Structured System Test Suite Generation Process for Multi-Agent System

Zina Houhamdi

Software Engineering Department
Al-Zaytoonah University
Amman, Jordan

Belkacem Athamena

Software Engineering Department
Al-Zaytoonah University
Amman, Jordan

Abstract— In recent years, Agent-Oriented Software Engineering (AOSE) methodologies are proposed to develop complex distributed systems based upon the agent paradigm. The implementation for such systems has usually the form of Multi-Agent Systems (MAS). MAS' testing is a challenging task because these systems are often programmed to be autonomous and deliberative, and they operate in an open world, which requires context awareness. In this paper, we introduce a novel approach for goal-oriented software system testing. It specifies a testing process that complements the goal oriented methodology Tropos and reinforces the mutual relationship between goal analysis and testing. Furthermore, it defines a structured and comprehensive system test suite derivation process for engineering software agents by providing a systematic way of deriving test cases from goal analysis.

Keywords— MAS Testing; Goal-Oriented Testing Methodology; System Testing; Test Case Generation

I. INTRODUCTION

MAS are increasingly taking over operations and controls in enterprise management, automated vehicles, and financing systems, assurances that these complex systems operate properly need to be given to their owners and their users [10]. This calls for an investigation of suitable software engineering frameworks, including requirements engineering, architecture, and testing techniques, to provide adequate software development processes and supporting tools.

There are several reasons for the increase of the difficulty degree of testing and debugging multi-agent systems:

- Increased complexity, since there are several distributed processes that run autonomously and concurrently;
- Amount of data, since systems can be made up by thousands of agents, each owning its own data;
- Irreproducibility effect, which means that it is not ensured that two executions of the systems will lead to the same state, even if the same input is used. As a consequence, looking for a particular error can be difficult if it is not possible to reproduce it each time [8].

As a result, testing software agents and MAS seeks for new testing techniques dealing with their peculiar nature. The techniques need to be effective and adequate to evaluate agent's autonomous behaviors and build confidence in them. It is quite hard to verify that agents or MAS satisfy user requirements, behave correctly and are not malicious.

Testing a single agent is different from testing a community of agents. When testing a single agent a developer is more interested in the functionality of one agent and whether the agent operates for a set of messages, contextual inputs and error conditions. But, when testing a community of agents, the tester is interested in whether the agents operate together, are coordinated, and if message passing between the agents is correct [5].

Several AOSE methodologies have been proposed [7]. In terms of testing and verification, while some consider specification-based formal verification [2,4,12], other borrow Object-Oriented (OO) testing techniques, taking advantage of a mapping of agent-oriented abstractions into OO constructs [1,11]. However, a structured testing process for AOSE methodologies is still absent.

At the system level of MAS testing, we must test the intended emergent and macroscopic characteristics and/or the intended qualities of the system as a whole. Some initial effort has been consecrating to the validation of macroscopic MAS behaviors. Sudeikat and Renz proposed to use the system dynamics modeling notions for the MAS testing. These make possible describing of the expected, macroscopic observable behaviors that create from cyclic causalities structure. System simulations are then used to compute system state values in order to examine whether causalities are observable [14].

To the best of our knowledge, there is no work dealing explicitly with testing MAS at the system level, currently. In this paper, we propose a structured testing process that exploits the link between requirements and test cases following the V Model. We describe the proposed approach with reference to the *Tropos* software development methodology [9] and consider MAS as the target implementation technology.

The remainder of the paper is structured as follows. Section 2 recalls basic elements of the *Tropos* methodology and introduces related works. Section 3 discusses the proposed approach, a system testing process and test suite derivation. An example that illustrates how to derive test suites is presented in Section 4. Finally, Section 5 gives conclusion and describes our future work.

II. Background and Related Works

A. Tropos

Tropos is an AOSE methodology that covers the whole software development process. Tropos is based on two key ideas. First, the notion of agent and all related mentalistic notions (for instance goals and plans) are used in all phases of software development, from early analysis down to the actual implementation. Second, Tropos covers also the very early phases of requirements analysis, thus allowing for a deeper understanding of the environment where the software must operate, and of the kind of interactions that should occur between software and human agents. Tropos methodology spans five phases [2,9]:

a) Early requirements, concerned with the problem understanding by studying an organizational setting where the intended system will operate. The output of this phase is an organizational model which includes relevant actors (representing stakeholders) their respective goals (stakeholder’s objectives) and their interdependencies.
b) Late requirements, where the intended system is described within its operational environment, along with relevant functions (hardgoals) and qualities (softgoals). The intended system is introduced as a new actor. It appears with new dependencies with existing actors that indicate the obligations of the system towards its context as well as what the system expects from existing actors in its environment**.**
c) Architectural design, where the system’s global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies. More system actors are introduced. They are assigned to subgoals or goals and tasks (those assigned to the system as a whole).
d) Detailed design, where behavior of each architectural component is defined in more detail including specification of communication and coordination protocols. Agents' goals, beliefs and capabilities are specified in detail using existing modeling languages like UML or AUML, along with the interaction between them should occur between software and human agents.
e) Implementation, during this phase, the Tropos specification, produced during detailed design, is transformed into a skeleton for the implementation. This is done through a mapping from the Tropos constructs to those of a target agent programming platform, such as JADE [15]. Recent work on mapping Tropos goal model to JADEX programming platform is described in [13].

B. Goal types versus test types

This section presents different goal types and testing types. The relationships between goal types and testing levels are presented with reference to the process.

Test type: There are four types of testing: Agent testing, Integration testing, System testing and Acceptance testing [10]. The objectives and scope of each type is described as follows:

- Agent testing: The smallest unit of testing in agent-oriented programming is an agent. Testing a single agent consists of testing its inner functionality and agent’s capabilities to fulfill its goals and to sense and effect the environment.
- Integration testing: An agent has been unit-tested; we have to test its integration with existing agents. In some circumstances, we have to test also the integration of that agent with the agents that will be developed and integrated subsequently. Integration testing make sure that a group of agents and environmental resources work correctly together which involves checking an agent works properly with the agents that have been integrated before it and with the “future” agents that are in the course of Agent testing or that are not ready to be integrated. This often leads to developing mock agents or stubs that simulate the behaviors of the “future” agents.
- System testing: Agents may operate correctly when they run alone but incorrectly when they are put together. System testing involves making sure all agents in the system work together as intended. Specifically, one must test the interactions among agents (protocol, incompatible content or convention, etc.) and other concerns like security, deadlock.

- Acceptance testing: Test the MAS in the customer execution environment and verify that it meets the stakeholder goals, with the participation of stakeholders.

Goal type: Different perspectives give different goal classifications. For instance, classify agent goals in agent programming into three categories, namely perform, achieve, and maintain, according to the agent's attitude toward them [3]. We use a general perspective on goals, but not from a specific subject, to classify them based on the Tropos software engineering process.

Goals are classified into the following types according to the different phases of the process:

- Stakeholder goals: Represent stakeholder objectives and requirements towards the intended system. This type of goal is mainly identified at the early requirements phase of Tropos.
- System goals: Represent system-level objectives or qualities that the intended system has to reach or provide. This type of goal is mainly specified at the late requirements phase of Tropos
- Collaborative goals: Require the agents to cooperate or share tasks, or goals that are related to emergent properties resulting from interactions. This type of goal can be called also as group goal, and they often appear at the architectural design phase of Tropos.
- Agent goals: Belong to or are assigned to particular agents. This type of goal appears when designing agents.

C. Goal-oriented testing

The V-Model is a representation of the system development process, which extends the traditional water-fall model. The left branch of the V represents the specification stream, and the right branch of the V represents the testing stream where the systems are being tested (against the specifications defined on the left-branch). One of the advantages of the V-model is that it describes not only construction stream but also testing stream (unit test, integration test, acceptance test) and the mutual relationships between them.

Tropos guides the software engineers in building a conceptual model, which is incrementally refined and extended, from an early requirements model to system design artifacts and then to code, according to the upper branch of the V depicted in Figure 1. Tropos integrates testing by proposing the lower branch of the V and a systematic way to derive test cases from Tropos modeling results [10].

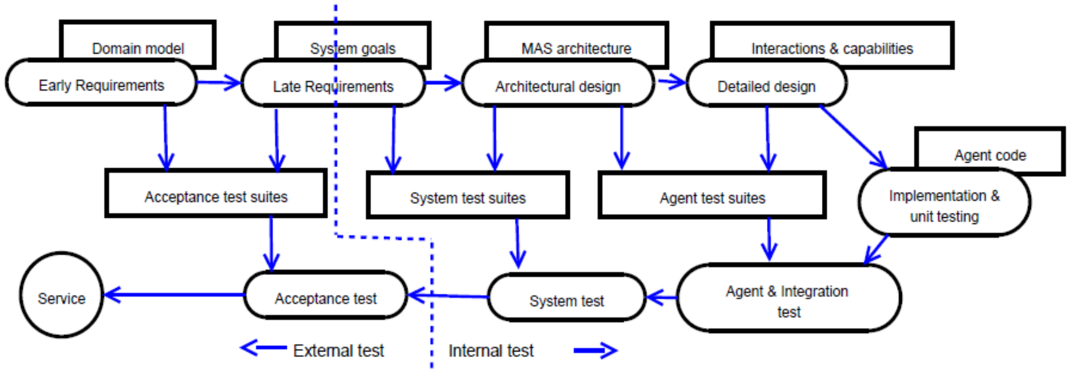

Figure 1. V-model of goal-oriented testing

Two levels of testing are distinguished in the model. At the first level of the model (external test executed after release), stakeholders (in collaboration with the analysts), during requirement acquisition time produce the specification of acceptance test suites. These test suites are one of the premises to judge whether the system fulfills stakeholders' goals. At the second level (internal test executed before release), developers refer to: goals that are assigned to the intended system, high-level architecture, detailed design of interactions and capabilities of single agents, and implement these agents.

In this work, we are interested by the first internal testing level exactly system testing. In next section, we present in details a testing process model and we discuss how to derive systematically test cases from goal models.

III. System Test Suite Derivation

The System testing builds on the previous levels of testing namely agent testing and Integration Testing. It focuses on testing the system as a whole. System Testing is a crucial step in Quality Management Process. In the Software Development Life cycle, System Testing is the first level where the System is tested as a whole. The System is tested to verify if it meets the functional and technical requirements. The System is tested in a context that closely resembles the production environment where the application will be finally deployed. The System Testing enables us to test, verify and validate both the Business requirements as well as the Application Architecture [6].

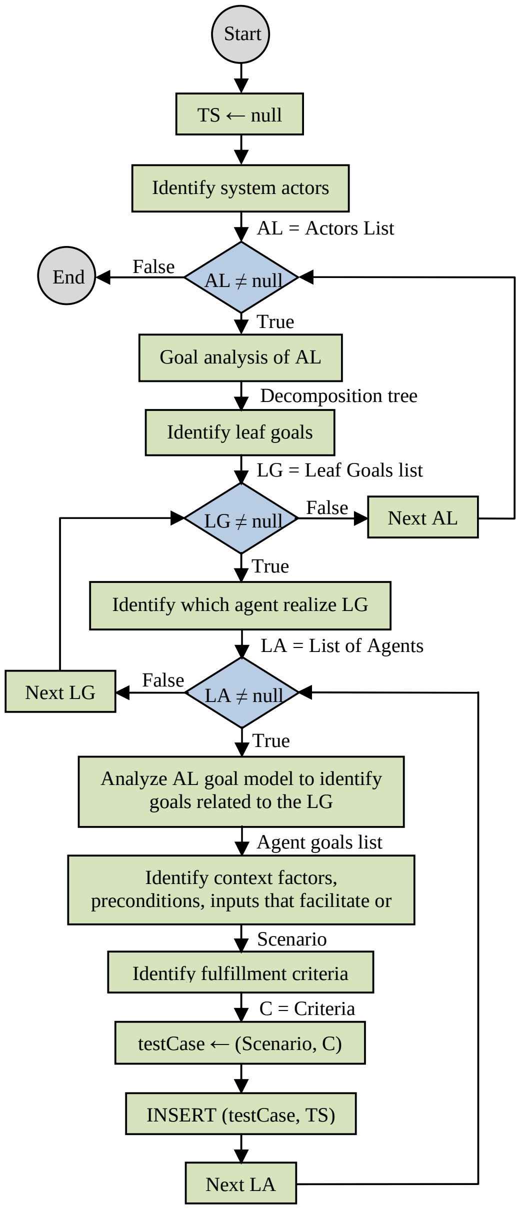

Figure 2. System tests suite derivation flowchart

System tests suite derivation occur in parallel with Late Requirement and Architectural Design. According to V model, the transition from Late Requirements to Architectural Design phase is a process which contains three steps:

- Identifying agents that realize the specified system actors,
- Allotting system actors' goals (called system goals) to agents' goals, and,
- Mapping system actors' dependencies to agents' dependencies and interactions.

At this stage, there are agents, their goals, roles, collaborative goals, agents' dependencies for goals and resources, the dependencies between agents and the context, regulations, constraints, and so on. System test suites should consider and take these artifacts in account.

Similar to acceptance test suite derivation where we take stakeholder actors' goals as foundation concepts, we use system actors' goals as foundations to create system test suites as they provide the system level objectives and requirements. When the system as a whole is built so that the system actors' goals (including functional hardgoals and quality softgoals) are fulfilled, it is ready to be passed to the customer for acceptance test.

System test suite derivation consists of the following steps (Figure 2): for each system actor, the goal model of the actor is analyzed to obtain its decomposition tree and then we filter the leaf goals. For each leaf goal *g* of a system actor, we must create a test suite to test the goal *g* accomplishment.

Test suite creation consists of five steps:

- Identifying which agent(s) realize(s) the goal *g*,
- Analyzing the goal model of each agent to identify goals related to the achievement of *g*,
- Identifying contextual factors, pre-conditions, inputs that facilitate or trigger *g*,
- Identifying fulfillment criteria for *g*,
- Creating a test suite having a set of test cases for the goal *g* that take inputs and criteria identified from previous steps.

In general, system actors can have more goals than those assigned to the system by stakeholder. So, the number of system test suites is usually higher than the number of acceptance test suites. Moreover, at this stage the system is designed, so more detailed information is available. As a consequence, we can reuse information from acceptance test suites, but much more details can be added, such as fulfillment criteria for goals and expected behaviors of involved agents.

The basic requirement for the system testing (i.e. all the derived test suites are passed) entails that all the goals of all the system actors are achieved or satisfied.

IV. CASE STUDY

To illustrate our approach, we introduce a multi-agent system that is composed of several cleaner agents working at a public garden. This software could be deployed on a physical platform composed of a set of moving Robots. Robots are in charge of keeping the garden clean and agents in the system have to collaborate to optimize their work and be nice with the visitors.

Following the guidelines of *Tropos*, we do the later requirement (Figure 3). There are two top softgoals that the stakeholder wants to achieve: SG1: minimize cleaning expense and SG2: please visitors. To reach these softgoals, three other sub-goals need to be fulfilled: G1: keep the garden clean, G2: Team work and G3: be polite. There could be more goals that the stakeholder wants to achieve, but we consider only these goals to keep the example simple and understandable.

Figure 3 shows the late requirements analysis for the cleaning Robot. The stakeholder delegates three goals SG1, SG2, and G1 to the multi-agent system under construction. At a high-level view, the system adds two hardgoals: G2: team work and G3: be polite in order to reach SG1, SG2, as required. Robots must achieve all the three hardgoals.

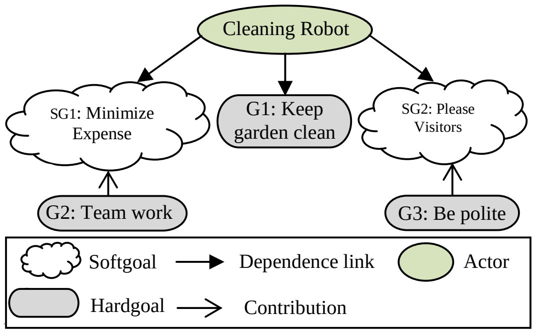

Figure 3. Late requirements for cleaning Robot

After the late requirements analysis, system actors become visible in the MAS architectural design. In this example, system actors are the cleaner agents. Goals of the system G1, G2, G3 are delegated to the agents. Figure 4 depicts the architecture system as a whole, showing set of cleaner agents. Notice that at the deployment time the number of agents will be determined by the number of available Robots. The mutual goal dependency G2 represents the fact that the group of agents will coordinate to better achieve the system goal SG1 and will reflect into individual agent goals.

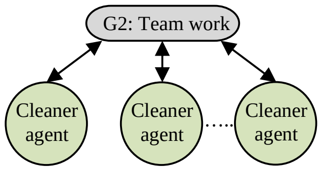

Figure 4. MAS architecture

The internal architectural design of the cleaner agent is described in Figure 5 which shows the architectural design of the cleaner agent. A number of goals and plans (tasks) are assigned to the agent. At the highest level there are four root goals: G1: keep the garden clean, G2: team work, G3: be polite and G4: maintain battery. G1, G2, G3 are delegated from the system, while G4 is the agent own goal to keep the agent alive. These goals are, then, decomposed into sub-goals. For instance, G4: maintain battery is AND-decomposed into two sub-goals G4.1: search charging station, and G: move to location. AND decomposition requires all sub-goals to be accomplished to obtain the satisfaction of their root goal. Finally, we add Plans to the cleaner agent design in order to achieve hardgoals. Besides the agents share resources, namely recharging-stations, bins, garbage and obstacles, and knowledge about them.

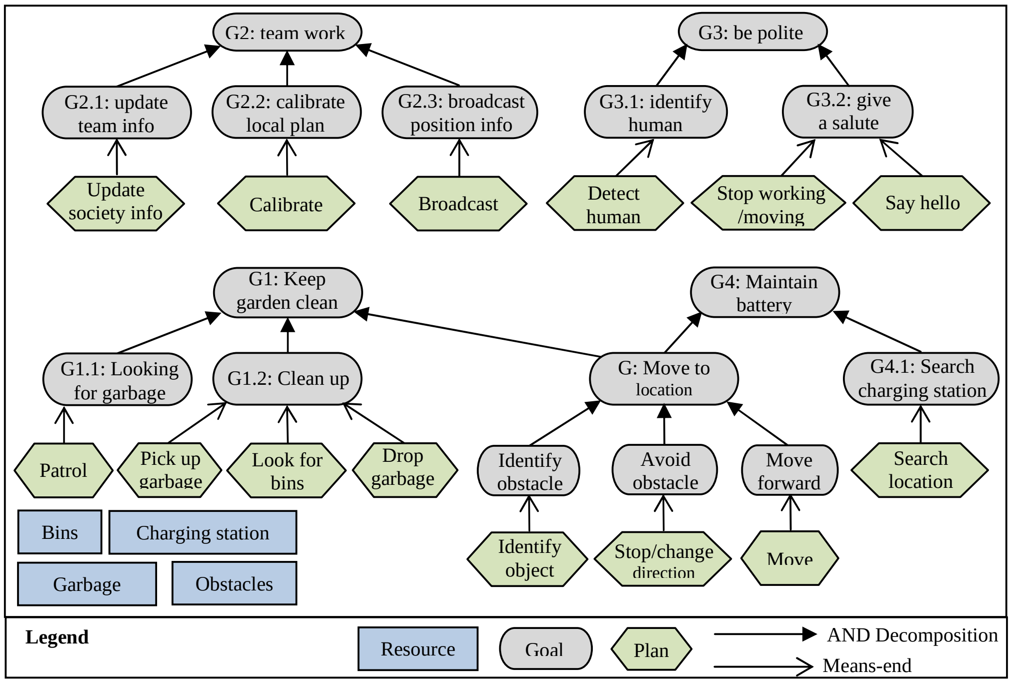

Figure 5. Cleaner agent architecture

To create system test suites, we start analyzing the late requirements for cleaning Robot (Figure 3) and understand that we need to test three goals: G1, G2, and G3. Next, based on the MAS architectural design and the cleaner agent architecture, we identify which agent goals to test and which resources of the context to configure. This identification can be straightforward based on goal identifiers, like in the case of the goal G2, G3, but it may need further analysis, when the transition from system actors' goals to agents' goals is implicit, like the goal G1. In this case, external information about problem domain described in analysis documents must be used. TABLE I, describes system test suites that we derived for Cleaning Robot. It shows the goal realization mapping between system actor and the cleaner agent.

TABLE I. TEST SUITES DERIVED FOR CLEANING ROBOT

Test suite	**System goal**	**Agent**	**Agent goal**
TS1	G1: Keep the garden clean	Cleaner Agent	G1: Keep the garden clean G4: Maintain battery G2: Teamwork
TS2	G2: Teamwork	Cleaner Agent	G2: Teamwork
TS3	G3: Be polite	Cleaner Agent	G3: Be polite

TABLE II describes some test cases that are created for each system actor's goal, G1, G2 and G3, accordingly. As apparent in TABLE II, the test case TC1.3 has an undefined test criteria (G*) with respect to the cleaner agent, because in the cleaner agent architectural design, there is no goal or plan that aims at adapting the behavior of the agent according to the amount of garbage and time. This is a clear indication that we have to further improve the design of the cleaner agent. For example, we can add a goal, G5: changing workload to the cleaner agent design, decompose it, and so forth. However, this example demonstrates that we can detect problems, such as incomplete specifications or implicit specifications, completely early. In fact, system test suites are used first to refine the system design and detect design problems early; and later, to perform system test.

TABLE II. TEST SUITE DERIVED FOR GOAL G1 (KEEP THE GARDEN CLEAN)

Test case	**Scenario**	**Criteria**
TC1.1	Given an actual area of the garden (A for short), garbage are placed at specified positions (p1; …; pn), the amount of garbage is (a1; a2; …; an), respectively. The Robot must clean this area.	The cleaner agent must fulfill two agent goals G1, G2 and maintain G4 with respect to the required time *t*.
TC1.2	Area A has garbage that is repeatedly thrown into in a random manner.	The cleaner agent must fulfill the agent goals G1, G2 in a periodical manner. It has to maintain the goal G4.
TC1.3	Depending on the time at the garden, area A can be more or less dirty: the amount of garbage is a function of time and position.	The cleaner agent must achieve the agent goals G1, G2, G4, so as to adapt its cleaning interval depending on the amount of garbage. This adaptation can be associated to a goal G*.

TABLE III. TEST SUITE DERIVED FOR GOAL G2 (TEAM WORK)

Test case	**Scenario**	**Criteria**
TC 2.1	The cleaner agents work together in area A.	The cleaner agents do not overlap their cleaning areas.
TC2.2	There is two recharging stations (X1;X2) in A.	There is no conflict with regard to the recharging station.

TABLE IV. TEST SUITE DERIVED FOR GOAL G3 (BE POLITE)

Test case	**Scenario**	**Criteria**
TC3.1	While the cleaning agents are moving or cleaning in area A, there are N humans moving in the area along different directions.	The cleaner agents stop moving/ working and nod their heads to say hello when they meet a human.

CONCLUSION

MAS system testing aims to test the system running in the target operational environment. As with the other testing levels, system test suites are purposed at two distinctive points.

- To test the expected emergent and macroscopic characteristics of the system as whole.
- To test the quality properties that the expected system must achieve such as performance, openness and fault tolerance.

This paper introduced a suite test derivation approach for system testing that takes goal-oriented requirements analysis artifact as the core elements for test case derivation. The proposed process has been illustrated with respect to the *Tropos* development process. It provides systematic guidance to generate test suites from modeling artifacts produced along with the development process. We have discussed how to derive test suites for system test from late requirement and architectural design. These test suites, on the one hand, can be used to refine goal analysis and to detect problems early in the development process. On the other hand, they are executed afterwards to test the achievement of the goals from which they were derived.

Specifically, the proposed methodology contributes to the existing AOSE methodologies by providing:

- A testing process model, which complements the development methodology by drawing a connection between goals and test cases, and,
- A systematic way for deriving test cases from goal analysis.

In this paper, we have presented a structured process for system test case generation with reference to the *Tropos* methodology. In the future work, we will investigate other testing type like integration testing and agents testing.

REFERENCES

[1] M. Cossentino, “From requirements to code with the *PASSI* methodology,” In Agent Oriented Methodologies, Hershey, PA, USA: Idea Group Publishing, Chapter IV, pp. 79-106, 2005.

[2] A. Dardenne, A. Lamsweerde, and S. Fickas, “Goal-directed requirements acquisition,” Science of Computer Programming, vol. 20, 1-2, pp. 3-50, 1993.

[3] M. Dastani, M. Riemsdijk, and J. Meyer, “Goal types in agent programming,” Proceeding 17th European Conference on Artificial Intelligence, pp. 220-224, 2006.

[4] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso, “Specifying and analyzing early requirements in *Tropos*,” Requirements Engineering, vol. 2, pp. 132-150, 2004.

[5] M. Gatti, and A. Staa, “Testing & debugging multi-agent systems: A state of the art report,” http://www.dbd.puc-rio.br/depto_informatica/06_04_gatti.pdf, 2006.

[6] Exforsys Inc, “System testing: Why? What? & How?” http://www.exforsys.com/tutorials/testing/system-testing-whywhathow.html, 2006.

[7] B. Henderson-Sellers, and P. Giorgini, “Agent-oriented methodologies,” Hershey, PA, USA: Idea Group Publishing, 2005.

[8] M. Huget, and Y. Demazeau, “Evaluating multi-agent systems: a record/replay approach,” Proceedings of IEEE/WIC/ACM International Conference, Intelligent Agent Technology (IAT 2004), pp. 536 – 539, 2004.

[9] J. Mylopoulos, J. Castro, “*Tropos*: A framework for requirements-driven software development,” Information Systems Engineering: State of the Art and Research Themes, Lecture Notes in Computer Science, Springer-Verlag, June 2000.

[10] C. Nguyen, A. Perini, and P. Tonella, “Goal-oriented testing for MAS,” International Journal of Agent-Oriented Software Engineering, LNCS 4951, pp. 58–72, 2008.

[11] J. Pavon, C. Sansores, and J. Gomez-Sanz, “Modelling and simulation of social systems with INGENIAS, International Journal of Agent-Oriented Software Engineering, vol. 2, 2, pp. 196-221, 2008.

[12] A. Perini, M. Pistore, M. Roveri, and A. Susi, “Agent-oriented modeling by interleaving formal and informal specification,” Agent-Oriented Software Engineering IV, 4th International Workshop, Melbourne, Australia, pp. 36-52, 2003.

[13] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos, “From capability specifications to code for multi-agent software,” In 21st IEEE/ACM International Conference on Automated Software Engineering, pp. 253-256, 2006.

[14] J. Sudeikat, and W. Renz, “A systemic approach to the validation of self-organizing dynamics within MAS,” Proceeding of the 9th International Workshop on Agent-Oriented Software Engineering, pp. 237-248, 2008.

[15] TILAB, “Java agent development framework,” http://jade.tilab.com/.

AUTHORS PROFILE

Dr. Zina Houhamdi received the M.Sc. and PhD. degrees in Software Engineering from Annaba University in 1996 and 2004, respectively. She is currently an Associate Professor at the department of Software Engineering, Al-Zaytoonah University of Jordan. Her research interest includes Agent Oriented Software Engineering, Software Reuse, Software Testing, Goal Oriented Methodology, Software Modeling and Analysis, Formal Methods.

Dr. Belkacem Athamena was born in Algeria. He received the M.Sc. and PhD. degrees in Computer Engineering and System/Software Modeling and Analysis from Annaba University in collaboration with UCL University, Belgium, in 1994 and 2004, respectively. He is currently an Associate Professor at the department of Software Engineering, Al-Zaytoonah University of Jordan. His research interests include System/Software Modeling and Analysis, Multi-Agent, Fuzzy Logic, Neural Networks, Petri Nets, UML, VVT, Formal Methods, Fault Diagnosis. He has published over 40 papers, chapter in books, and conferences.