectives are to verify the reactivity, the timeliness, the liveness, the power autonomy and the safety management in individual agents.

The following step consists in analyzing in a small society of agents, enough representative of the global system, the functionalities and the integrity of the society. We call this step collective features testing.

B. Collective Features Testing

In this testing level, the objective is to test an organization, or some organizations, in a society of agents. To this end, we consider two kinds of interactions, by communication or without communication (*i.e.*, synchronization only). To realize this goal, we can split this testing level in 3 items, which are:

Interaction Test for Organization: focuses on the effect of a message sending on the behavior of the society organization.

Working Group Test: The objective of this item is to estimate the achievement of an organization's goal. This can be quantified as an *achievement goal rate*.

Deadlock Analysis: Focuses on interactions without communication, to avoid deadlock in society.

More specifically, this testing level has as objective to verify the concurrency, the heterogeneous aggregation, the self-organization and the safety management of the society. Most of these items have existing testing methods. After this step, we need to focus on the collective production and analyze the conformity of this production with the specification.

C. Acceptance Testing

It is the final testing level of embedded MAS, and the objective of this level is to evaluate the ability of the system to reach its goal, and its compliance with the specification. This detail level verifies the management of the complexity, the integrity of the system and the interface integration to communicate with users. To satisfy this level, two points should be developed:

Goal Testing: The objective is to estimate the achievement level of a given global goal within the system, to evaluate the realization of the collective production of the whole MAS.

Response Time Analysis for EMAS: In this item, the goal is to estimate if in the worst case the goal is reached within a time limit.

There are existing methods for the Goal Testing axis [8], but they consider specific design cycles. In our approach, we consider both hardware and software aspects of the EMAS to verify and validate its conformity with the specifications and imposed limits and constraints. After handling all the points of the proposed method, we can consider the EMAS as verified and the specification conformity as validated.

D. Synthesis of Testing Methods for Different Kinds of Systems

For all these testing strategies, we can see similarities on detail levels. Most of these testing strategies are articulated around 3 detail levels, which are unit testing, integration testing and acceptance testing. There is one exception for MAS in which integration testing is divided in two parts, agent testing and society testing. For each case, the number of test cases needed is an issue [9], and for MAS and EMAS this is even more crucial for scalability reasons. Difficulties of testing come from intrinsic parameters of systems. For embedded systems, the mobility of systems and energy storage are crucial. For SMA, the autonomy of agents with implies non-reproducibility and additional complexity due to decentralized decision making, lead to difficulties on testing. EMAS inherit all the characteristics of embedded systems and SMA, so testing such systems is a real issue.

V. MAS TESTING DISCUSSION

In this section, we present a survey of major MAS testing methods, synthesized on tables, and covering agent testing (A facet) and society testing (I and O facets). In these tables (Table 1 and Table 2), the context of each method is presented and related to the specific vowel of the AEIO model. Besides a brief description, the main research issues and limits of the presented methods are provided.

TABLE I. Survey of major MAS Agent testing methods

Name of method	**Using context**	**Description**	**Research issues**	**Limits**
Semantic Mutation Testing [10], [11]	Usable on rule based agents. A coverage testing. Agent testing. **level (A)**	Applies mutation testing on the program semantic, to verify if the mutant is found by the testing set	The enhancement on mutation generation and improvement on mutant finding (2015-2016)	Difficult to implement, when the semantic is too complex
Mutation Testing, [11], [12]	Usable in all cases, but frameworks have been developed for Java agents (eMuJava). Unit testing **level (A)**	Mutation testing technique	Future works will target artificial immunity system and particle swarm optimization [13]	Easier to implement than classical mutation testing methods
Model Based Testing, [14]	Usable on Prometheus agents. Unit testing **level (A)**	Generates a testing set which cover unit testing in the agent based on the agent's model	Model-based Testing is going to be industrialized	Limited to one type of agent
Goal Based Testing, [15]	The utilization of SEAUnit is necessary. Agent testing **level (A)**	Considers agents as the smallest testable unit on MAS for these methods.	In recent research issues, no improvement is suggested	Adapted on Tropos design cycle (V-cycle adapted for MAS)
Mock Agent Testing methods, [2], [16], [17]	Usable on FIPA agents developed by Agile methods. Agent testing **level (A)**	Analyze the behavior of one agent in communication with a Mock Agent	No improvement suggested	For FIPA agent, developed in eXtrem programming method
Automatic Failure Detection [18], [19]	Usable on rule based agent. Unit testing **level (A)**	This is a framework which facilitates the error finding, on cognitive agent	Research ended on this method (2016)	plugin in Eclipse
Execution Trace Method, [20]	Agent testing **level (A)**	Finds the execution trace to verify the agent's behaviors	The improvement on this method is on experimenting it on real projects	Difficulties to extract the execution trace due to non-reproducible execution

A. Agent Testing Discussion

A brief description of testing methods used to unit testing and agent testing for MAS that can be used in agent testing for EMAS is available on Table 1.

Lot of works was done concerning unit testing and agent testing. Indeed, these methods look like classic software testing, because the difficulties and particularities of MAS testing are not yet encountered. These testing methods focus mainly on software aspects of MAS. The embedded aspects for MAS were not really handled yet.

B. *Society Testing Discussion*

A brief description of testing methods used to integration testing and acceptance testing is presented in Table 2.

TABLE II. Survey of major MAS testing methods for Society testing

Name of method	Using context	Description	Research issues	Limits
Test by nets within nets [21]	Usable in all cases. This testing method is in society testing **level (I)**	Modeling the system by a Petri net, to automatize the test case generation	Future works concerning the integration of structural testing (2017)	Only functional testing methods
ACLAnalyser [22]	Usable in all cases. This framework is in society testing **level (I)**.	Analysis tools, which facilitate the analysis of communication route between agent's society	No improvement suggested	No recommendation found
MAS-DRIVE, [23]	Usable on agent society where decision making can be centralized. This method is in society testing **level (I)**.	Formalizes the messages between agents with CATN, and uses the agent manager to save messages in runtime and analyze them after execution	Future improvement consists on experimenting on complex existing protocols	Needs a pyramidal organization in MAS
Interaction Ontology Analysis, [24]	Usable on systems that have an agent communication language implemented. This method is in society testing **level (I)**.	Creates ontology based on the languages used in MAS, to define set of tests	Needs more experiments	Framework usable eCAT for JADE and JADEX agents
GOST, [25]	Usable on Tropos (V-cycle for MAS). This method is in society testing **level (I)**.	Uses the stakeholders and systems goal to generate test cases	Investigates in metrics to evaluate goal-oriented testing coverage.	The framework eCAT used in ontology analysis is used for this method
Agile BEAST [26]	Usable on Prometheus agents developed by Agile methods. This testing method is at acceptance testing **level (O)**	Tools which generate mock agents based on specification data base	Future works on improvement and adaptation on other types of agents and other entries than a data base	For Prometheus agents on Agile design cycle

Most existing works on society testing point out the reality concerning the difficulty to apprehend MAS testing on the Interaction (I) and Organization (O) axes of the vowels model (table 2). Methods on interaction testing are mainly limited to the aspects of message transferring. The interest of testing the interaction and the organization of MAS, is to verify if the transferred message is well interpreted and the expected behavior is realized, without creating deadlock within the agent society. This kind of test is realized in the Tropos design cycle [25]. A goal-oriented testing method exists also for acceptance testing [8]. The Tropos design cycle has some limitations as this technique considers that the MAS is closed and all agents known by the tester to generate test cases.

VI. FUTURE WORK

In the preceding sections, the difficulties of EMAS testing are highlighted and our first thoughts and work direction for developing a methodology for EMAS testing are presented. A survey of existing MAS testing approaches is also provided. In the rest of this paper, we develop our position on MAS and EMAS testing, based on an axis that is not developed yet in MAS and EMAS testing, which is interaction testing.

An interaction is to establish a dynamic relationship among two or more agents by sets of mutual actions. There is an interaction when an agent's behavior is influenced by other agents. There are 3 kinds of interactions:

- **Interaction without communication:** inference of other's actions.
- **Interaction by primitive communication:** finite set of signals. Complex coordination is hard to implement.
- **Interaction by message sending:** the communication between agents is done by messages sending

Communication in MAS

Communication is primordial for cooperative systems, and it is difficult to design a cooperative MAS if there is no communication system. This section covers the two last kinds of interactions, introduced above. Communications between agents are mostly structured and based on a typical model, called interaction protocol, which is characterized by specific sequences of messages. The need for designing a new language adapted to agent communication has led to the development of the Agent Communication Language (ACL). This language has evolved during the last years from a simple KQML syntax [27], with no formal semantic, and no infrastructure for agent management, to a formal semantic-augmented language (FIPA ACL [28]).

TABLE III. Syntaxes of KQML and FIPA ACL

KQML ACL	FIPA ACL
(KQML-performative :sender <word> :receiver <word> :language <word> :ontology <word> :content <expression> .....)	(inform :sender i :receiver j :content "message" :language l :ontology o)

These two syntaxes are similar but in FIPA ACL the agent handling is taken into consideration. This implies more complex execution (for example, a message that changes the state of the receiver can be used)

Nowadays, a Social ACL, which is an open dialect, with a formal semantic and an ontology for a better management of language heterogeneity, is used. However, no representative syntax can be defined for Social ACL. The interaction protocol needs more significant tests, these tests can be considered like a society testing. Our first approach is to extract the semantic of the message to determine if the behavior of an agent is in adequacy with this semantic and the goals of the agent, and measure the influence of the behavior of the agent on the society, in one hand. On the other hand, we aim to validate by a formal proof the self-organization protocol of MAS, using the WMAC model [5].

VII. Conclusion

In this paper, we presented a preliminary work concerning comparison between software testing and MAS testing, and an update of MAS testing methods survey. Moreover, a testing methodology for embedded MAS was presented. We can admit the major difficulties on MAS testing are due to the complexity of distributed systems, the autonomy of agents, and the message protocol used for communication between agents. Despite lot of works on MAS testing exist, they are limited mainly to unit testing and agent testing. A few papers concerning society testing can be found, and the interaction protocol in MAS is not really tested during the design. In future work, we plan to develop more in depth the proposed methodology of MAS testing, particularly in testing the interaction protocol in a society of agents.

Acknowledgment

The authors would like to acknowledge with thanks the Trust Chaire of Grenoble INP foundation for the financial support.

References

[1] J.-P. Jamont and M. Occello. Meeting the challenges of decentralised embedded applications using multi-agent systems. *Int. Journal of Agent-Oriented Software Engineering*, 5(1):22–68, 2015.

[2] H. Knublauch. Extreme programming of multi -agent systems. *1st int. joint conference on Autonomous agents and multiagent systems: part 2*, pages 704–711. ACM, 2002.

[3] J.-P. Jamont. *DIAMOND: Une approche pour la conception de systèmes multi-agents embarqués*. PhD thesis, Institut National Polytechnique de Grenoble 2005.

[4] Y. Demazeau and AC Rocha Costa. Populations and organizations in open multi-agent systems. *1st National Symposium on Parallel and Distributed AI*, pages 1–13, 1996.

[5] J.-P. Jamont, M. Occello, and A. Lagrèze. A multiagent approach to manage communication in wireless instrumentation systems. *Measurement*, 43(4):489–503, May 2010.

[6] A. Khenifar-Bessadi, J.-P. Jamont, M. Occello, M. Koudil, *et al*. About cooperation of multiagent collective products: An approach in the context of cyber-physical systems. In *Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2016 IEEE*, pages 19–24.

[7] Y. Kissoum *Test de systèmes multi agents*. PhD Thesis, Université Mentouri Constantine 2010.

[8] M. Kaur, B. Singh, and A. Kaur. Goal Oriented Acceptance Testing for Multi Agent System: V-Model Extension. 2nd Intl. Conf. on Advances in Electronics, Electrical and Computer Engineering, EEC 2013

[9] J. B. Goodenough and S. L. Gerhart. Toward a theory of test data selection. *IEEE Transactions on software Engineering*, (2):156–173, 1975.

[10] Z. Huang and R. Alexander. Semantic Mutation Testing for Multi-Agent Systems. In *Int. Workshop on Engineering Multi-Agent Systems*, pages 131–152. Springer, 2015.

[11] Z. Huang, R. Alex, and J. Clark. *Mutation Testing for Jason Agents*. In: Dalpiaz F., Dix J., van Riemsdijk M.B. (eds) Engineering Multi-Agent Systems. EMAS 2014. LNCS, vol 8758. Springer

[12] S. Savarimuthu and M. Winikoff. Mutation Operators for the Goal Agent Language. In *Engineering Multi-Agent Systems*, LNCS, pages 255–273. Springer, Berlin, Heidelberg, May 2013.

[13] M. B. Bashir and A. Nadeem. Improved Genetic Algorithm to Reduce Mutation Testing Cost. *IEEE Access*, 5:3657–3674, 2017.

[14] T. Zhang, J. Gao, O. Aktouf, and T. Uehara. Test Model and Coverage Analysis for Location-based Mobile Services. In *SEKE*, pages 80–86, 2015.

[15] E. E. Ekinci, A. Murat Tiryaki, O. Cetin and O. Dikenelli. Goal-Oriented Agent Testing Revisited. In *Agent-Oriented Software Engineering IX*, LNCS, pages 173–186. Springer, Berlin, Heidelberg, May 2008.

[16] M. A. Khamis and K. Nagi. Designing multi-agent unit tests using systematic test design patterns. *Engineering Applications of Artificial Intelligence*, 26(9):2128–2142, October 2013.

[17] R. Coelho, U. Kulesza, A. von Staa, and C. Lucena. Unit Testing in Multi-Agent Systems Using Mock Agents and Aspects. *2006 Int. Workshop on Software Engineering for Large-scale Multi-Agent Systems*, SELMAS '06, pages 83–90, New York, USA, 2006. ACM.

[18] V. J. Koeman, K. V. Hindriks, and C. M. Jonker. Automating failure detection in cognitive agent programs. *2016 Int. Conf. on autonomous agents & multiagent systems*, p. 1237–1246.

[19] V. J. Koeman, K. V. Hindriks, and C. M. Jonker. Using Automatic Failure Detection for Cognitive Agents in Eclipse (AAMAS 2016 DEMONSTRATION). In *Engineering Multi-Agent Systems*, LNCS, pages 59–80. Springer, Cham, 2016.

[20] D. N. Lam and K. S. Barber. *Debugging Agent Behavior in an Implemented Agent System*. ProMAS 2004. LNCS, vol 3346. Springer, Berlin, Heidelberg

[21] S. Kerraoui, Y. Kissoum, M. Redjimi, and M. Saker. MATT: Multi Agents Testing Tool Based Nets within Nets. *Journal of Information and Organizational Sciences*, 40(2):165–184, 2016.

[22] J. A. Bota, A. Lopez-Acosta, and A. G. Skarmeta. ACLAnalyser: A Tool for Debugging Multi-agent System. *16th European Conf. on Artificial Intelligence*, ECAI'04, pages 967–968, Amsterdam, 2004.

[23] D. Ancona, D. Briola, A. Ferrando, and V. Mascardi. MAS-DRiVe: a Practical Approach to Decentralized Runtime Verification of Agent Interaction Protocols. In *WOA*, pages 35–43, 2016.

[24] Cu D. Nguyen, A. Perini, and P. Tonella. Ontology-based Test Generation for Multiagent Systems. *7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems – Vol. 3*, AAMAS '08, pages 1315–1320, Richland, SC, 2008.

[25] C.d. Nguyen, A. Perini, and P. Tonella. Goal-oriented testing for MASs. *Int. Journal of Agent-Oriented Software Engineering*, 4(1):79–109, Dec. 2009.

[26] l. Carrera, C. A. Iglesias, and M. Garijo. Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. *Information Systems Frontiers*, 16(2):169–182, 2014.

[27] Y. Labrou and T. Finin. Semantics for an agent communication language. In *Intelligent Agents IV Agent Theories, Architectures, and Languages*, LNCS, pages 209–214. Springer, Berlin, Heidelberg, July 1997.

[28] Y. Labrou, T. Finin, and Y. Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems and Their Applications*, 14(2):45–52, 1999.

[29] Y. Randrianirina. Etude des productions collectives dans les systemes multi-agent. Master Thesis. Antananarivo University. 2015

[30] L. Nguyen, L. Lefèvre, D. Genon-Catalot and Y. Lami, Asynchronous information consensus in distributed control of irrigation canals. *21st IEEE Int. Conf. on Emerging Technologies and Factory Automation,* ETFA, Berlin, Sept. 2016

[31] R. Oikawa, M. Takimoto and Y. Kambayashi. Distributed formation control for swarm robots using mobile agents. *In Applied Computational Intelligence and Informatics (SACI)*, IEEE 10th Jubilee Int. Symposium pages 111-116. 2015