Volume 3, Issue 1, January 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Analysing the Scope for Testing in PASSI Methodology

N.Sivakumar
Department of CSE
Pondicherry Engineering College

K.Vivekanandan
Department of CSE
Pondicherry Engineering College

D.Rama
Department of CSE
Pondicherry Engineering College

***Abstract*— Though software agents employ some of the mechanisms and characteristics of objects, both are reasonably different. Current research and development of Agent Oriented Software Engineering (AOSE) focuses mainly on architectures, protocols and framework of agent based system. Several AOSE methodologies such as Passi, MaSE, Tropos, Gaia, Prometheus, ADELFE, Message were proposed and those methodologies are meant for providing the guidelines and the abstraction to be exploited in the various phases of agent-based software development such as analysis, design, implementation. PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies. By evaluating the lifecycle coverage of PASSI, it is understood that the methodology deals with analysis, design and implementation models for building agent based system whereas testing is not considered much. The main objective of this paper is to analyze the scope for testing in PASSI methodology and to provide few insights of testing in the methodology.**

***Keywords*— PASSI, Software Agent, AOSE Methodology, MAS**

I. INTRODUCTION

Agents are the software program which works on behalf of human to carry some task which has been delegated to it and take their own decision according to the requirement. We think it is more natural to describe agents using a psychological and social language. Therefore we believe that there is a need for specific methods or representations tailored for agent-based software. This belief originates from the related literature. To give an example, Yiu and Li [2] say: "an agent is an actor with concrete, physical manifestations, such as a human individual. An agent has dependencies that apply regardless of what role He/she/it happens to be playing". On the other hand, Jennings [1] defines an agent as "an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives". Also, Wooldridge and Ciancarini see the agent as a system that enjoys autonomy, reactivity, pro-activeness, and social ability [3]. It is also a computer program that works toward goals 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 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 [6]. The agents interact in a concurrent, asynchronous and decentralized manner [7]; hence MAS turn out to be complex systems [8]. They are also non-deterministic, since it is not possible to determine a priori all interactions of an agent during its execution. Consequently, they are difficult to debug and test. Therefore, multi-agent systems (MAS) differ from non-agent based ones because agents are meant to be autonomous elements of intelligent functionality. Consequently, this requires that the agent-based software engineering methods need to encompass Standard design activities and representations as well as models of the agent society.

Agent Oriented Software Engineering (AOSE) is an umbrella term in which several researches have been proposed on new varieties of metaphors, formal modelling approaches and techniques, and development methodologies and tools, specifically suited towards agent-oriented paradigm [11].The Agent oriented methodologies aid us in analysing and designing the agent in a system. Each methodology differ from other in various aspects like design, implementation etc. An important step is to understand the differences between all methodologies, their strengths, weaknesses and domains of applicability. Several agent oriented development methodologies were proposed to build open, heterogeneous and complex internet based systems. Some of the most popular AOSE methodologies are MaSE, Tropos, Gaia, Prometheus, Passi, ADELFE, Message etc. Our survey states that the agent based software are currently been tested by using Object-Oriented (OO) testing techniques, upon mapping of Agent-Oriented (AO) abstractions into OO constructs[12]. However agent properties such as Autonomy, Proactivity, and Reactivity etc., cannot be mapped into OO constructs. There arises the need for proper testing techniques for agent based software. The main objective of our paper is to propose a testing mechanism based on agent's important mental state, the role. 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[10]. The main aim of a test is to find faults.

Software testing is inseparable in a software development process and it is an activity that determines the correctness and the quality of the software. One of the most fundamental obstacles to large-scale take-up of agent oriented software engineering methodologies in industry is the lack of proper testing techniques for testing the Multi-agent system. MAS testing are complex and challenging too. Existing software testing techniques (both conventional and object-oriented) cannot be applied over software agents as these agents are designed to be autonomous, proactive, collaborative and ultimately intelligent. One more important fact that traditional testing techniques cannot be applied towards software agents is that the agents communicate through message passing rather than method invocation [Automated continuous testing]. Moreover, agents can be programmed to learn and thus the successive tests with the same test data may give different result [test agent for testing agent communities].

Passi (process for agent societies specification and implementation) is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies, integrating design models and concepts from both object-oriented (OO) software engineering and artificial intelligence approaches using the UML Notation. Because of the specific needs of agent design, the UML semantics and notation will be used as reference points, but they will be extended, and UML diagrams will be often used to represent concepts that are not considered in UML and/or the notation will be modified to better represent what should be modelled in the specific artifact. The PASSI process is composed of five process components: System Requirements, Agent Society, Agent Implementation, Code model, and Deployment model. Testing is very important for any system to decide whether the goal and quality of the system is been achieved or not. Passi doesn't provide any testing support and the main objective of this paper is to propose a testing mechanism that might be suitable for agent based system developed using passi.

II. PASSI METHODOLOGY BACKGROUND

In PASSI, UML representation has been adopted since it is widely accepted both in the academic and industrial environments. The PASSI methodology represented in figure.1 is made up of five models namely

1. System Requirement model
2. Agent society model
3. Agent Implementation model
4. Code model
5. Deployment model

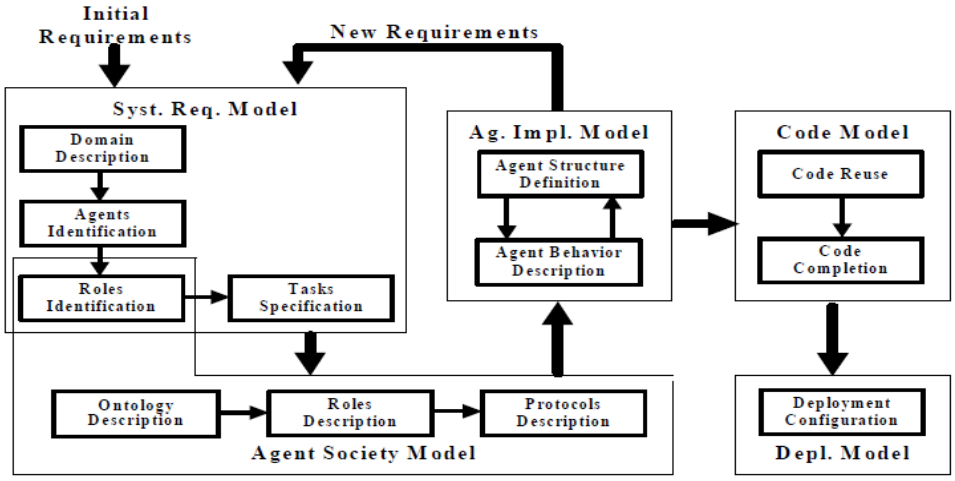

Fig 1.PASSI Methodology

A. *System Requirements Model*

It is an anthropomorphic model of the system requirements in terms of agency and target. It comprises four steps namely Domain Description, Agent Identification, Roles Identification and Task Specification

1) Domain Description phase

The functional description of the system composed of a hierarchical series of use case diagrams are represented in domain description phase. Scenarios of the detailed use case diagrams are then explained using sequence diagrams.

2) Agent Identification phase

More than one agent works in a co-ordinated way forms a Multi-Agent System (MAS). From the use case diagram depicted in domain description phase, every use case is seen as functional decomposition and are mapped to agent's responsibilities. In this phase, the agents are identified for the system in such a way that the functionalities of the system are mapped to individual agents so that the system goal is achieved.

3) Role Identification Phase

Roles identification phase is a series of sequence diagrams exploring the responsibilities of each agent through role-specific scenarios. Roles Identification is based on exploring all the possible paths of the Agents Identification diagram involving inter-agent communication

4) Task Specification Phase

Tasks specification phase is a specification of the capabilities of each agent with activity diagrams. Tasks generally encapsulate some functionality that forms a logical unit of work. At this step, for each agent we focus on its behaviour in order to decompose it into tasks.

B. *Agent Society Model*

Agent society model describes the social interactions and dependencies between the agents. Developing this model involves three steps namely Ontology description, Role description and Protocol description.

1) *Ontology Description Phase*

Ontology description yields two diagrams namely Domain Ontology Description (DOD) and Communication Ontology Description (COD). In DOD, the domain ontology is represented as an XML schema and are used to define the pieces of domain's knowledge. In COD the communications are represented by a class diagram and it depicts the agents with both their knowledge and the ontology of their communication.

2) *Role Description Phase*

This phase describes the lifecycle of an agent taking into account its roles, the collaborations and conversation it is involved in. This phase yields a collection of class diagram in which classes are used to represent roles. Each role is obtained composing several tasks in a resulting behaviour.

3) *Protocol Description*

Protocol plays a major role in agent communication. As specified by the FIPA architecture, a protocol has been used for each communication among agents.

C. *Agent Implementation Model*

A classical model of the solution architecture in terms of classes and methods; the most important differences with common object-oriented approach is that we have two different levels of abstraction, the social (multi-agent) level and the single level. This model is composed by: agent structure definition and agent behavior description.

1) Agent Structure Definition Phase

This phase produces several class diagrams logically subdivided into two views: the multi-agent and single-agent. Multi-agent view brings out the general architecture of the system so that the agents and their tasks are found. In the single-agent view, the agent's internal structure that reveals the attributes and methods of agent class and its inner tasks are found.

2) Agent Behaviour Description Phase

The Agent Behaviour Description phase produces several diagrams that are subdivided into the multi-agent and the single-agent views. In Multi-Agent Behaviour Description (MABD) view, one or more activity diagrams are drawn to show flow of events between and within both the main agents. The Single-Agent Behaviour Description (SABD) view is quite a common one as it involves methods implementation, exactly the ones introduced in the SASD diagrams. classes and their inner classes (representing their tasks).

D. *Code Model*

Code Model is a model of the solution at the code level. It consists of two phases: Code Reuse phase and Code production phase.

1) Code Reuse Phase

Predefined patterns of agents and tasks are reused in this phase. Pattern includes design diagram too. Rational Rose UML CASE tool has been extended to support design reuse

2) Code Production Phase

This phase is the classical work of the programmer, who just needs to complete the body of the methods yielded to this point, by taking into account the design diagrams.

E. *Deployment Model*

This model is meant for the distribution of the system's parts across hardware Processing units, and of their migration.

1) Deployment Configuration Phase

The Deployment Configuration phase describes the allocation of agents to the available processing units and any constraints on migration and mobility. It has been thought to comply with the requirements of detailing the agents' positions in distributed systems or more generally in mobile-agents' contexts. The Deployment Configuration diagram is a UML deployment diagram and illustrates the location of the agents, their movements, and their communication support.

III. LIFE CYCLE COVERAGE OF PASSI

Lifecycle coverage specifies what elements of software development are dealt with within the methodology. Every methodology may have elements that are useful in several stages of the development lifecycle such as requirement, analysis, design, implementation, and testing. It leads the developers to an understanding of an agent–oriented system as an organization of roles. These seek to achieve goals by means of roles and have dependencies to other roles. In fact a Task in PASSI is related to a particular behavior and the agent performs his action to achieve its goals. Lifecycle coverage is an important evaluation aspect for a methodology because a detailed description of the each phase is included in the development lifecycle. It enhances the appropriate use of a methodology and increases its acceptability as a well-formed engineering approach. PASSI methodology is incomprehensive with respect to lifecycle coverage. PASSI covers analysis and design stage within the development lifecycle. When it comes to implementation, the analysis and design results may be translated to UML notations and then use an object-oriented language for implementation. In PASSI, UML has been adopted because it is widely accepted both in the academic and industrial environments. Its extension mechanisms (constraints, tagged values and stereotypes) help in customizing the representations of agent-oriented designs so as to avoid the adoption of a totally new modeling language. Our use of UML is therefore intended to establish a common language such that it can be easily understood and in so doing we kept in our minds experiences coming from the AUML standardization effort. In our approach, arguments that are typical of the object-oriented approach (such as classes, attributes, etc) are only used in the implementation phases as this could not be made in a different way. Some methodologies use Roles as a concept to structure a multi agent system and to identify single agent classes. At last the testing stage is defined but not described in detailed manner. It has Agent and Society Testing. In order to evaluate it Traditional testing techniques cannot be applied towards software agents as the agents communicate through message passing rather than method invocation. Existing software testing techniques (both conventional and object-oriented) cannot be applied over software agents as these agents are designed to be autonomous, proactive, collaborative and ultimately intelligent.

IV. PROPOSED WORK

In Passi, a role acts as an important abstraction on MAS analysis and design. Role identification phase in analysis and Role description phase in design makes us understand the importance of role concept in Passi. Thus to ensure the proper working of MAS developed using Passi methodology, it is proposed to have a role oriented testing mechanism. Role is defined as a capability enabler that exposes to the agent that plays it a set of actions.

A) Role Oriented Testing

Role can be defined as an abstract description of an entity's expected function and encapsulates the system goals that it has been assigned the responsibility of fulfilling. Role can be identified in the system by following a Goal-oriented approach. Roles provide a well-defined interface between agents and cooperative processes [14]. This allows an agent to read and follow, normative rules established by the cooperation process even if not previously known by the agent. Their major motivation to introduce such roles is to increase the agent system's adaptability to structural changes. Several AOSE methodologies were analyzed and compared and found that the strong weakness observed from almost all the methodologies were, there is no proper testing mechanism for testing the agent-oriented software. Our survey states that the agent based software are currently been tested by using Object-Oriented (OO) testing techniques, upon mapping of Agent-Oriented (AO) abstractions into OO constructs[3]. However agent properties such as Autonomy, Proactivity, and Reactivity etc., cannot be mapped into OO constructs. There arises the need for proper testing techniques for agent based software. The main objective of our paper is to propose a testing mechanism based on agent's important mental state, the role. Every individual agent has its own goal to be achieved and plans to do to fulfill the goal. In addition to goal and plan, role is one important mental state of the agent, which is defined as a set of capabilities and expected behavior. A role [9][10] can be represented as <Goal, Responsibilities, Protocol, Permissions>.

- Goal, for which the agent playing this role is responsible.
- Responsibilities, Which indicates the functionalities of agents playing such roles
- Protocol, which indicates how an agent playing such role can interact with agents playing other role
- Permissions, which are a set of rights associated with the role.

Process of Role Oriented Unit Testing

STEP 1: Select the Agent to be tested
STEP 2: Identify Goals (G_i), Roles (R_j) and their corresponding Responsibilities (Re_k)
STEP 3: Design Role Model Diagram (A_x (G_i R_j Re_k)
STEP 4: Analyze role model (A_x (G_i R_j Re_k)
STEP 5: Define the interacting agents and situations.
STEP 6: Make the interacting agents as Pseudo agents.
STEP 7: Identify environmental factors pre-conditioning input trigger Re_k
STEP 8: Identify fulfillment criteria that satisfies Responsibility
STEP 9: Create test suite for Re_k corresponding to R_j
STEP 10: Run test cases

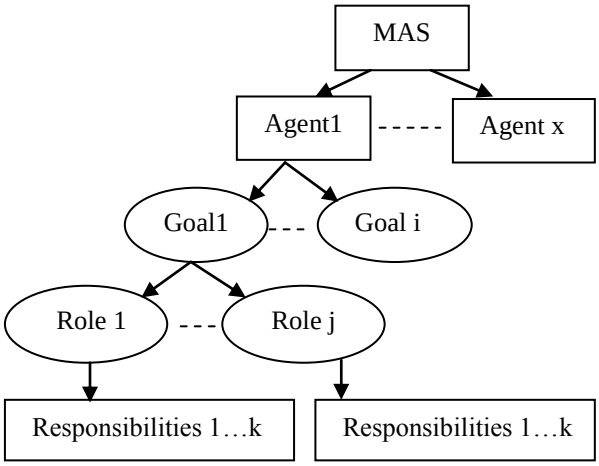

Fig 2.Role Model

Let us consider an agent based shopping system in which buyer is one of the agents involved in the system. The goal of the buyer agent is to buy the quality and low cost item in the shop. To achieve this goal, the buyer agent has to take many roles such as customer role and negotiator role etc. Every role has its own responsibilities; say for example, customer role has the following responsibilities such as Searching for products, selecting the best product, paying for the product etc. Thus to test a single agent, scenarios (test cases) are to be developed to test whether all the responsibilities of the corresponding agent's role got satisfied. When all the roles are been tested and working fine, then by default we claim that the goal of the agent is been tested, thereby the individual agent is tested successfully

```
<Agent>
  <Goal>
    <Role1>
      <Responsibilities 1>
        <Protocol>
        .
      <Responsibilities n>
        <Protocols>
    </Role1>
    <Role n>
      <Responsibilities 1>
        <Protocol>
        .
      <Responsibilities n>
        <Protocol>
    </Role n>
  </Goal>
</Agent>
```

```
<Buyer>
  <To buy an item>
    <Customer>
      <Searching for products>
      <Selecting the best product>
      <Paying for the product>
    </Customer>

    <Negotiator>
      <Involvement in negotiating>
      <Searching for brands >
      <Selecting best price>
    </Negotiator>
  </To buy an item>
</Buyer>
```

Fig 3.XML Schema Representation for Buyer Agent

Let us consider the buyer agent. One of the goals of the buyer agent is to buy the best item. To achieve this goal the agent has to take customer role and the negotiator role etc. Every role has its own responsibilities; say selecting the best product is one of the responsibilities of the customer role. The role of an agent comprises the logic of the test. As every role of an agent has number of responsibilities to get satisfied, the derivation of test case focuses on the responsibilities and thereby validates whether the role hold by the agent servers the purpose. Table 1 brings out the test case structure and some sample test case adopted for role oriented testing mechanism towards agent based shopping system

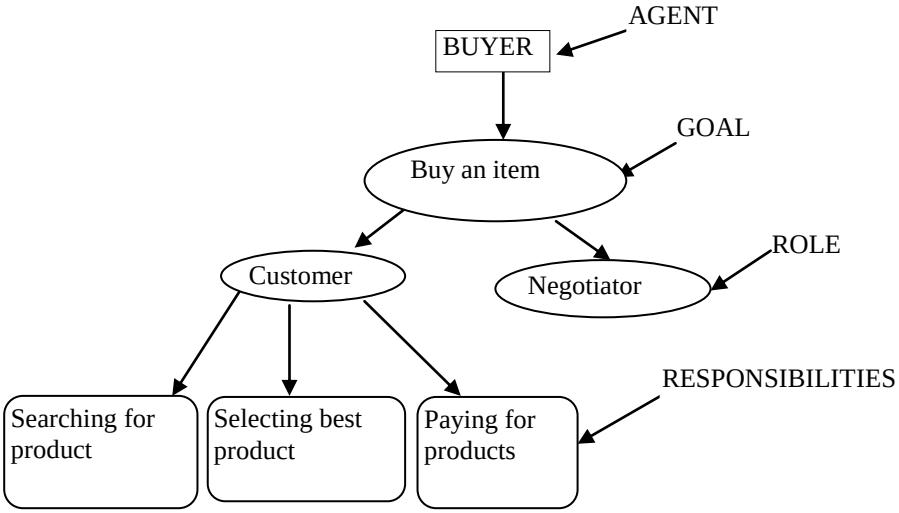

Fig 4.Role Model representation for Buyer Agent

TABLE I
TEST CASE STRUCTURE

T.ID	TESTED AGENT	GOAL	ROLE	SITUATION	INPUT	EXPECTED RESULT	OBSERVED RESULT	RESULT
SE1	Buyer	To buy an item	customer	Buyer needs to buy products	Cash/credit card	Accept buyer and distribute	Buyer agent purchased the product	Passed

V. CONCLUSION

The main objective of our paper is to propose a testing mechanism based on agent's important mental state, the role. Almost no methodology deals with testing issues, stating that the testing can be carried out using the existing object-oriented testing techniques. Though objects and agents have some similarities, they both differ widely. Our survey states that the agent based software are currently been tested by using Object-Oriented (OO) testing techniques, upon mapping of Agent-Oriented (AO) abstractions into OO construct [12]. However agent properties such as Autonomy, Proactivity, and Reactivity etc., cannot be mapped into OO constructs. There arises the need for proper testing techniques for agent based software. Most of existing work on MAS uses role modeling for system analysis; however, role models are only used at conceptual level with no realizations in the implemented system. In this paper, we propose a PASSI methodology for role-based modeling of multiagent software systems. Roles have a limited use within object-oriented (OO) analysis and design. In extending OO concepts to agent oriented software engineering (AOSE), roles must be adapted to incorporate autonomy and other agent essentials. Every agent has its own role to perform so as to achieve its goal. The Responsibilities will vary for each Role based on the environment. This leads to test the functionalities of each role by deriving appropriate test cases by using Role Oriented method.

REFERENCES

[1] Jennings, N.R.On agent-based software engineering. In Artificial Intelligence, 117 (2000), 277-296
[2] Yu, E., Liu, L. Modeling Trust in the I* Strategic Actors Framework. Proc. of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies at Agents2000 (Barcelona, Catalonia, Spain, June 2000).
[3] M. Wooldridge, P. Ciancarini. Agent-Oriented Software Engineering: The State of the Art. Agent-Oriented Software Engineering, P. Ciancarini and M. Wooldridge eds., Springer-Verlag, Berlin (2001), 1-28.
[4] Searle, J.R., Speech Acts. Cambridge University Press, 1969.
[5] J.Odell, H. Van Dyke Parunak, B. Bauer. Representing Agent Interaction Protocols in UML, Agent-Oriented Software Engineering, P. Ciancarini and M. Wooldridge eds., Springer-Verlag, Berlin (2001), 121–140.
[6] http://agtivity.com/agdef.htm, accessed in Oct/2005.
[7] 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 Page(s):536 - 539 Digital Objects Identifier 10.1109/IAT.2004.1343013.

[8] Jennings, Nicholas R., an Agent-Based Approach for Building Complex Software Systems, Communications of the ACM, 44(4), 35-41, April 2001.

[9] 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), Vienna, Austria, April 2004.

[10] Rouff, C.; A Test Agent for Testing Agents and Their Communities. Aerospace Conference Proceedings, 2002. IEEE Volume 5, 2002 Page(s):5 - 2638 vol.5 Digital Object Identifier 10.1109/AERO.2002.1035446.

[11] Mailyn Moreno, Juan Pavon, Alejandro Rosete. Testing in Agent Oriented Methodologies. IWANN 2009, Part II, LNCS 5518, pp. 138–145, 2009. © Springer-Verlag Berlin Heidelberg 2009.

[12] Praveen Ranjan Srivastava, Karthik Anand V, Mayuri Rastogi, Vikrant Yadav, G.Raghurama. Extension of Object-oriented Software testing techniques to Agent Oriented software testing, in Journal of Object Technology, vol. 7, no. 8, November-December 2008, pp. 155-63.

[13] Grady Booch, James Rumbaugh, and Ivar Jacobson, the Unified Modeling Language User Guide, Addison-Wesley, 1999.

[14] Giacomo Cabri, Letizia Leonardi, Luca Ferrari and Franco Zambonelli. Role-based Software agent interaction models: a survey. The Knowledge Engineering Review, Vol. 25:4, 397 419, 2010.

[15] M. Cossentino, P. Burrafato, S. Lombardo, L. Sabatucci - "Introducing Pattern Reuse in the Design of Multi-Agent Systems" - AITA'02 workshop at NODe02 - 8-9 October 2002 - Erfurt, Germany (www.csai.unipa.it/cossentino/paper/AITA02.pdf).

