IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: 2249-9555
Vol. 2, No.6, December 2012

Evaluating the Coverage of Development Lifecycle in Agent Oriented Software Engineering Methodologies

N.Sivakumar
Department of Computer Science and Engineering
Pondicherry Engineering College
Puducherry, INDIA.

K.Vivekanandan
Department of Computer Science and Engineering
Pondicherry Engineering College
Puducherry, INDIA.

***Abstract*— Agent Oriented Software Engineering (AOSE) methodology borrows concepts from artificial intelligence and can also be considered as the next evolution of Object-Oriented Software Engineering (OOSE). A methodology aims at describing all the elements necessary for the development of a software system. It is the responsibility of a methodology to provide formal guidelines on how to proceed through the phases involved in the software development process such as analysis, design, implementation, testing and maintenance. A methodology will be successful and acceptable only when it covers all the important phases of Software Development Life Cycle (SDLC). There are plenty of AOSE methodologies that provides conceptual frameworks, notations and techniques for building an agent based system. The main objective of this paper is to evaluate the development lifecycle coverage criteria of eight different AOSE methodologies such as MASCommonKADS, MaSE, GAIA, MESSAGE, Tropos, Prometheus, PASSI and SODA.**

Keywords- Agent Oriented Software Engineering, Software Development Lifecycle, Agent based system, Software Testing

I. INTRODUCTION

A software development methodology refers to the framework that is used to structure, plan, and control the process of developing a software system. A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. Now an increasing number of problems in industrial, commercial, medical, networking and educational application domains are being solved by agent-based solutions [1]. The key abstraction in these solutions is the agent. An "agent" is an autonomous, flexible and social system that interacts with its environment in order to satisfy its design agenda. In some cases, two or more agents should interact with each other in a multi agent system (MAS) to solve a problem that they cannot handle alone. According to the above mentioned facts, developing agent-based systems, developers will face new abstractions and concepts. Also they should handle main challenges that exist in the development of complex, open and distributed systems. For example, autonomy of agents, emergent behaviour and dynamic configuration are among the prominent features of multi-agent systems but in practice and in development time these features make system development more complex and practitioners need new methods and tools to handle these types of complexity.

Agent-oriented software engineering (AOSE) is a new discipline that encompasses necessary methods, techniques and tools for developing agent-based systems. It is a powerful way of approaching complex and large scale software engineering problems and developing agent-base systems. Several AOSE methodologies were proposed for developing software, equipped with distinct concepts and modelling tools, in which the key abstraction used in its concepts is that of an agent. Some of the more popular AOSE methodologies [2] were listed below.

1. MAS CommonKADS (1996-1998)
2. MaSE(1999)
3. GAIA(2000)
4. MESSAGE(2001)
5. TROPOS(2002)
6. PROMETHEUS(2002)
7. PASSI(2002)
8. SODA(2002)

Though agent-based systems are developed for solving complex distributed problem and also can be claimed to be the extension of object-oriented approach, there is a obstacle for it growth as there are no standard implementation and testing approach. When we analysed and compared the strengths and weaknesses of the above mentioned AOSE methodologies, the strong weakness that we 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 softwares are currently being 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 evaluate eight most popular AOSE methodologies in terms of lifecycle coverage criteria, so that how much the methodologies were complete can be analysed. In the next section, all the eight methodologies are briefed in its original form and in the later sections; the methodologies are compared with respect to coverage of software development lifecycle criteria.

II. AOSE METHODOLOGY

Designing software in agent oriented approach deals with new concepts such as: agent, goal, task, services, organization, interactions, environment, etc. New tools and models are therefore needed to help engineers to work with these new notions. Methodologies help designers to deal with these problems and thereby help to manage this complexity. A methodology aims to prescribe all the elements necessary for the development of a software system. A number of methodologies have been proposed for implementing agent oriented software [2]. Some of the most popular agent-oriented software engineering methodologies were discussed below.

A. MAS-CommonKADS

MAS-CommonKADS is an agent-oriented software engineering methodology that guides the process of analysing and designing multi-agent systems [4].

1) Analysis

This phase results with the requirement specification of MAS. The analysis phase defines the following models

Agent model that specifies the agent characteristics: reasoning capabilities, skills (sensors/effectors), services, agent groups, and hierarchies (both modelled in the organisation model).

a) Task model that describes the tasks that the agents can carry out: goals, decompositions, ingredients, problem-solving methods, and so forth.

b) Expertise model that describes the knowledge needed by the agents to achieve their goals.

c) Organisation model that describes the organisation into which the MAS is going to be introduced and the social organisation of the agent society.

d) Coordination model that describes the conversations between agents, their interactions, protocols, and required capabilities.

e) Communication model that details the human-software agent interactions and the human factors for developing these user interfaces. This model uses standard techniques for developing user interfaces.

2) Design

During the design phase, the design model is developed. This phase is extended for multi-agent systems and consists of the following phases:

a) Agent network design: The infrastructure of the multi-agent system (socalled network model) is determined and consists of network, knowledge, and coordination facilities. The agents (so-called network agents) that maintain this infrastructure are also defined, depending on the required facilities. Some of these required facilities can be:

- Network facilities: agent name service, yellow/white pages service, de/registering and subscription service, security level, encryption and authentication, transport/application protocol, accounting service, and the like.
- Knowledge facilities: ontology servers, PSM servers, knowledge representation language translators, and so forth.
- Coordination facilities: available coordination protocols and primitives, protocol servers, group management facilities, facilities for assistance in coordination of shared goals, police agents for detecting misbehaviors and the control of the usage of common resources, and so forth. The result of the common facilities shared by the agents allows the efficient communication between the agents and is expressed by an ontology, in the same way as a service ontology.

b) Agent design : the most suitable architecture is determined for each agent, and some agents can be introduced or subdivided according to pragmatic criteria. Each agent is subdivided into modules for user-communication (from communication model), agent communication (from coordination model), deliberation and reaction (from expertise, agent, and organisation models), and external skills and services (from agent, expertise, and task models). The agent design maps the functions defined in these modules onto the selected agent architecture. The issue of designing an agent architecture is not addressed in the methodology, since the agent architecture is provided by the agent development platform.

c) Platform design: selection of the software (multi-agent development environment) and hardware that is needed (or available) for the system.

B. MASE (Multi-Agent System Engineering)

MASE [5] is an iterative process. It deals the capturing the goals and refining the roles of an agent. It appears to have significant tool support. MASE treats the agents as simple software processes that interact with each other to meet the overall system goals.

The development process consists of two main phases namely

1) Analysis

Capturing goals: High-level goals are identified from requirements analysis in the beginning step. These goals are then decomposed into subgoals and collected into a tree-like structure.

a) Applying use cases: The second step generates use-cases and their corresponding sequence diagram.

b) Refining roles: The last step of the analysis phase involves role refinement. The main task during this step is to map goals into roles where every goal in the system needs a delegated role.

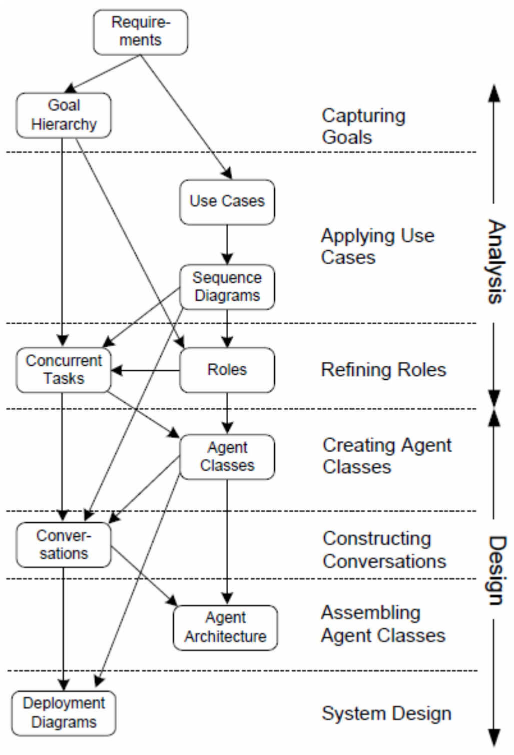

Figure.1 MaSE Methodology

2) *Design*

a) Creating agent classes: The first step is a process, which creates Agent classes and their interactive behavior.

b) Constructing conversations: After each Agent class is recognized, constructing a conversation is the next step. In this step, designers construct conversation models used by Agent classes.

c) Assembling agent classes: The assembling Agent class step creates Agent class internals.

d) System design: The final step of design is system design, where the Agent classes are instantiated into actual Agents.

C. GAIA (Generic Architecture for Information Availability)

GAIA [6] is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly. Gaia deals with both the macro (societal) level and the micro (agent) level aspects of design. Gaia encourages a developer to think of building agent-based systems as a process of organisational design. Analysis and design can be thought of as a process of developing increasingly detailed *models* of the system to be constructed. The main models used in Gaia are summarised in Figure 2

1) *Analysis*

The objective of the analysis stage is to develop an understanding of the system and its structure. In Gaia, the understanding is captured in the system's *organization* which is viewed as a collection of roles, which stand in certain relationships to one another, and that take part in systematic, institutionalized patterns of interactions with other roles. In Gaia, an agent-based system is viewed as an artificial society or organization. The main reason for viewing an agent-based system as organization is to incorporate the concept of roles. The organization model in Gaia is comprised of two further models namely the *roles model* and the *interaction model.*

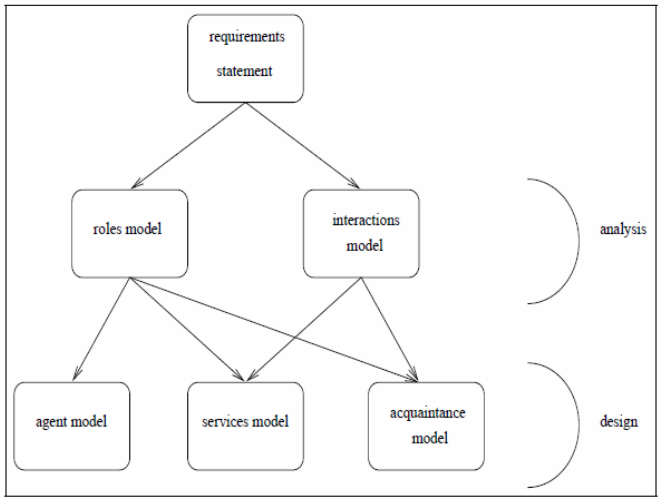

Figure.2 GAIA Methodology

a) Role model

The roles model identifies the key roles in the system. Here a role can be viewed as an abstract description of an entity's expected function. A role is defined by four attributes: *responsibilities*, *permissions*, *activities*, and *protocols*. A role has a certain functionality that is represented by an attribute known as role's *responsibilities. Responsibilities* determine functionality and perhaps the key attribute associated with a role. Responsibilities are divided into two types: *liveness properties* and *safety properties*. Liveness properties intuitively state that "something good happens". They describe those states of affairs that an agent must bring about, given certain environmental conditions. In contrast, safety properties are *invariants*. Intuitively, a safety property states that "nothing bad happens" (i.e., that an acceptable state of affairs is maintained across all states of execution). In order to realise responsibilities, a role has a set of *permissions*. The type and amount of resources that can be exploited when carrying out the role are captured in an attribute known as role's *permission.* Permissions are the "rights" associated with a role. An *activity* is somewhat like a method in object-oriented term, or a procedure in a Pascal-like language. It corresponds to a unit of action that the agent may perform, which does not involve interaction with any other agents. Protocols, on the other hand, are the activities that do require interaction with other agents.

b) Interaction model

Dependencies and relationship among different roles involved in a multi-agent organization are inevitable. Such interactions are captured and represented in the interaction model. This model consists of a set of protocol definitions, one for each type of inter-role interaction.
A protocol definition consists of the following attributes:
purpose: brief textual description of the nature of the interaction.
initiator: the role(s) responsible for starting the interaction;
responder: the role(s) with which the initiator interacts;
inputs: information used by the role initiator while enacting the protocol;
outputs: information supplied by/to the protocol responder during the course of the interaction;
processing: brief textual description of any processing the protocol initiator performs during the course of the interaction

2) Design

In design phase, the abstract model derived in the analysis phase is transformed into models that can be easily implemented. The Gaia design process involves generating three models namely the agent model, the service model and the acquaintance model.

a) The agent model identifies the agent types that will make up the system, and the agent instances that will be instantiated from these types.

b) The services model identifies the main services that are required to realize the agent's role. Service can be termed as a function of an agent. For each service performed by an agent, it is important to identify its properties such as inputs, outputs, pre-conditions and post-conditions.

c) Tthe acquaintance model simply defines the lines of communication between the different agents. Defining communication lines indicates the existence of communication pathway and doesn't mean to define what and when messages are sent. An agent acquaintance model is simply a graph, with nodes in the graph corresponding to agent types and arcs in the graph corresponding to communication pathways.

D. MESSAGE (Methodology for Engineering Systems of Software Agents)

The MESSAGE [7] methodology has analysis and design phase. It extends the UML meta-model adding new meta-concepts (such as Agent, Goal and Task) in order to provide developers with a higher degree of expressiveness. A proper notation is also defined to represent graphically instances of these meta-concepts. Furthermore a process and a number of heuristic rules are provided to guide the developer in carrying out the analysis and design of a system under development. Basically the MESSAGE methodology comprises

1) A meta-model extending the UML meta-model and therefore adding new meta-concepts (such as Agent, Goal and Task) to those already considered in UML (e.g. Class, Actor).

2) A set of views focusing each one on specific aspects of the analysis and design models while hiding the complexity of the model as a whole.

3) A number of guidelines and heuristic rules helping the developer in building the analysis and design model and in using them to actually implement the system under development.

4) A proper notation that allows easily and intuitively representing the above views in a graphical (i.e. through diagrams) and textual (i.e. through schemas) way. For example the fact that an agent A (that is an instance of the Agent meta-concept) tries to achieve goal G (that is an instance of the Goal meta-concept) can in facts be represented in different ways depending on the notation (symbols) used.

E. TROPOS

Tropos [8] is a requirements engineering approach. It is known for its rigorous requirements analysis. The agent oriented software engineering methodology TROPOS, offers a structured development process and supporting tools for developing complex, distributed systems. It is a goal oriented process and displays the flexible nature of the agent in adapting to the environment. The software development process in TROPOS is structured in 5 main phases:

- Early requirements
- Late requirements
- Architectural design
- Detailed design
- Implementation

1) Early requirements stage

The main focus is on the understanding of the problem by studying an existing organizational setting; the output of this phase is an organizational model, which includes relevant actors and their dependencies. More precisely, during the first phase, the requirements engineer identifies the domain stakeholders and models them as social actors, who depend on one another for goals to be achieved, plans to be performed, and resources to be furnished. By clearly defining these dependencies, it is then possible to state the why, beside the what and how, of the system functionalities and, as a last result, to verify how the final implementation matches initial needs.

2) Late requirement stage

It deals with the analysis of the system-to-be within its operational environment along with relevant functions and qualities. During this phase, the conceptual model is extended including a new actor, which represents the system, and a number of dependencies with other actors of the environment. These dependencies define all the functional and non-functional requirements of the system-to-be.

3) The Architectural Design

In the architectural design stage, the system's global architecture is defined in terms of subsystems, interconnected through data and control flows. The subsystems are represented as actors and data/control interconnections are represented as system/actor dependencies. The architectural design also provides a mapping of the system actors to a set of software agents, each characterized by specific capabilities.

4) Detailed design

In this stage, the various architectural components are further defined in detail in terms of input, output, control and other relevant information. The Detailed Design phase aims at specifying agent capabilities and interactions. At this point, usually, the implementation platform has already been chosen and this can be taken into account in order to perform a detailed design that will map directly to the code.

5) Implementation

The Implementation activity follows step by step, in a natural way, the detailed design specification on the basis of the established mapping between the implementation platform constructs and the detailed design notions. In *implementation phase*, the agent oriented code is generated in ACL (agent communication language) with the help of certain tools like JADE, JADEx, and JACK.

F. PROMETHEUS

Prometheus [9] methodology supports the development of intelligent agents which use goals, beliefs, plan, and event. It provides support from specification to detailed design and implementation. The Prometheus methodology consists of three phases namely,

1) System Specification phase.

It focuses on identifying the basic functionalities of the system, along with inputs (percepts), outputs (actions) and any important shared data sources. It involves two activities such as determining the system's environment, and determining goals and functionality of the system.

2) Architectural Design phase

This phase uses the outputs from the previous phase to determine which agents the system will continue and how they will interact. Simply this phase defines agent types, the interaction between agents, and designs the system structure.

3) Detailed Design phase

This phase looks at the internals of each agent and how it will accomplish its tasks within the overall system. Simply this phase focuses on defining capabilities, internal events, plans and detailed data structures for each agent.

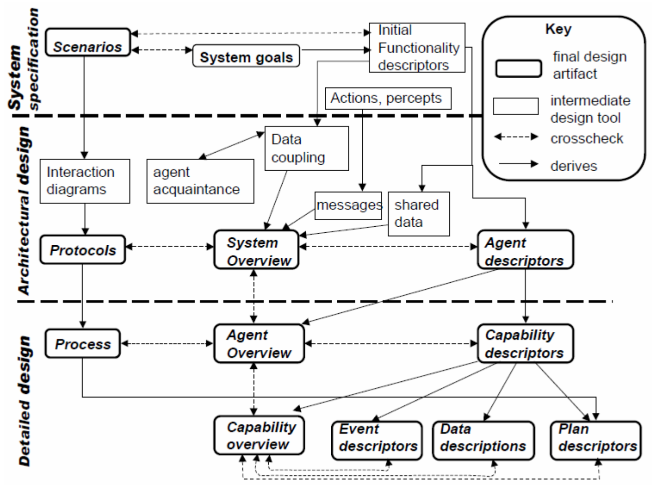

Figure.3 Prometheus Methodology

G. PASSI (Process for Agent Societies Specification and Implementation)

PASSI [10] is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies integrating design models and concepts from both the object-oriented software engineering and the MAS using the Unified Modeling language (UML) notation. PASSI methodology is made up of five models namely System requirement model, Agent society model, Agent implementation model, Code model and Deployment model. The models of PASSI are represented in the following figure 4.

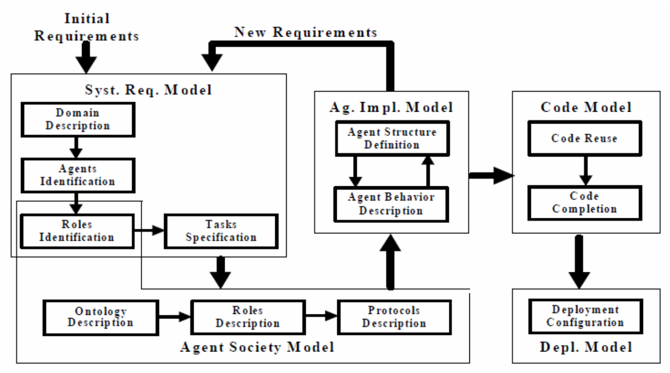

Figure.4 PASSI Methodology

1) System Requirement Model. An anthropomorphic model of the system requirements in terms of agency and target. It comprises four steps namely

a) Domain Description Phase

It is a functional description of the system composed of a hierarchical series of use case diagrams. Scenarios of the detailed use case diagrams are then explained using sequence diagram

b) Agent Identification Phase

From the use case diagram of the previous phase, the agents involved in a scenario and their respective functionalities are identified

c) Role Identification Phase

Roles identification is based on exploring all the possible paths of the Agents Identification diagram involving inter-agent communication. An agent may participate in different scenarios playing distinct roles in each. It may also play distinct roles in the same scenario.

d) Task Specification Phase

In this phase, the behavior of every agent is decomposed into tasks. Tasks generally encapsulate some functionality that forms a logical unit of work

2) Agent Society Model

A model of the social interations and dependencies between the agents playing a part in the solution. It comprises four steps namely

a) Role Identification Phase

As role is considered to be a social concept, this phase is also a part of Agent Society Model.

b) Ontology Description Phase

The ontology description yields two diagrams namely Domain Ontology Description diagram (domain ontology is represented as an XML schema) and Communication Ontology Description (communication is explained by a class diagram)

c) Role Description Phase

This phase models the lifecycle of an agent taking into account its roles, the collaborations it needs and the conversation it is involved in. This phase yields a collection of class diagram in which classes are used to represent roles.

d) Protocol Description Phase

The protocol used for each communication is described in this phase.

3) Agent Implementation Model

A model of the solution architecture in terms of classes and methods. It comprises two steps namely

a) Agent Structure Definition Phase

In this phase, definitions of the agents involved and their tasks and also the internal structure, revealing all the attributes and methods of the agent class together with its inner tasks classes are described

b) Agent Behaviour Description Phase

The flow of events by method invocation and the messages exchange in the multi-agent scenario are described in this phase.

4) Code Model

A model of the solution at the code level. It comprises two steps namely

a) Code Reuse Phase.

In this phase, the predefined patterns (code as well as design) of agents and tasks are reused.

b) Code Completion Phase.

This phase completes the body of the method from design diagram

5) Deployment Model

A model of the distribution of the system's part across hardware processing units, and their migrations. It comprises one step namely

a) Deployment Configuration Phase

This phase illustrates the location of agents, their movements and their communication pattern

H. SODA (Societies in Open and Distributed Agent Space)

SODA [11] is a methodology for the analysis and design of internet based application as multi-agent systems. SODA concentrates on inter-agent issues, like the engineering of societies and infrastructure for multi-agent system.

1) Analysis

During the analysis phase, the application domain is studied and modelled, the available computational resources and the technological constraints are listed, the fundamental application goals and targets are pointed out. analysis phase exploits three different models:

a) The role model

The application goals are modelled in terms of the tasks to be achieved, which are associated to roles and groups

a) The resource model

The application environment is modelled in terms of the services available, which are associated to abstract resources

b) The interaction model

The interaction involving roles, groups and resources is modelled in terms of interaction protocols, expressed as information required and provided by roles and resources, and interaction rules, governing interaction within groups.

2) Design

Design is concerned with the representation of the abstract models resulting from the analysis phase in terms of the design abstractions provided by the methodology. Differently from the analysis phase, a satisfactory result of the design phases is typically expressed in terms of abstractions that can be mapped one-to-one onto the actual components of the deployed system. The SODA design phase is based on three strictly related models:

a) The agent model – Individual and social roles are mapped upon agent classes

b) The society model – groups are mapped onto societies of agents, which are designed and organised around coordination abstractions *c) The environment model* – resources are mapped onto infrastructure classes, and associated to topological abstractions.

III. EVALUATION OF AOSE METHODOLOGIES

Several Agent-Oriented Software Engineering (AOSE) methodologies were developed and each has its own features and purpose. A methodology can be acceptable as a well-framed engineering approach, one and only when it provides a detailed description of the activities involved in the development lifecycle such as analysis, design, implementation, testing and maintenance. Thus lifecycle coverage is an important evaluating criterion for comparing several AOSE methodologies. The main core of this work is to perform comparative evaluation of several well-known agent-oriented methodologies based on lifecycle coverage criteria.

A) Lifecycle coverage in MAS-CommonKADS

MAS-CommonKADS is an agent-oriented methodology that covers the software development life cycle of a multiagent system such as analysis and design, through the development of seven models, that can be reused. MAS-CommonKADS is not complete with respect to lifecycle coverage as it lacks in implementation and testing issues

B) Lifecycle coverage in MaSE

MaSE addresses the traditional stages of software development process, however, it lacks in addressing the supportive activities [12]. MaSE covers requirement stage (goal capturing and applying use cases), analysis stage(role and task definition), design stage (construction and assembling of agents) and implementation stage(code generation). Testing stage has not been dealt in MaSE methodology.

C) Lifecycle coverage in GAIA

Lifecycle coverage is very limited with respect to Gaia. Gaia handles only analysis and design stage of the entire software development lifecycle [12]. The fact that the implementation phase and the testing phases are not covered by Gaia is a major drawback of the methodology. When it comes to implementation, the analysis and design parts are translated into UML notation and then the existing object-oriented languages are used for implementing. Verification and validation issues are not at all dealt within Gaia.

D) Lifecycle coverage in MESSAGE

The MESSAGE methodology extends the UML meta-model adding new meta-concepts (such as Agent, Goal and Task) in order to provide developers with a higher degree of expressiveness. The methodology defines proper notation to represent graphically the instances of these meta-concepts. The methodology supports the analysis and design phase of software development process and not proper emphasis on implementation and testing.

E) Lifecycle coverage in Tropos

Tropos supports the traditional stages of software development, yet, it does not deal with the supportive activities [12]. Tropos covers most of the development lifecycle phases such as analysis, design and implementation. However, the testing phase is not at all covered in Tropos.

F) Lifecycle coverage in Prometheus

Prometheus covers the system specification, high level design and detailed design activities with minor discussion on the implementation issues and on using design models for debugging agent based system. However, when it comes to lifecycle coverage, the support for implementation and testing is very limited.

G) Lifecycle coverage in PASSI

PASSI methodology covers the steps from analysis to deployment of an agent based system. PASSI is almost a complete agent oriented software engineering methodology as it exhibit all the development lifecycle phases such as analysis, design, implementation and testing too at some extent but not fully. When it comes to testing agent testing (unit testing) and society testing (integration testing) is discussed but the technique or framework through which the agent and society can be tested is not elaborated.

H) Lifecycle coverage in SODA

SODA is a methodology for the analysis and design of internet-based application as multi-agent system. Lifecycle coverage in SODA is very limited, that it covers only analysis and design but not implementation and testing phases. SODA explicitly focuses on how agent societies and environment are analysed and designed but not address intra-agent issues and thats why SODA is not a complete methodology and focuses instead on the social aspects of agent-oriented software engineering.

IV. COMPARISONS OF AOSE METHODOLOGIES

After careful study of various AOSE methodologies, we analysed and compared the life cycle coverage of different AOSE methodologies and we found that none of the AOSE methodologies possesses testing phase. Table II clearly indicates that there is a vacancy for software testing perception in the existing AOSE methodologies.

TABLE II LIFE CYCLE COVERAGE OF AOSE METHODOLOGIES

Sl.No	**Name of the Methodology**	**Life-Cycle Coverage**
1.	MAS-CommonKADS (1996)	Analysis Design
2.	MASE(1999)	Analysis

		Design
3.	GAIA (2000)	Analysis Design
4.	MESSAGE(2000)	Analysis Design
5.	TROPOS(2001)	Analysis Design Implementation
6.	PROMETHEUS(2002)	Analysis Design
7.	PASSI(2002)	Analysis Design Implementation
8.	SODA	Analysis Design

V. CONCLUSION

In this paper, we compared and evaluated almost eight existing AOSE methodologies such as MAS-CommonKADS, MaSE, Gaia, MESSAGE, Tropos, Prometheus, PASSI and SODA. The evaluation criteria adopted for comparison is development lifecycle coverage. Overall, all eight methodologies provide a reasonable support for analysis and design whereas implementation is touched only by very few methodologies such as Tropos and Passi. When it comes to testing, no methodologies dealt with it, stating existing object-oriented testing techniques can be used. But the fact is there are open issues concerning agent characteristics that are yet to be covered by existing testing techniques and there arises the emphasis on the specialized agent-oriented software testing technique. Thus our evaluation of AOSE methodologies states that none of the existing methodologies were complete with respect to development lifecycle coverage criteria.

REFERENCES

[1] Nwana.G, "Software Agents: An Overview", The Knowlwdge Engineering Review, 11(3).

[2] B. Henderson-Sellers and P. Giorgini. Agent-Oriented methodologies. Idea Group Inc., 2005.

[3] 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.

[4] C Iglesias, M Garijo, J González, J Velasco, "Analysis and Design of Multi-Agent system using MAS-CommonKADS", Intelligent Agent, Springer, 2001.

[5] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. "Multiagent systems engineering". International Journal of Software Engineering and Knowledge Engineering, 11(3):231–258, 2001.

[6] M. Wooldridge, N.R. Jennings, and D. Kinny, "The Gaia methodology for agentoriented analysis and design" Autonomous Agents and Multi-Agent Systems, 3(3), 2000.

[7] Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., & Massonet, P. (2001), "Agent-oriented analysis using MESSAGE/UML" In M. Wooldridge, G. Wei, & P. Ciancarini (Eds.), Agent-oriented software engineering *II* (p. 119-135). LNCS 2222. Berlin: Springer-Verlag.

[8] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini, "Tropos: An agent-oriented software development methodology", Journal of Autonomous Agents and Multi-Agent Systems, 8:203–236, May 2004.

[9] L. Padgham, J. Thangarajah, and M. Winikoff: "The Prometheus Design Tool – A Conference Management System Case Study." In: M. Luck and L. Padgham (Eds.): AOSE 2007, LNCS 4951, pp. 197–211, Springer-Verlag Berlin Heidelberg 2008.

[10] Massimo Cossentino. From requirements to code with the PASSI methodology. In Brian Henderson-Sellers and Paolo Giorgini, editors, Agent-Oriented Methodologies, pages 79–106. Idea Group Inc., 2005.

[11] Andrea Omicini, "SODA: Societies and Infrastructures in the Analysis and Design of Agent-based Systems", Agent Oriented Software Engineering, Springer,2001

[12] Federico Bergenti, Marie-Pierre Gleizes, Franco Zambonelli, " Methodologies and Software Engineering For Agent Systems", Kluwer Academic Publishers, 2004.