Evaluating the Lifecycle Coverage of GAIA Methodology

N.Sivakumar¹, K.Vivekanandan², S.Gayatri³
Department of computer Science and Engineering
Pondicherry Engineering College
Puducherry, INDIA

Abstract

Agent Oriented Software Engineering (AOSE) is one of the latest contenders in the historic development of programming paradigms. AOSE promises to be an exciting new software engineering approach with techniques more suited to the construction of large, complex software systems than those techniques belonging to the object oriented approach. To be a viable concept, there is a need for a uniform approach throughout the development life cycle of an agent oriented system, analogous to the unified process for objects oriented analysis and design. The agent-oriented paradigm can be considered a natural extension to the object-oriented (OO) paradigm. Agents differ from objects in many issues which require special modelling elements but have some similarities. The main objective of this paper is to evaluate the lifecycle coverage in existing AOSE methodology, say Gaia methodology. The purpose of this paper is to provide some insight of testing in GAIA software methodology.

Keywords- Software Agent, AOSE Methodology, Software Development Lifecycle, SoftwareTesting

1. Introduction

In recent years, agent-based systems have received considerable attention in both academics and industry. The agent-oriented paradigm can be considered a natural extension to the object-oriented (OO) paradigm [2]. A multi-agent system is a software systems made up of multiple independent and encapsulated loci of control (i.e., the agents) interacting with each other in the context of a specific application viewpoint. If agents are to realise their potential as a software engineering paradigm, then it is necessary to develop software engineering techniques that are specifically tailored to them. Existing software development techniques (for example, object-oriented analysis and design are unsuitable for this task. There is a fundamental mismatch between the concepts used by object-oriented developers (and indeed, by other mainstream software engineering paradigms) and the agent-oriented view. In particular, extant approaches fail to adequately capture an agent’s flexible, autonomous problem-solving behaviour, the richness of an agent’s interactions, and the complexity of an agent system’s organizational structures. Hence, there is a need of new software development methodologies that support the design and implement organizations of agents able to interact with one another in order to achieve some common or individual goal. AOSE methodologies mainly try to suggest a clean and disciplined approach to analyze, design and develop multi-agent systems, using specific methods and techniques. AOSE methodologies [1] typically start from a meta-model, identifying the basic abstractions to be exploited in development. On this base, they exploit and organize these abstractions so as to define guidelines on how to proceed in the analysis, design, and development, and on what output to produce at each stage. Meta-model enables checking and verifying the completeness and expressiveness of a methodology by understanding its deep semantics, as well as the relationships among concepts in different languages or methods.

Some of the AOSE methodologies are MaSE, Tropos, Gaia, Prometheus, Passi, Soda, Message etc., AOSE methodologies offer different conceptual frameworks, notations and techniques, thereby provide a platform to make the system abstract, generalize, dynamic and autonomous. A great deal of research in AOSE focuses in proposing methodologies that identifies the guidelines and the abstraction to be exploited in the various phases of agent-based software development such as analysis, design and implementation. However, very little attention has been paid towards the testing phase of agent-based software development process [3], [4], stating that testing can be done by extending the object-oriented testing technique. The Gaia methodology is both general, in that it is applicable to a wide range of multi-agent systems, and comprehensive, in that it deals with both the macro-level (societal) and the micro-level (agent) aspects of systems. Gaia is founded on the view of a multi-agent system as a computational organisation consisting of various interacting roles [8]. From the software development life-cycle point of view, GAIA mentions the requirements, architectural design and detailed design to some
ext. Implementation, Testing & Debugging, Deployment and Maintenance are not also discussed GAIA.

2. Gaia Methodology

GAIA is one of the first methodologies which is specifically tailored to the analysis and design of agent-based systems. Its main purpose is to provide the designers with a modelling framework and several associated techniques to design agent-oriented systems [9], [10]. However, the original version of Gaia suffered from the limitations of being suitable for the analysis and design of closed multiagent systems and of adopting non-standard notation techniques. The official extension of Gaia, to which we will refer here as Gaia v.2, extends Gaia based on the key consideration of organization abstractions and environmental abstractions. The motivation behind Gaia is that existing methodologies fail to represent the autonomous and problem-solving nature of agents; they also fail to model agents’ ways of performing interactions and creating organizations. Using Gaia, software designers can systematically develop an implementation-ready design based on system requirements.

The Gaia process starts with the analysis phase, whose aim is to collect and organize the specification, which is the basis for the design of the computational organization (which implies defining an environmental model, preliminary roles and interaction models, and a set of organizational rules). Then, the process continues with the architectural phase, aimed at defining the system organizational structure in terms of its topology and control regime (possibly exploiting design patterns), which, in turn, helps to identify complete roles and interaction models.

An overview of the methodology and models is shown in Figure 1.

2.1 Analysis phase.

The main goal of the analysis phase is to organize the collected specifications and requirements for the system-to-be into an environmental model, preliminary role and interaction models, and a set of organizational rules for each of the sub organizations composing the overall system.

The analysis phase includes the identification of:

1) The Organizations: This includes the goals of the organizations that constitute the overall system and their expected global behavior. At this step it is important to identify useful decomposition of the global organization into sub-organizations. Generally speaking, such sub-organizations can be found when there are portions of the overall system that (1) exhibit a behavior specifically oriented towards the achievement of a given sub-goal, (2) interact loosely with other portions of the system, or (3) require competencies that are not needed in other parts of the system.

2) The Environment Model: The environmental model that represents the environment (in terms of computational variables/resources) in which the multiagent system will be situated. It is difficult to provide general modelling abstractions and general modelling techniques because, the environments for different applications can be very different in nature and also because they are somehow related to the underlying technology. To develop a reasonably general approach (without the ambition for it to be universal), we suggest treating the environment in terms of abstract computational resources, such as variables or tuples, made available to the agents for sensing (e.g., reading their values), for effecting (e.g., changing their values) or for consuming (e.g., extracting them from the environment). Following such identification, the environmental model (in its simplest form) can be viewed as a list of resources, each associated with a symbolic name, characterized by the type of actions that the agents can perform on it, and possibly associated with additional textual comments and descriptions.
3) *The Preliminary Role Model*: The Preliminary Role Model identifies the “basic skills” that are required by the organization to achieve its goals, as well as the basic interactions that are required for the exploitation of these skills. Such identification activities may even be facilitated if a goal-oriented early requirements analysis has already modelled the characteristics of the system in terms of actors involved and their goals. Given the identification of the basic skills and of their basic interaction needs, respectively, the analysis phase can provide a *preliminary definition* of the organization’s roles and protocols. However, this definition cannot be completed at this stage. In fact, the basic skills (or *preliminary roles*) can only truly become organizational roles when it is known how and with which other roles they will interact.

The roles model identifies the key roles in the system [5]. Here a role can be viewed as an abstract description of an entity’s expected function. In other terms, a role is more or less identical to the notion of an *office* in the sense that “prime minister”, “attorney general of the United States”, or “secretary of state for Education” is all offices. Such roles (or offices) are characterized by two types of attribute.

- **The permissions/rights associated with the role.**

A role will have associated with it certain permissions, relating to the type and the amount of resources that can be exploited when carrying out the role. In our case, these aspects are captured in an attribute known as the role’s permissions.

- **The responsibilities of the role.**

A role is created in order to *do* something. That is, a role has certain functionality. This functionality is represented by an attribute known as the role’s responsibilities. In Gaia, responsibilities are divided into two types: liveness properties and safety properties (Manna & Pnueli, 1995). Liveness properties intuitively state that “something good happens,” that is, describes those states of affairs that an agent must bring about, given certain conditions. In contrast, safety properties are invariants. Intuitively, a safety property states that “nothing bad happens,” that is, that an acceptable state of affairs is maintained.

4) *Preliminary Interaction Model*: The Gaia interaction model captures the dependencies and relationships between the various roles in the MAS organization in terms of one protocol definition for each type of inter-role interaction. Since the roles model is still preliminary at this stage, the corresponding protocols model must also necessarily be preliminary, for the same reasons.

5) *The Organizational Rules*: The preliminary roles and interaction models capture the basic characteristics, functionalities, and interaction patterns that the MAS system must realize, independently of any predefined organizational structure. However, as previously stated, there may be general relationships between roles, between protocols, and between roles and protocols that are best captured by organizational rules. In Gaia, the perspective on organizational rules is consistent with that on roles’ responsibilities—organizational rules are considered *responsibilities* of the organization as a whole. Accordingly, it is possible to distinguish between safety and liveness organizational rules.

*Liveness rules* define how the dynamics of the organization should evolve over time. These can include, for example, the fact that a role can be played by an entity only after it has played a given previous role or that a given protocol may execute only after some other protocol.

*Safety rules* define time-independent global invariants for the organization that must be respected. These can include, for example, the fact that a given role must be played by only one entity during the organization’s lifetime or that two roles can never be played by the same entity.

### 2.2 Design phase

The design phase includes the following sub-phases:

1) **Architectural Design.**

- **Choosing the Organizational Structure.**

The choice of the organizational structure is a highly critical phase in MAS development, affecting all subsequent phases. Definition of the overall architecture of the systems, i.e., of the organizational structure, taking care that it accommodates all preliminary roles and interactions identified in the analysis phase, and taking care that the adopted structure facilitates the enactment of the organizational rules.

- **Organizational Rules.**

Another factor impacting on the choice of the organizational structure (primarily of the control regime) is the need for the MAS to respect organizational rules and to be able to enact them during execution. Some types of organizational rules, typically safety ones, may express very direct and intuitive constraints directly driving the adoption of specific organizational structures (e.g., constraints specifying that two roles have to be played by the same agent or that the same agent cannot play two specific roles concurrently). A great number of different organizational structures may be available for designers to better deal with functional and efficiency requirements. Nevertheless, it is highly probable that a reduced subset of these structures is normally adopted. This
opens up the opportunity to exploit, in this phase, existing catalogs of organizational patterns.

- **Exploiting Organizational Patterns.**

This is the stage where the availability of a catalogue of possibly modular and composable “organizational structures,” describing in much more detail than we have done how and when to exploit a specific structure, will greatly help the designer (Shaw & Garlan, 1996) and will effectively complement our general guidelines. This helps the designer to re-use both well-documented and motivated design choices and the design work related to the representation of such a structure.

- ** Representing the Organizational Structure.**

Once the identification of a suitable organizational structure for the MAS is complete, the designer has to determine how to effectively represent it. The obvious means by which to formally specify an organization is to explicate the inter-role relationships that exist within it (to represent the topology of the organizational structure) and their types (to represent the control regime of the organization). To give a few examples: a control relationship may identify an authority relationship of one role over another, in which a role can (partially) control the actions of another role; a peer relationships may express the fact that two roles have equal status; a dependency relation may express the fact that one role relies on some resources or knowledge from other roles for its accomplishment.

- **Completion of Role and Interaction Models.**

Once the organizational structure is defined, the preliminary role and interaction models (as identified in the analysis phase) can be transformed into complete roles and interactions models (describing in detail the characteristics of all roles and stages involved in the MAS that will actually be produced). In fact, with the organizational structure identified, the designer knows which roles will have to interact with which others (as derived from the organization topology) and which protocols will have to be executed (as derived from the control regime of the organization). Thus, the completion of the role and protocol model amounts to:

- Defining all the activities in which a role will be involved, as well as its liveness and safety responsibilities;
- Defining organizational roles – those whose presence was not identified in the analysis phase and whose identification derives directly from the adoption of a given organizational structure;
- Completing the definition of the protocols required by the application, by specifying which roles the protocol will involve; and
- Defining organizational protocols – those whose identification derives from the adopted organizational structure.

2) **Detailed Design.**

The detailed design phase is responsible for eventually identifying the agent model and the services model that, in turn, act as guidelines for the actual implementation of agents and their activities.

- **Definition of the Agent Model.**

In the Gaia context, an agent is an active software entity playing a set of agent roles. Thus, the definition of the agent model amounts to identifying which agent classes are to be defined to play specific roles and how many instances of each class have to be instantiated in the actual system.

- **The Services Model.**

As its name suggests, the aim of the Gaia services model is to identify the services associated with each agent class or, equivalently, with each of the roles to be played by the agent classes. Thus, the services model applies both in the case of static assignment of roles to agent classes and in the case where agents can dynamically assume roles.

### 3. Life Cycle Coverage of GAIA

A methodology is the set of guidelines for covering the whole lifecycle of system development. 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 gathering, analysis, design, implementation, and testing. Lifecycle coverage is an important evaluation criteria for a methodology [7] because a detailed description of the activities included in the development lifecycle would enhance the appropriate use of a methodology and increase its acceptability as a well-formed engineering approach.

Gaia methodology is incomprehensive with respect to lifecycle coverage. The Gaia methodology is based on a well-founded organizational metaphor and exploits in a clean and rational way a suitable set of organizational abstractions. Therefore, we consider Gaia to be an effective general-purpose methodology for the development of MAS in a large set of scenarios (from distributed workflow management systems, to systems for the control of physical and manufacturing processes), Gaia does not deal with implementation issues and considers the output of the design phase as a specification that can be
picked up by using a traditional method or that could be implemented using an appropriate agent-programming framework. The testing stage is not at all covered by Gaia. Regarding the software engineering models, we agree with all the responses that GAIA adhere to an iterative development process rather than sequential or waterfall ones. Developers are encouraged to freely move between development phases and steps although there is a tendency in GAIA that specific activity are described in sequence. In terms of the development perspectives, GAIA seems to suit a top-down approach. Traditional testing techniques cannot be applied towards software agents as the agents communicate through message passing rather than method invocation. Existing software testing techniques [6] (both conventional and object-oriented) cannot be applied over software agents as these agents are designed to be autonomous, proactive, collaborative and ultimately intelligent.

4. Conclusion

Software Agent technology has drawn much attention as the preferred architectural framework for the design of many distributed software systems. Agent-based systems are often featured with intelligence, autonomy, and reasoning. Such attributes are quickly becoming alluring to both legacy and new systems. Agents are building blocks in these software systems, while combinations of attributes are composed to form the software entities. In this paper, an agent oriented software methodology named Gaia is analyzed and evaluated. It is seen that Gaia covers the requirements, architectural design and detailed design to some extent. Implementation, Testing & Debugging, Deployment and Maintenance are not also discussed GAIA. Testing is an important activity in software development in order to assure the correctness of software. However, testing is often disregarded in most agent oriented methodologies, mainly because they focus on analysis and design activities, and consider that implementation and testing issues can be performed using traditional techniques. But multi-agent systems implementation has some features that make it distinctive from traditional software. Hence, a separate testing technique is needed in order to ensure the correctness of an agent oriented software.

5. References