Journal of Computer Science 7 (5): 690-697, 2011
ISSN 1549-3636


# Structured Integration Test Suite Generation Process for Multi-Agent System

Zina Houhamdi and Belkacem Athamena
Department of Software Engineering,
Faculty of Science and Information Technology,
Al-Zaytoonah University, Jordan

**Abstract: Problem statement:** 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). Testing of MAS 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. **Approach:** We introduce a novel approach for goal-oriented software integration testing. It specifies an integration testing process that complements the goal oriented methodology *Tropos* and strengthens the mutual relationship between goal analysis and testing. **Results:** The derived test suites from the system goals can be used to observe emergent properties resulting from agent interactions and make sure that a group of agents and contextual resources work correctly together. **Conclusion:** This approach defines a structured and comprehensive integration test suite derivation process for engineering software agents by providing a systematic way of deriving test cases from goal analysis.

**Key words:** Multi-Agent Systems (MAS), integration testing, test case generation, deriving test, Agent-Oriented Software Engineering (AOSE), Object-Oriented (OO), *tropos* methodology, architectural design, collaborative goals, detailed design

## 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 (Nguyen *et al*., 2010). 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 (Huget and Demazeau, 2004).

As a result, testing software agents and MAS seeks for new testing techniques dealing with their peculiar nature (Maamri and Sahnoun, 2007). 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, is coordinated and if message passing between the agents is correct (Gatti and Staa, 2006). The agent society test is a kind of integration test and the integration strategy depends on the agent system architecture where agents' dependencies are usually in terms of communications (but sometimes context mediated interactions could be present).

Several AOSE methodologies have been proposed (Henderson-Sellers and Giorgini, 2005). In terms of

**Corresponding Author:** Zina Houhamdi, Software Engineering Department, Faculty of Science and Information Technology, Al-Zaytoonah University, Jordan

testing and verification, while some consider specification-based formal verification (Dardenne *et al*., 1993; Fuxman *et al*., 2004; Perini *et al*., 2003), other borrow Object-Oriented (OO) testing techniques, taking advantage of a mapping of agent-oriented abstractions into OO constructs (Cossentino, 2008; Pavon *et al*., 2005). However, a structured testing process for AOSE methodologies is still absent.

In this study, we propose a testing process that exploits the link between goal analysis and test cases following the V Model. We describe the proposed approach with reference to the *Tropos* software development methodology (Mylopoulos and Castro, 2000) and consider MAS as the target implementation technology.

**Background and related works:**

***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 (Dardenne *et al*., 1993; Mylopoulos and Castro, 2000).

**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.

**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.

**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).

**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.

**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 (Bellifemine *et al*., 2007). Recent work on mapping *Tropos* goal model to JADEX programming platform is described in (Penserini *et al*., 2006).

**Goal types versus test types:** We present 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 (Nguyen *et al*., 2010). 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.

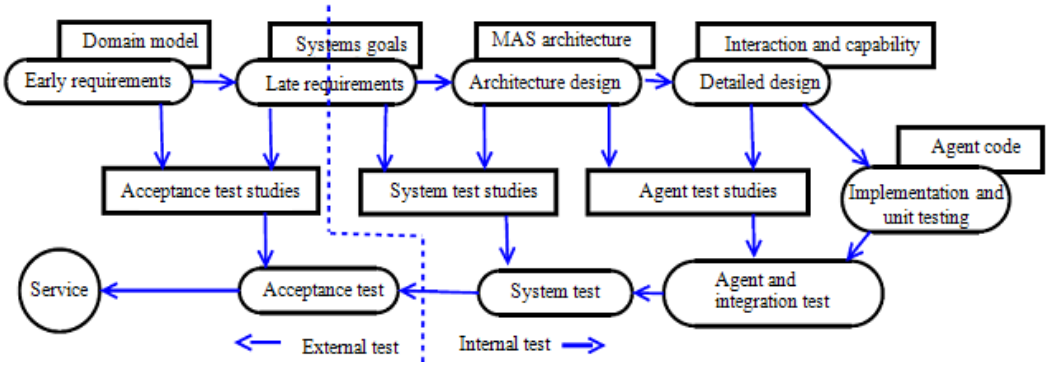


Fig. 1: V-model of goal-oriented testing

**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) and other concerns like security, deadlock (Houhamdi and Athamena, 2011).

**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 (Dastani *et al*., 2006). 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.

**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 Fig. 1. *Tropos* integrates testing by proposing the lower branch of the V and a systematic way to derive test cases from *Tropos* modeling results (Pavon *et al*., 2005).

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 study, we are interested by the internal testing level exactly integration testing. In next section, we present in detail a testing process model and we discuss how to derive systematically test cases from goal models.

## MATERIALS AND METHODS

The purpose of integration testing is to assure that agents work together correctly sharing tasks and resources to achieve collaborative or agent goals. MAS Integration testing consist of the following tasks:

- Tests the interaction of agents, communication protocol and semantics, interaction of agents with the context, integration of agents with shared resources, regulations enforcement
- Observe emergent properties, collective behaviors
- Make sure that a group of agents and contextual resources work correctly together

To acquire these objectives, we consider dependencies between agents for collaborative goals and dependencies between agents and resources. In fact, these dependencies are sources that lead to interactions, i.e. agent-agent and agent-context interactions.

We can use them to derive test suites that apply these dependencies and then evaluate the result of the interactions. Following the V model, Integration test suite derivation takes place once detailed design is completed, so that we can use the interaction protocol design.

The test suite derivation for collaborative goals, represented by agent-agent interaction, consists of the following steps (Fig. 2): In the system architectural design we describe a set of collaborative goals. For each of these goals we recognize agents that are involved, interaction scenarios, protocols and ontology. Then, we identify fulfillment criteria for the goal. Finally, for each scenario we can determine a test suite making use of data identified, i.e. agents, protocols, criteria and so on.

The test suite derivation for collaborative goals, represented by agent-context interaction, consists of testing their perception and influence capabilities. That is, we need to make sure that the agents under test are able to gather changes concerning the interested resources. We test whether they can take on such resources properly. The following steps guide us when deriving test suites for testing the agent-context interaction (Fig. 3): For each agent type in the system we list resources that the agent uses. Then, we describe set of interaction scenarios, access policies, protocols and other related factors. Finally, we define criteria for each scenario and create a test suite for it.

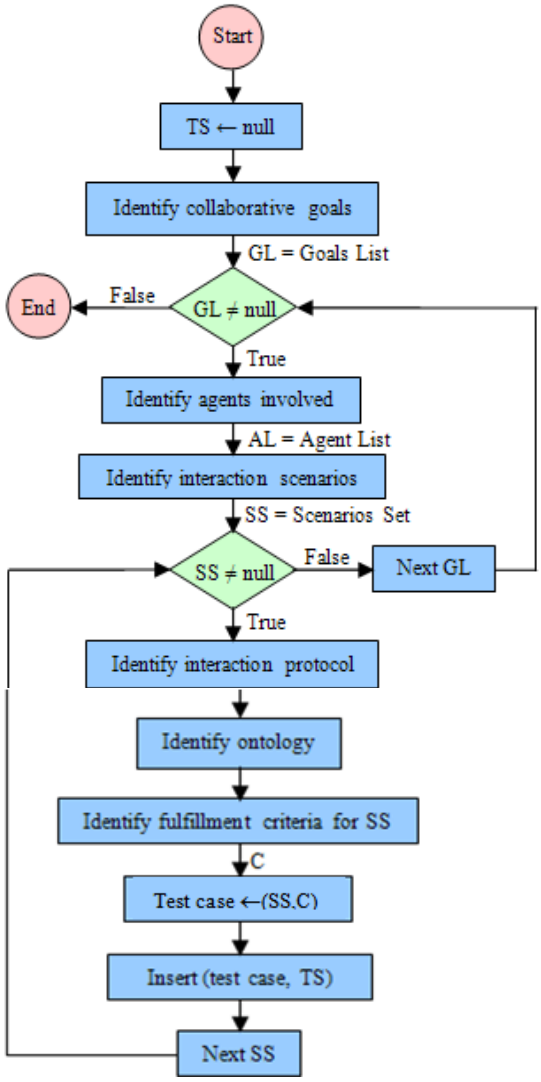


Fig. 2: Test suite derivation for Agent-Agent interaction flowchart

Testing for emergent characteristics resulting from agents interaction consists of verifying that all the involved agents respect predefined rules and that the expected group behaviors are actually noticed. Test suites created for this objective should pay particular attention on providing necessary context, so as to facilitate the agent interaction under test and on binding the rules that control the comportment of the agents under test.

In addition, test criteria for occurrence involve human observation and common viewpoint because different observers, with different viewpoints, may view the testing outputs, i.e. emergent characteristics, differently. So the definition of test criteria needs to consider these issues into account.

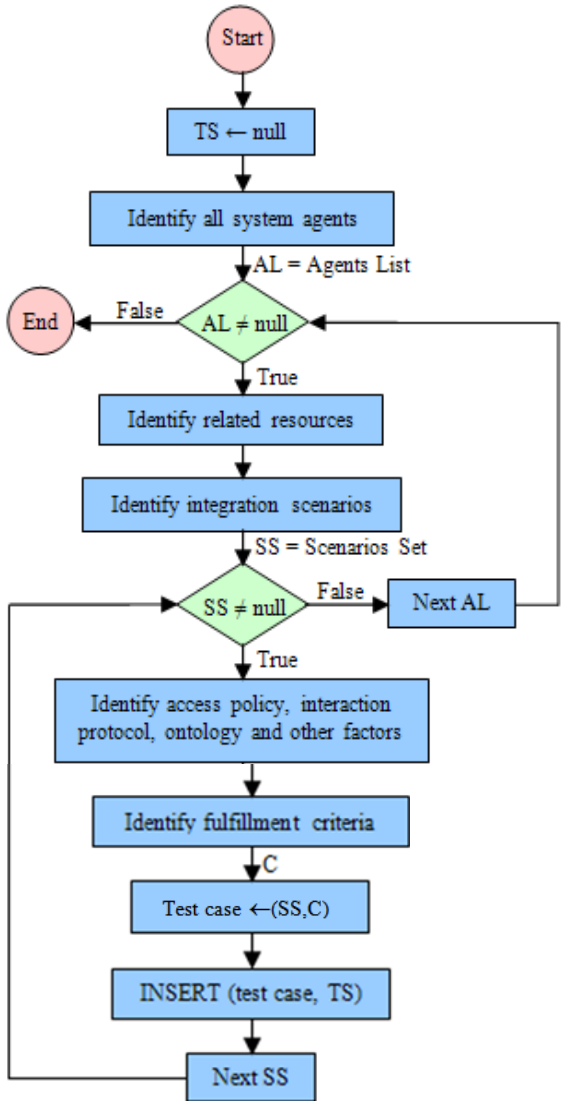


Fig. 3: Test suite derivation for Agent-Context interaction flowchart

As with the other testing levels, integration test suites are purposed at two distinctive points:

- To elaborate the interaction design and find a way of solving integration problems as early as possible. It is realized during the Detailed Design phase and integration test suite derivation.
- To test the integration of the implemented agents with one another and with the context, once these are available. It can be started as soon as an agent or a contextual resource is implemented. We do not need to wait until all the involved entities are implemented to start integration testing. We can use Mock agents, which simulate agents' behaviors.

## RESULTS

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. Following the guidelines of *Tropos*, we do the architectural design of the cleaning MAS (Fig. 4).

**G1:** Teamwork is a collaborative goal that involves all the cleaner agents. When we go further into the detailed design of the agent, in Fig. 5, we determine two interaction scenarios:

- One cleaner agent broadcasts information about its location.
- The agent receives a message broadcast from another cleaner agent.

Let's consider scenario 1, Fig. 6 shows the detailed design of the scenario: first, an agent sends a request to the Agent Management System (AMS) to get the addresses of other cleaner agents. Once a list of agents is returned, the agent broadcasts a message containing position information to all the agents in the list.

In order to test this scenario, we create the test case described in Table 1.

Based on goal models specified in the cleaner agent architectural design (Fig. 5), we identify four resources (garbage, bins, obstacles and recharging stations) that give rise to four integration test cases, following the steps described in agent-context interaction flowchart (Fig. 3). The test scenarios presented in Table 2 are abstract and we keep them so to make our example simple and understandable.

## DISCUSSION

At the MAS integration testing level, effort has been put in agent interaction to verify dialogue semantics:

- Padgham use design artifacts (e.g., agent interaction protocols and plan specification) to make available automatic identification of the source of errors detected at run-time (Padgham *et al*., 2005). A central debugging agent is inserted in the MAS to control the agent interactions. It receives a copy of each message exchanged between agents, during a specific conversation. Interaction protocol specifications corresponding to the conversation are sent back and then analyzed to discover automatically incorrect situations.

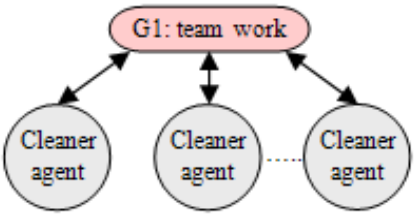


Fig. 4: MAS architecture

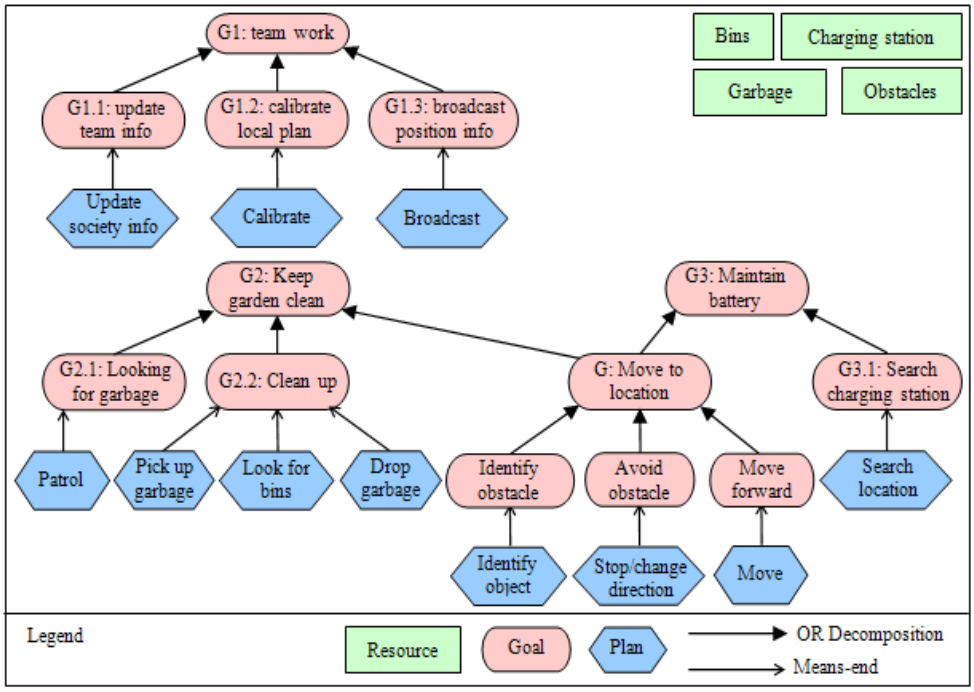


Fig. 5: Cleaner agent architecture

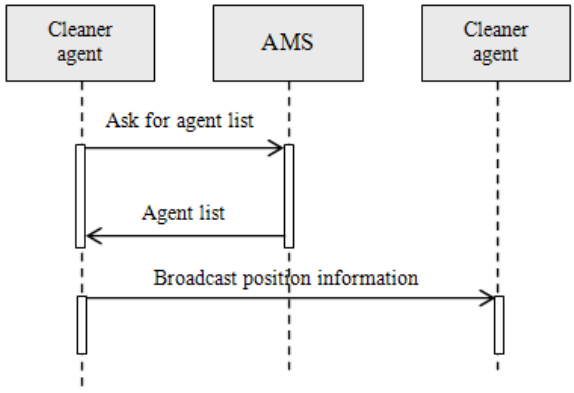


Fig. 6: Broadcast position information protocol

Table 1: Test case derived for broadcast position information

| Test case | Scenario | Criteria |
|---|---|---|
| TC1 | Instantiate two cleaner agents working together and monitor the communication between these two agents and between each of them and the AMS | The two agents register themselves with the AMS<br>The two agents send requests to the AMS<br>The two agents send messages to each other<br>The content of the messages is valid |

Table 2: Test case derived for G1 (teamwork)

| Test case | Scenario | Criteria |
|---|---|---|
| TC1 | Given an actual area of the garden (A for short) Cleaner Agents work together in area A | The agents do not overlap their cleaning areas |
| TC2 | There is two recharging stations (X1; X2) in A | There is no conflict with regard to the recharging station |
| TC3 | There is a bin in each area | The cleaner agent put the garbage in the nearest bin |
| TC4 | There is p obstacles in the area A | The cleaner agent must identify the objects and avoid obstacles by changing the direction |

- The ACL Analyzer tool runs on the JADE platform. It acquires all exchanged messages between agents and stores them in a relational database. This approach use clustering techniques to construct agent interaction graphs that assist the detection of omitted interaction between agents that are expected to communicate, unbalanced execution configurations, overhead data exchanged between agents. This tool has been improved with data mining techniques to apply results of the execution of large scale MAS (Botia *et al*., 2006).
- (Ekinci *et al*., 2009) view integration testing of MAS rather abstract. They considered system goals as the source cause for integration and apply the same approach for testing agent goals (unit according to their view) to test these goals.
- (Rodrigues *et al*., 2005) take benefit of social behavior, i.e., norms, rules, that prescribe permissions, obligations and/or prohibitions of agents in an open MAS to integration test. Information available in the specifications of these communications causes a number of assertions types, such as time to live, role, cardinality and so on. During test execution, a special agent called Report Agent will observe events and messages in order to generate analysis report afterwards.

In summary, most of the modern researches work on testing software agent and MAS focuses essentially on agent and integration level. Basic issues of testing software agents like message passing, distributed/asynchronous have been considered; testing frameworks have been proposed to facilitate testing process. However, there are still many points for further investigations, like:

- A complete and comprehensive testing process for software agents and MAS.
- Test inputs definition and generation to deal with open and dynamic nature of software agents and MAS.
- Test criteria: How to judge if an emergent property is correct? How to check the mutual relationship between macroscopic and agent behaviors.
- Reducing/removing side effects in test execution and monitoring because introducing new entities in the system, e.g., mock agent tester and monitoring agent as in many approaches can influence the behavior of the agents under test and the performance of the system as a whole.

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.
- Systematic way for deriving test cases from goal analysis.

## CONCLUSION

This study introduced a test suite derivation approach for integration 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 integration test from architectural and detailed design of the system goals. These test suites can be used to observe emergent properties resulting from agent interactions and make sure that a group of agents and contextual resources work correctly together.

In this study, we have presented a process for integration test case generation. In the future work, we will investigate other testing type like system testing and agents testing.

## REFERENCES


Bellifemine, F.L., G. Caire and D. Greenwood, 2007. Developing Multi-Agent Systems with JADE. 1st Edn., Willey, ISBN: 978-0-470-05747-6, p: 300.

Botia, J., J. Gomez-Sanz and J. Pavon, 2006. Intelligent data analysis or the verification of multi-agent systems interactions. Proceedings of the 7th International Conference of Intelligent Data Engineering and Automated Learning, Sep. 20-23, Burgos, Spain, pp: 1207-1214. DOI: 10.1007/11875581_143

Cossentino, M., 2008. From Requirements to Code with PASSI Methodology. In: Intelligent Information Technologies: Concepts, Methodologies, Tools and Applications, Sugumaran, V. (Ed.). Oakland University, USA., pp: 491-512. ISBN: 10: 1-59904-941-4

Dardenne, A., A. Lamsweerde and S. Fickas, 1993. Goal-directed requirements acquisition. Sci. Comput. Programm., 20: 3-50. DOI: 10.1016/0167-6423(93)90021-G

Dastani, M., M. Riemsdijk and J. Meyer, 2006. Goal types in agent programming. Proceeding of the 17th European Conference on Artificial Intelligence, Aug. 28-Sep. 1st, IOS Press, Amsterdam, Netherlands, pp: 220-224. DOI: 10.1145/1160633.1160867

Ekinci, E., M. Tiryaki, O. Cetin and O. Dikenelli, 2009. Goal-oriented agent testing revisited. Proceedings of the 9th International Workshop on Agent-Oriented Software Engineering, May 12-13, Springer-Verlag, Portugal, pp: 85-96. DOI: 10.1007/978-3-642-01338-6_13

Fuxman, A., L. Liu, J. Mylopoulos, M. Pistore and M. Roveri *et al*., 2004. Specifying and analyzing early requirements in *Tropos*. Requirements Eng., 9: 132-150. DOI: 10.1007/s00766-004-0191-7

Gatti, M. and A. Staa, 2006. Testing and Debugging Multi-Agent Systems: A State of the Art Report. http://www.dbd.pucrio.br/depto_informatica/06_04_gatti.pdf

Henderson-Sellers, B. and P. Giorgini, 2005. Agent-oriented methodologies. 1st Edn., Idea Group Inc., Hershey, PA, ISBN: 1591405815, pp: 413.

Houhamdi, Z. and B. Athamena, 2011. Structured system test suite generation process for multi-agent system. Int. J. Comput. Sci. Eng., 3: 1681-1688. http://www.enggjournals.com/ijcse/doc/IJCSE11-03-04-036.pdf

Huget, M.P. and Y. Demazeau, 2004. Evaluating multiagent systems: A record/replay approach. Intelligent Agent Technology. IAT 2004. Proceedings of IEEE/WIC/ACM International Conference Sep. 20-24, Beijing, China, pp: 20-24. DOI: 10.1109/IAT.2004.1343013

Maamri, R. and Z. Sahnoun, 2007. MAEST: Multi-agent environment for software testing. J. Comput. Sci., 3: 249-258. DOI: 10.3844/jcsp.2007.249.258

Mylopoulos, J. and J. Castro, 2000. *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. Berlin, 2068: 108-123. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.4590

Nguyen, C., A. Perini and P. Tonella, 2010. Goal-oriented testing for MASs. Int. J. Agent-Oriented Software Eng., 4: 79-109. DOI: 10.1504/IJAOSE.2010.029810

Padgham, L., M. Winikoff and D. Poutakidis, 2005. Adding debugging support to the Prometheus methodology. Eng. Appli. Artificial Intell., 18: 173-190. DOI: 10.1016/j.engappai.2004.11.018

Pavon, J., J. Gomez-Sanz and R. Fuentes-Fernandez, 2005. The INGENIAS Methodology and Tools. In: Agent Oriented Methodologies, Henderson-Sellers and Giorgini (Eds.). Idea Group, USA., pp: 236-276. ISBN: 1-59140-581-5

Penserini, L., A. Perini, A. Susi and J. Mylopoulos, 2006. From capability specifications to code for multi-agent software. Proceedings of the 21st IEEE International Conference on Automated Software Engineering, Sep. 18-22, IEEE Computer Society Tokyo, Japan, pp: 253-256. http://doi.ieeecomputersociety.org/10.1109/ASE.2006.38

Perini, A., M. Pistore, M. Roveri and A. Susi, 2003. Agent-oriented modeling by interleaving formal and informal specification, agent-oriented software engineering IV. Proceedings of the 4th International Workshop, July 15, Melbourne, Australia, pp: 36-52. DOI: 10.1007/978-3-540-24620-6_3

Rodrigues, L., G. Carvalho, P. Barros and C. Lucena, 2005. Towards an integration test architecture for open MAS. Proceedings of the 1st Workshop on Software Engineering for Agent-Oriented Systems, Oct. 3, Uberlância, Brasilia, pp: 60-66. http://www.les.inf.pucrio.br/seas2005/file/lRodrigues.pdf