Study of Multi-agent Testing Techniques and Future Research Direction

Mohammad Mottahir Alam, Asif Irshad Khan, Noor-ul-Qayyum and Abdullah Maresh Ali

Abstract— Multi Agents System (MAS) is seen as the necessary software paradigm for realizing massive open distributed systems, testing the MAS evolves challenging task, there are several reasons for the MAS testing to be challenging. In this paper we mainly reported state of art Multiagent based testing techniques and its challenges and listed, some future research direction for testing Multi Agents System (MAS) are also highlighted in this paper.

Index Terms— Agents, Multi-agent Systems , Unit Testing, Agent-Oriented Software Engineering , Test challenges , Model Driven Architecture, debugging

1 Introduction

IN recent years, agent-based systems have received considerable attention in both academics and industry. Agents are seen as the necessary software paradigm for realizing massive open distributed systems. A software agent is a computer program that perform tasks in pursuit of a goal in a dynamic environment on behalf of another entity (human or computational), possibly over an extended period of time, without continuous direct supervision or control, and exhibits a significant degree of flexibility and even creativity in how it seeks to transform goals into action tasks. A software agent is similar to a robot, but operates in cyberspace, on a computer network.

Macal and North [1] believe that ''There is no universal agreement on the precise definition of the term 'agent', although definitions tend to agree on more points than they disagree''. It seems very complicated to extract agent characteristics from the literature in a consistent and constant perspective, because they are utilized in different ways [2].

Brawshaw [3] states the following definition: Software agent is a software entity that functions continuously and autonomously in a particular environment, which may contain another agents and processes.

For instance, a number of experts take into consideration any sort of independent components (e.g. software, individual, etc.) an agent, while some others believe that a component's behavior needs to be adaptive in order to be considered an agent, where the term agent is reserved for components that can learn through their environments and change their behaviors accordingly [1]. Nevertheless, several common features exist for most agents [5] & [6]—extended and explained further by [6], [8] and [9].

The paper is organized as follows: sections II cover Agent and Multiagent concepts and its feature Section III Software Verification, Testing and Debugging techniques Section IV describes related work Section V listed Challenges in testing of Multi Agents System (MAS) VI some research directions for testing of Multi Agents System (MAS) and Section VII conclusion and future work.

2. Concepts of Agent and Multiagent

2.1 Agent

Russell and Norvig [4] define an agent as follows: "The concept of an agent is meant to be a tool for system analyzing, not an absolute classification where entities can be defined as agents or non-agents." From the literature review, following characteristics can be defined for a software agent:

- Autonomy: Agents are independent and autonomous units that are capable of information processes and exchanging them with other agents to independently make decisions. They are also capable of being interactive with other agents and this may not necessarily influence their autonomy [6],[10] & [11].
- Heterogeneity: Agents can exist and act as groups, but they are constructed through a bottom-up way and combinations of similar autonomous individuals.

Mohammad M. Alam is currently working as a Lecturer, Faculty of Engineering, King Abdul Aziz University, Jeddah, Saudi Arabia, E-mail: mohammad.mottahir@gmail.com

Asif I. Khan, Noor-ul-Qayyum and Abdullah M. Ali are currently working as Lecturers, Faculty of Computing and Information technology, King Abdul Aziz University, Jeddah, Saudi Arabia, and their E-mails are: {aikhan, nqayyum, ammali}@kau.edu.sa

• Mobility: The mobility of agents is particularly a practical characteristic for spatial simulations. Agents can move around the space within a model.
• Adaptation and Learning: Agents are flexible to be adaptive to produce Complex Adaptive Systems [12]. Agents can be designed to change their locations depending on their current state, following their designed memory [11].
• Activity: Agents have to be active since they perform independent impacts.

The following active features can be identified:

- Pro-active (i.e. goal-directed): Agents are often considered goal-directed elements, following goals to be accomplished with respect to their behaviors [5]. For instance, agents in a geographic environment can be designed to discover a set of spatial manipulations to achieve an aim within a certain limitation (e.g. time), while evacuating a building during an urgent situation.
- Reactive (i.e. perceptive): Agents can be developed to have a consciousness of their surroundings (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined) and respond in a timely fashion to changes that occur in it [5].
- Bounded Rationality: In social sciences, a dominant type of modeling based on rational-choice paradigm has to exist. Rational-choice models commonly assume that agents are perfectly rational optimizers with easy access to gathered information, foresight, and infinite analytical capability. These agents are therefore able to solve deductively complex mathematical optimization matters.
- Interactive (i.e. communicative): Agents communicate with other agents (and possibly humans) via some kind of agent-communication language (ACL).This is also called as "Social ability"[5].For instance, agents can enquire other agents and the environment within a neighborhood, searching particular attributes, with the ability to disregard an input which does not match a desirable threshold.

Therefore, a software agent should be autonomous or at least semi-autonomous. It can act on behalf of another entity that is not directly apparent to the "user" interacting with the agent (similar to the real-world agents). It may have some level of "intelligence" in order to deal with a dynamic environment in which the unexpected is the norm. Moreover ,a software agent may be "mobile" and move or be moved around the network, but a software agent may also be "static" and do all its work on one host computer on the network, including accessing resources which are on hosts other than the host on which the agent is executing.

Agent-based models consist of several interactive agents placed within a system. Relationships between the existing agents are formulated, linking agents to other agents within a system. Relationships can be specified in a number of ways, from simply reactive (i.e. agents only accomplish events when activated to do so by external stimulus e.g. behavior of another agent), to goal-directed (i.e. seeking a particular purpose). In some cases, the action of predefined agents can be programmed to occur synchronously (i.e. each particular agent executes events at each discrete time point), or asynchronously (i.e. agent reactions are planned by the actions of other agents and/or with reference to a predefined time) [13].

In the definition we saw that a software agent is a piece of software that is able to act autonomously in particular environment. Figure 1 from Wooldridge [14] illustrates how agent interacts with its environment.

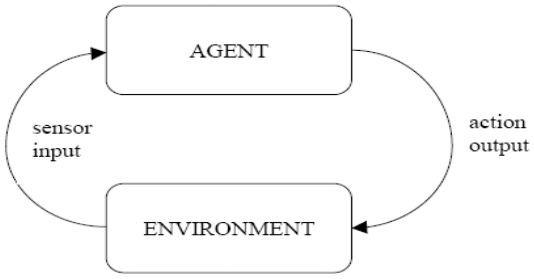

Figure 1- Software agent and its environment [14]

According to Castle and Crooks [6], "Environments define the space in which agents operate, serving to support their interaction with the environment and other agents. Agents within an environment may be spatially explicit, meaning agents have a location in geometrical space, although the agent itself may be static. For example, within a building evacuation model agents would be required to have a specific location for them to assess their exit strategy. Conversely, agents within an environment may be spatially implicit; meaning their location within the environment is irrelevant. For instance, a model of a computer network does necessarily require each

computer to know the physical location of other computers within the network." Therefore, software environments include operating systems, computer applications, databases, networks, and virtual domains.

2.2 MULTI-AGENT SYSTEM (MAS)

Agents are seen as the necessary software paradigm for realizing massive open distributed systems. But, as the complexity of software has increased, it has become harder to handle the complexity with single agent frameworks. Since the mid nineties, multi-agent systems have received widespread attention in many fields of science and engineering.

A multi-agent system can be characterized by a group of interacting, self-directed agents having varied sensory and motor abilities. Thus, a multi-agent system (MAS) is a computational environment in which individual software agents interact with each other, in a cooperative or competitive manner, and sometimes autonomously pursuing their individual goals, accessing resources and services of the environment, and occasionally producing results for the entities that initiated those software agents [22].

There are some works which address the problem of building confidence in the owners and users of agent-based systems with particulars techniques which we are going to describe in this work. Some of them are based on testing and monitoring, others are based on debugging, and others on simulation.

Moreover these works are still at very early stage. Actually formal methodologies provide validation tests that are applicable in very few and quite irrelevant cases though. The main reason of this lack applicability is that activities, which should assure that the program performs satisfactorily, are very challenging and expensive since it is quite complicated to automate them [2].

Agent Communication: Agent communication can be defined as the exchange of information between software agents. An agent needs some agent communication language to be able cooperate with other agents and react to its environment. Communication between agents encourages autonomy and also encourages the existence of societies of agents that are able to provide solutions to more complex problems.

Communication may be direct with one another or through an interpreter, communicate is usually took place through a language, Knowledge Query and Manipulation Language (KQML) is the most widely used agent communication language (ACL) [27]. Shared vocabularies of words are used in communication which is also known as Ontology. To ensure that two agents are communicating in the same language KQML uses ontologies.

Agent Cooperation: Co-operation among agents allows a community of specialized agents to pool their capabilities to solve large problems [15].
In multi-agent concepts side, there are various definitions for cooperation. (Gustasfon & Matson, 2003) defines cooperation as:
"The multi-agents working together for doing something that creates a progressive result such increasing performance or saving time" (Gustasfon & Matson, 2003).
(Changhong et al., 2002) definition of agent cooperation is as follows:
"One autonomous agent adopts another autonomous agent's goal. Its hypothesis is that cooperation only occurs between the agents, which have the ability of rejecting or accepting the cooperation" (Changhong et al., 2002).

Negotiation in Multi-Agents The multi-agent cooperation was defined in third definition as "The multi-agents working together for doing something" (Gustasfon & Matson, 2003). The vital member in multi-agents technology is group working; which needs a communication and negotiation between agents. Negotiation means "A key form of interaction that enables groups of agents to arrive at a mutual agreement regarding some belief, goal or plan" (Beer et al., 1998).

The negotiation between agents is implemented by different types, such as argumentation, protocols in the style of the contract net and auctions. The selection of negotiation type depends on the environment of problem, which has to be solved (D'Inverno et al., 1997).

Coalition/Cooperation in Multi-Agent

In "Cooperation structure" section, all types of the cooperation structures CATC, CCTA,CCTC and

CGE include agent coalition; so the agent coalition is the most important part in multi-agent cooperation systems. Operations of coalition have to configure legacy or foreign systems (Allsoop et al., 2002).

Agent coalition is special type of agents, which concentrate on the coordination and the communication among agents to collaboratively accomplish tasks. For an example, the stations work together as form of agent coalition. In this example, agents in this system are owners of power stations, groups of customers and coordinators. The objective of the Multiagent system is to derive effective with gainful coalitions under the fair play practice subject to the constraints also requirements of power generation and transmission.

The modern industries needed a more efficient approach to facilitate a stable searching for new partners, coalition formation and a fair system used to identify the contribution from each participant (Yen et al., 1998). Some game theory models can be borrowed to improve the theoretical foundation for the multi-agent system.

The recent coalition operations are effected by many factors, such as data overload, starvation of information, labour-intensive information collection and coordination. The agent-based computing presents a new promising approach to effective coalition operations; since this approach embraces the coalition environment's open, heterogeneous diverse dispersed nature (Allsopp et al., 2002).

Coordination relationship may be positive as well as negative. Positive coordination relationship benefits both the agent by working together to reach to their assigned goals for example suppose agents are coordinating to switched on a machine if they found machine is off any one agent can switch on the machine to accomplished the common goal (to switched on the machine) , while negative coordination relationship agents cannot complete their assigned task at the same time, for example agents are coordinating to print some assigned job, both of them issue print command but one agent command will be accepted others put in the printer queue.

3. Software Verification, Testing and Debugging

Testing is an activity in which a system or component is executed under specified conditions, the results are observed or recorded and compared against specifications or expected results, and an evaluation is made of some aspect of the system or component.
A test is a set of one or more test cases. The main aim of a test is to find faults.

An error is a mistake made by the developer misunderstanding something. A fault is an error in a program. An error may lead to one or more faults. When a fault is executed an execution error may occur. An execution error, error for short, is any result or behavior that is different from what has been specified or is expected by the user.

The observation of an execution error is a failure. Notice that errors may go on unnoticed and hence may play serious havoc with the remaining computation and use of the results of this computation. The longer the period of unobserved operation, the larger is the probability of serious damage due to errors that is due to unobserved failures.

There are two kinds of tests: static verification and dynamic validation. The former is based on code inspection or "walk through", symbolic execution, and symbolic verification. The later generates test data and execute the program. Figure 2 shows where static verification and dynamic validation tests occur during the software life cycle [8].

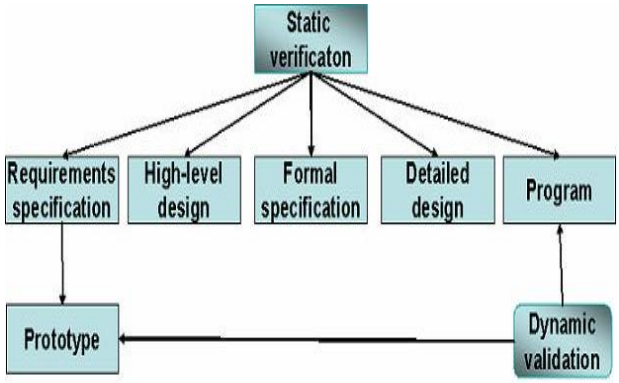

Figure 2 - Kinds of Testing [8]

There are several strategies for testing software and the goal of this survey is not to explain all of them. However, we will describe the main strategies found in literature ([8], [23]) for testing software which are related to some of the works presented in the fourth section. Here they are:

- Black-box testing: also know as functional testing or specification-based testing. Testing without reference to the internal structure of the component or system.
- White-box testing: testing based on an analysis of the internal structure of the component or system. Test cases1 are derived from the code e.g. testing paths.
- Progressive testing: it is based on testing new code to determine whether it contains faults.
- Regressive testing: process of testing a program to determine whether a change has introduced faults (regressions) in the unchanged code. It is based on re-execution of some/all of the tests developed for a specific testing activity.
- Performance testing: verify that all worst case performance targets have been met, and that any best-case performance targets have been met.

There are several types of tests. The most frequently performed are the unit test and integration test. A unit test performs the tests required to provide the desired coverage for a given unit, typically a method, function or class. A unit test is white-box testing oriented and may be performed in parallel with regard to other units. An integration test provides testing across units or subsystems. The test cases are used to provide the desired coverage for the system as a whole. It tests subsystem connectivity.
There are several strategies for implementing integration test:

(i) bottom-up, which tests each unit and component at lowest level of system hierarchy, then components that call these and so on;
(ii) top-down, which tests top component and then all components called by this and so on;
(iii) big-bang, which integrates all components together; and
(iv) Sandwich, which combines bottom-up with top-down approach.

The techniques and strategies presented in this section will appear in the approaches in the following section.

The main idea is to relate them with the works presented and classify them according to each strategy or technique.

4. Literature Review in the field of MAS Testing

In the Agent-Oriented Software Engineering (AOSE) methodologies, research works are mainly focused on the disciplined approaches to analyze, design and implement MASs [18]. Only a few of these methodologies define an explicit verification process. MaSE [19] and MASCommonKADs [20] methodologies propose a verification phase based on model checking to support automatic verification of inter-agent communications.

Now, with the increasing demand of agent-based systems, there is a growing need for the quality and correctness of the software-agents made. Unfortunately, testing remains a challenging activity where a systematic approach to testing multi-agent system is still missing.

Desire [21] proposes a verification phase based on mathematical proofs - the purpose of this process is to prove that, under a certain set of assumptions, a system adheres to a certain set of properties. Only some iterative methodologies propose incremental testing processes with supporting tools. These include: PASSI/Agile PASSI [22], AGILE [23].

[19] Proposed a new approach based on a simple testing framework called PASSI (Process for Agent Societies Specification and Implementation) which lets developers build a test suite effortlessly in a cheap and incremental way. It provides a unifying application model and a partial implementation of it, trying to support the developer in creating and executing tests in a uniform and automatic way.

They aim to reduce time and cost when developing MAS, guarantee quality assurance, and provide automatic activities which should assure that the program performs satisfactorily.

The PASSI framework is built on top of JADE and it allows developers to create tests at different levels (hierarchical approach) simply acting as a support for running tests and visualizing results. The framework is based on a two-level model as shown in figure 3. At the first level they identify the agent as an atomic entity. In order to check the correctness of the activities carried out by a single agent a number of different cases must be tested.
This leads us to the second level where they identify specific agent tasks. There is a "test-agent" which performs the set of tests related to all the

capabilities of a given agent. Tests on specific tasks, on the other hand, will be referred to as "task-test". In order to reflect the two-level model, the following classes are provided:

- Test class, representing the test of a specific task of an agent.

- TestGroup class, representing the group of all the agent tests. It is basically a collection of Test objects. The list of task-tests to be included in a TestGroup is described in an XML file.

In general, all test methods in a TestGroup share the same fixture, which consists of objects and anything else needed to perform the test.

A Test or a TestGroup is executed by a tester agent i.e. an agent that extends the TesterAgent class. Each tester agent has a behavior, an extension of the TestGroupExecutor class, which is in charge of getting the group of tests to be executed and for each test adds the corresponding behavior to the tester agent scheduler.

The list of all the agent-tests that can be tested and the list of task-tests to be performed for each of them are described by means of XML files. There is a single main XML file that contains the list of all the agent-tests of the application and one XML file for each agent-test that contains the list of task-tests to be executed. Developing an agent-test means therefore developing a new tester agent in charge of the group of task-tests described in the associated xml file.

Finally the utility class Logger provides methods to create logs. By extending this class it is possible to create sophisticated loggers in order to provide reported information in more suitable formats. To date, reported information can be displayed in a graphical user interface (where very essential information is shown), written to a text file, printed to the standard error or organized into web pages.

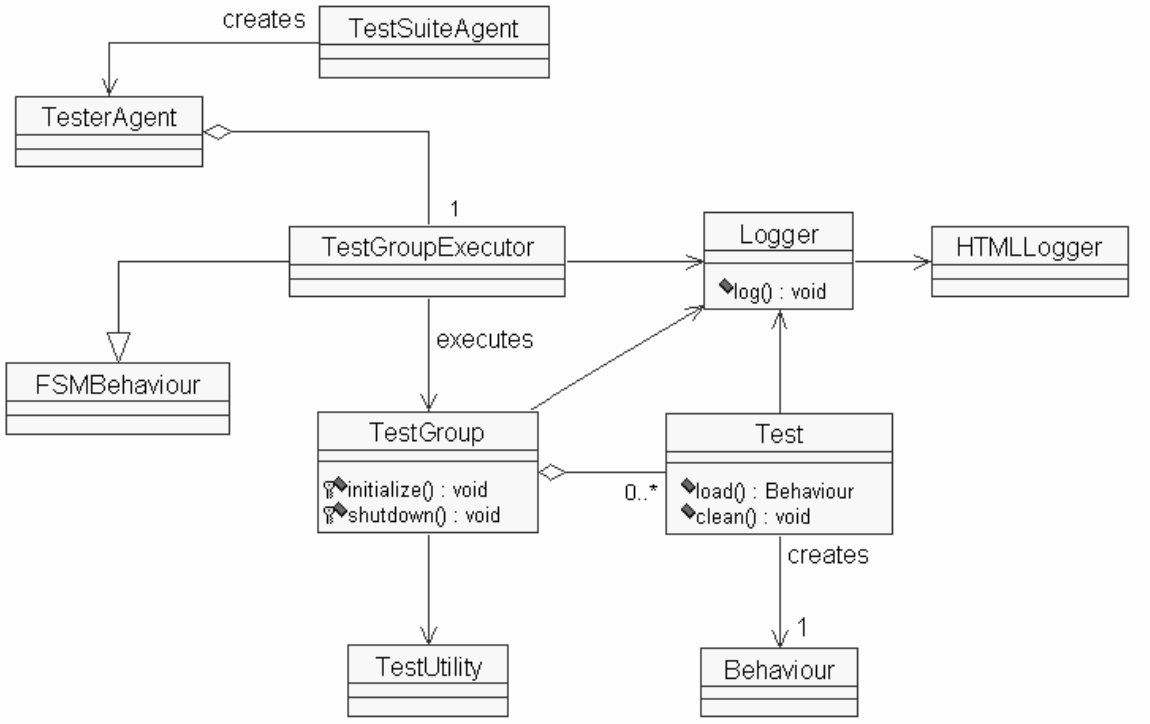

Figure 3 - PASSI -Test Framework main classes [19]

A single test and group of tests can be executed by simply launching the corresponding tester agent. A more convenient way of performing them is by means of the TestSuiteAgent, an agent that provides a valuable graphical interface to run tests. When a test or a group of tests are launched the TestSuiteAgent creates the proper tester agent and delegates to it the execution of the tests. During the testing activity the tester agent will send FIPA ACL messages to the TestSuiteAgent, informing it about the test outcomes and giving eventually detailed information concerning the causes of failure.

The TestSuiteAgent, as stated before, provides a graphical interface to run tests, by means of which it is possible to:
(i) View information related to the agent-tests and all the task-tests they include;
(ii) Select and load the tests to be executed;
(iii) Execute all agent-tests of the list in sequence and produce a final report indicating, for each agent-test, the number of task-tests which have passed and failed and the corresponding causes of failure.

The author [23] proposes an approach called MadKit which focuses on a specific kind of testing called the Record/Replay mechanism [24] used in regression testing. The Record/Replay mechanism is a test performed during the execution of the system either in simulation or in production. It is realized through system inspection. The record phase records actions in the system (memory, environment, data update, messages, etc.).

When an error occurs in the system, designers have a system that they can play and replay until they found the error and fixed it. They also compare model checking and testing, and say that contrarily to the former, testing checks that agents behave properly rather than agents are proved correct. In their approach, the Record/Replay mechanism is coupled to testing via post-mortem analysis. It uses the events and data stored during the record phase and checks properties without re-executing the system.
The author [25] in his research proposes an approach for integration tests in open multi-agent systems. This approach supports the creation of test cases based on the information provided by the definition of system rules. They propose to use XMLaw, which is a language for the specification of agents' interactions' regulation in open multi-agent systems. In open multi-agent systems, agents must obey social conventions in order to maintain predictable integration.

Usually, these social conventions are hard coded, leading to unsuitable systems. A solution to hard coded conventions is separate the system's social convention into a separate module insuring agents compliance. This technique is called law enforcement.

In [26],a framework for agent oriented testing based on the V-Model is proposed .The V-model is extended by incorporating the characteristics of the agent perspective approach.

The authors [27] in his paper propose an approach to verify the correctness of execution scenario in a multi-agent system. In this approach, scenarios are specified by Protocol Diagrams in AUML (Agent Unified Modeling Language) .

The pre and post conditions of the scenarios are formalized and an extension property class in JPF (Java PathFinder) model checker is defined to verify if the execution of scenarios satisfies their constraints. This approach has been illustrated by using a well-known scenario of a book trading multi-agent system.

The author [28] in his paper described the ongoing works to develop a systems modeling approach to allow design-time system models to be reused by an autonomous system at runtime. He identified the properties associated with the engineering of autonomous systems that differentiate them from other types of complex system.

A framework to support the verification and validation of aspects of autonomous systems at runtime is then presented which uses the principles of MDA (Model Driven Architecture) to underpin it, and discussed the rationale behind its structure., and we develop a specific aspect of this framework – a run-time Computation Independent Model (CIM), using a language from the automated planning domain, the Planning Domain Definition Language (PDDL).

The author [29] proposed a testing model for multi-agent systems which classifies the test techniques into the five dynamic test levels: unit integration test, agent acceptance test, agent integration test, system test and user acceptance test. The test processes that proposed by the International software testing qualification board (ISTQB) are extended and modified to address the properties of a comprehensive test process in AOSE.

The proposed process is divided by introducing four sub-processes: Test planning and control, Test analysis and design, Test implementation and execution and Test evaluation exit criteria and reporting. These sub-processes contain specific activities, metrics and tangible input and also output artifacts. The preferred agent-oriented test methods are employed in designing the test cases and in executing the tests sub-processes.
The authors [30] presented their studies of formal verification of multi-agent system using model checking approach. They have utilized model checking tool in order to execute the formal

verification procedures based on a particular basic theory to verify certain kind of properties of requirement specifications.

We show an example of how model checking tool could support the verification of Universiti Teknologi Malaysia (UTM) multi agent online application system and conclude that the propose model checking approach will benefit multi agent system.

They extended the generic model checking procedures i.e. specification, modeling, and verification by proposing model checking cycle (MCC) in which model checking procedures are done in cycle so that the informal and formal requirements specifications and system modeling can be further improved and refined as the verification is performed. Figure 4 below shows the extended model checking processes cycle adapted from quality management of industrial practice [6].

The model checking is implemented by executing multiple stages of processes organized into four phases of MCC processes flow as shown in Figure 4. The first phase is the informal specification and modeling of system requirements. Next, the second phase is the formal specification of properties and modeling of system.

The output of the informal and formal specification and modeling are used to execute automated verification using model checking tools in the third phase of MCC. Finally, the output of the verification phase is analyzed in the fourth phase and the analysis will be used to improve the next implementation cycle.

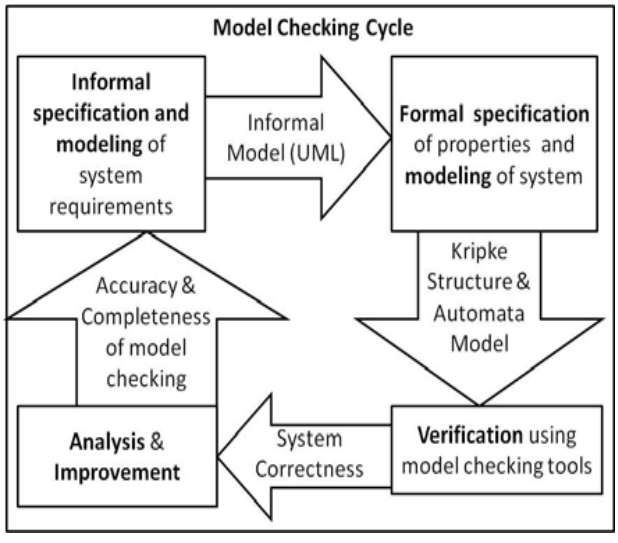

Figure 4- Proposed model checking cycle (MCC)

5. Challenges in testing of MAS

Testing the MAS is a challenging task. There are several reasons for the multi-agent system testing to be challenging:

- In multi-agent software, several distributed processes run autonomously and concurrently; which makes MAS to be complex and it's testing very challenging.
- Amount of data, since systems can be made up by thousands of agents, each owning its own data;
- Irreproducibility effect, since we can't ensure 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 impossible to reproduce it each time [22].
- They are also non-deterministic, since it is not possible to determine a priori all interactions of an agent during its execution.
- Agents communicate primarily through message passing instead of method invocation, so existing object-oriented testing approaches are not directly applicable.
- Agents are autonomous and cooperate with other agents who implies that they may run correctly by themselves but incorrectly in a community or vice versa.

Agents' characteristics such as autonomous behavior, pro-activity, mutual relationships of these agents and relationships with the environment, make it difficult to verify the quality and correctness of MAS. Therefore, there is a need for new testing methods dealing with their specific features. The methods need to be effective and adequate to evaluate agent's characteristics such as autonomous behaviors, pro-activity, reactivity etc. There is an emerging need for detailed guidelines for the processes to follow during the testing of multi-agent systems. This is a very essential step towards the adoption of Agent-Oriented Software Engineering (AOSE) methodology by industry.

6. Future research direction for testing of MAS

- There can be problems of bottlenecks in the multi-agent system despite its distributed

character, as there are often high-load hubs and central points that can slow down the entire system. This area can be one of the researches.

- Study the scope of stress testing on a multi-agent system, in order to determine behavior and stability in the case of mass collapse of a great number of agents. This is because, even in such a situation, the multi-agent system should remain stable, without substantial impact on its performance and its global behavior
- The other research study can focus on the proposing a new testing methodology to support agent-oriented software engineering (AOSE) that will be having advantage over the existing ones.

7. Conclusion and Future work.

Testing of multi-agent systems poses more complex problems than testing of ''traditional'' computer systems. Emergent properties and behaviors of multi-agent systems are their inseparable traits that add to the system such characteristics that no single one of their parts possess.

Although there are a number of tools and approaches designed for testing various kinds of problems in multi-agent systems, we are still lacking a consistent method for multi-agent testing.

In this paper we mainly reported state of art Multiagent based testing techniques and its challenges and listed, some future research direction for testing Multi Agents System (MAS) are also highlighted in this paper.

Our future work will deal with testing of multi-agents systems with a special focus on autonomous system. We will also focus on testing different bottle-neck scenarios which might have an impact on the multi-agent systems.

REFERENCES

[1] Macal CM, North MJ (2006) Tutorial on agent-based modeling and simulation part 2: how to model with agents. In: Perrone LF, Wieland FP, Liu J, Lawson BG, Nicol DM, Fujimoto RM (eds) Proceedings of the 38th conference on winter simulation, Winter Simulation Conference, Monterey, California, pp 73–83.

[2]Bonabeau E (2002) Agent-based modeling: methods and techniques for simulating human systems. Proc Natl Acad Sci U S A 99(90003):7280–7287

[3] Bradshaw. 1997. Software Agents. MIT Press, Cambridge, MA, USA.

[4]Russell S, Norvig P (2009) Artificial intelligence: a modern approach, 3rd edn. Prentice Hall, Englewood Cliffs

[5] Wooldridge MJ, Jennings NR (1995) Intelligent agents: theory and practice. Knowl Eng Rev 10(2):115–152

[6] Castle CJ, Crooks AT (2006) Principles and concepts of agent-based modelling for developing geospatial simulations, centre for advanced spatial analysis (UCL). UCL (University College London), London

[7] Franklin S, Graesser A (1996) Is it an agent, or just a program?: a taxonomy for autonomous agents. In: Müller JP, Wooldridge MJ, Jennings NR (eds) Proceedings of the third international workshop on agent theories, architectures, and languages, Springer, pp 21–35

[8] Macal CM, North MJ (2005) Tutorial on agent-based modeling and simulation. In: Kuhl ME,Steiger NM, Armstrong FB, Joines JA (eds) Proceedings of the 37th conference on winter simulation, Winter Simulation Conference, Orlando, Florida, pp 2–15.

[9] Epstein JM (2007) Agent-based computational models and generative social science, in generative social science studies in agent-based computational modeling. Princeton University Press, Princeton, pp 41–60.

[10] Benenson I, Torrens PM (2004) Geosimulation: automata-based modeling of urban phenomena.Wiley, New York.

[11] Smith MJD, Goodchild MF, Longley PA (2007) Geospatial analysis: a comprehensive guide to principles, techniques and software tools, 2nd edn. Troubador Publishing Ltd, Kibworth.

[12]Holland J (1996) Hidden order: how adaptation builds complexity, 1st edn. Addison Wesley Longman, Redwood City.

[13] Showalter P, Lu Y (2009) Geospatial techniques in Urban hazard and disaster analysis. Springer,The Netherlands.

[14] Wooldridge. 1998. Agent-based computing. Interoperable Communication Networks.vol. 1, no. 1. pp. 71-97.

[15] Andreas S. J. 2010. Multi-Agent Systems: An Investigation of the Advantages of Making Organizations Explicit. MSc Thesis, Department of Informatics and Mathematical Modeling, Technical University of Denmark.

[16] Gustasfon, D. A.; & Matson, E. (2003). Taxonomy of Cooperative Robotic Systems, Proceeding of 2003 IEEE International Conference on Systems, Man and Cybernetics, pp.1141-1146, ISBN:0-7803-7952-7, Crystal City Hyatt Regency, October 2003, IEEE, Washington DC.

[17] Changhong, L.; Minqiang, L.; & Jisong, K.; (2002). Cooperation Structure of Multi-agent and Algorithms, Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems, pp. 303–307, ISBN:0-7695-1733-1, September 2002, IEEE, Divnomorskoe , Russia.

[18] Cernuzzi, L., Cossentino, M., Zambonelli, F. Process Models for Agent-based Development, Journal of Engineering Applications of Artificial Intelligence, 18(2), 2005.

[19] DeLoach, S., Wood, M. and Sparkman, C. Multiagent Systems Engineering. International Journal of Software Engineering and Knowledge Engineering, vol. 11, No. 3, pp. 231-258, 2001.

[20] Iglesias, C., Garijo, M., Gonzalez, J.C., Velasco, J.R. Analysis and Design of Multiagent Systems using MAS-CommonKADS.Springer, LNCS 1365, pp. 312-328, 1997.

[21] Jonker, C.M., and Treur, J. Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. Proc. of COMPOS'97, Springer, LNCS 1536,1998.

[22] Caire, G., Cossentino, M., Negri, A., Poggi, A., and Turci, P., Multi-agent systems implementation and testing. In Proc. of From Agent Theory to Agent Implementation - Fourth International Symposium (AT2AI-4), 2004.

[23] Ronsse, M., Bosschere, K. D., Christiaens, M., Kergommeaux, J. C., and Kranzlmüller,D., Record/replay for nondeterministic program executions. Communications of the ACM, 46(9), September 2003.

[24] Huget, M.-P.; Demazeau, Y.; Evaluating multiagent systems: a record/replay approach.Intelligent Agent Technology, 2004. (IAT 2004). Proceedings.
IEEE/WIC/ACM International Conference on 2004 .

[25] Rodrigues, L.F; Carvalho G. R.; Paes, R. B.; Lucena, C.J.P.; Towards an Integration Test Architecture for Open MAS (SEAS), SBES, 2005.

[26]M Moreno, J Pavón, and A Rosete1;Testing in Agent Oriented Methodologies, Lecture Notes in Computer Science, 2009, Volume 5518.

[27]Thanh-Binh Trinh, Quang-Thap Pham, Ninh-Thuan Truong, and Viet-Ha Nguyen; A Runtime Approach to Verify Scenario in Multi-Agent Systems, 2010 Second International Conference on Knowledge and Systems Engineering.

[28]Glenn Callow, Graham Watson, Roy Kalawsky; System Modelling for Run-time Verification and Validation of Autonomous Systems, 2010 5th International Conference on System of Systems Engineering.

[29] Saeed Zamani, Ramin Nassiri, Sam Jabbehdari; A New Test Process in Agent-oriented Software Engineering, International Journal of Advancements in Computing Technology Volume 3, Number 7, August 2011.

[30] Najwa Abu Bakar, Ali Selamat; Analyzing Model Checking Approach for Multi Agent System Verification, 2011, 5th Malaysian Conference in Software Engineering (MySEC).

[31]Cu D. Nguyen, Anna Perini, Carole Bernon, Juan Pavón and John Thangarajah;
Testing in Multi-Agent Systems, Agent-Oriented Software Engineering X , Lecture Notes in Computer Science, 2011, Volume 6038/2011.

[32]Tomas Salamon; A Three-Layer Approach to Testing of Multi-agent Systems, Information Systems Development ,2010, 393-401, DOI: 10.1007/b137171_41.

Authors Bibliography

Mohammad Mottahir Alam has around six years of experience working as Software

Engineer (Quality) for some leading software multinationals where he worked on projects for companies like Pearson and Reader's Digest. He has received his Bachelors degree in Electronics & Communication and Masters in Nanotechnology from Faculty of Engineering and Technology, Jamia Millia Islamia University, New Delhi.

He is presently working as a Lecturer in the Faculty of Electrical and Computer Engineering, King Abdul Aziz University, Jeddah, Saudi Arabia. His research interest includes Software Engineering, Component Based Software Engineering and Agent Based Software Engineering.

Mr. Asif Irshad Khan received his Bachelor and Master degree in Computer Science from the Aligarh Muslim University (A.M.U), Aligarh, India in 1998 and 2001 respectively. He is presently working as a Lecturer Computer Science at the Faculty of Computing and Information Technology, King Abdul Aziz University, Jeddah, Saudi Arabia.

He has more than seven years experience of teaching as lecturer to graduate and undergraduate students in different universities and worked for four years in industry before joining academia full time.

He has published more than 10 research papers in national and International journals, and his research interest includes Software Engineering, Component Based Software Engineering and Agent Oriented Software Engineering.

Mr. Noor-ul-Qayyum received his Master degree in Information Technology from National University of Sciences and Technology, Islamabad, Pakistan in 2009. He had been working for Ikonami Technologies and Decker Intellectual Properties as a software engineer before joining King Abdul Aziz University.

He is currently working as a lecturer in King Abdul Aziz University. He has industry experience in SCORM based e-learning courseware development using ADDIE model. His research interest includes e-learning, software watermarking, and mobile agent security issues.

Mr. Abdullah Maresh Ali received his Master degree in Computer Science from King Abdul Aziz University, Jeddah, Saudi Arabia in 2011. He is presently working as a Lecturer Computer Science at the Faculty of Computing and Information Technology, King Abdul Aziz University, Jeddah, Saudi Arabia.

His research interest includes Computer networks and Agent Oriented Software Engineering.