International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Volume 3, Issue 2 (Mar-Apr 2015), PP. 62-66

STUDY OF AGENT ASSISTED METHODOLOGIES FOR DEVELOPMENT OF A SYSTEM

Reshma Kazmi, Brijesh Pandey, **Namarata dhandha**
(Goel Institute of Technology and Management)

Abstract. Agent Assisted Methodologies have become an important subject of research in advance Software Engineering. Several methodologies have been proposed as, a theoretical approach, to facilitate and support the development of complex distributed systems. An important question when facing the construction of Agent Applications is deciding which methodology to follow. Trying to answer this question, a framework with several criteria is applied in this paper for the comparative analysis of existing multiagent system methodologies. The results of the comparative over two of them, conclude that those methodologies have not reached a sufficient maturity level to be used by the software industry. The framework has also proved its utility for the evaluation of any kind of Agent Assisted Software Engineering Methodology.

I. INTRODUCTION

The aim of state of the art is to provide an overview of the rapidly evolving area of multi-agent systems (MAS) and its related methodologies. This leads to a discussion of what makes an agent oriented methodologies that can be used to build MAS. In this chapter literature concerning agent systems and agent oriented methodologies is reviewed in detail. The field of agent oriented methodologies is examined. The aim is establishing the characteristics of agent oriented methodologies, and seeing how these characteristics are suited to develop multi-agent systems. An analysis of agent oriented methodologies is examined giving a clearer picture of their application domain, advantages and limitations.

Designing and building high quality industrial-strength software is difficult. Indeed, it has been claimed that such development projects are among the most complex construction tasks undertaken by humans. Against this background, a wide range of software engineering paradigms have been devised (e.g., procedural programming, structured programming, declarative programming, object-oriented programming, design patterns, application frameworks and component-ware). Each successive development either claims to make the engineering process easier or to extend the complexity of applications that can feasibly be built. Although there is some evidence to support these claims, researchers continually strive for more efficient and powerful software engineering techniques, especially as solutions for ever more demanding applications are required.

This paper will argue that analyzing, designing and implementing software as a collection of interacting, autonomous agents (i.e., as a *multi-agent system* represents a promising point of departure for software engineering. While there is some debate about exactly what constitutes an autonomous agent and what constitutes interaction, this work seeks to abstract away from particular dogmatic standpoints. Instead, we focus on those characteristics for which there is some consensus. From this standpoint, the paper's central hypothesis will be advanced: for certain classes of problem (that will be defined), adopting a multi-agent approach to system development affords software engineers a number of significant advantages over contemporary methods. Note that we are not suggesting that multi-agent systems are a silver bullet there is no evidence to suggest they will represent an order of magnitude improvement in software engineering produc2tivity. However, we believe that for certain classes of application, an agent-oriented approach can significantly improve the software development process.

In seeking to demonstrate the efficacy of the agent-oriented approach, the most compelling form of analysis would be to quantitatively show how adopting such techniques had improved, according to some standard set of software metrics, the development process in a range of projects. However, such data is simply not available (as it is still not for more established methods such as object-orientation). However, there are compelling arguments for believing that an agent-oriented approach will be of benefit for engineering certain complex software systems. These arguments have evolved from a decade of experience in using agent technology to construct large-scale, real world applications in a wide variety of industrial and commercial domains.

The contribution of this paper is twofold. Firstly, despite multi-agent systems being touted as a technology that will have a major impact on future generation software ("pervasive in every market by the year 2000" and "the new revolution in software"), there has been no systematic evaluation of *why this may be the case*. Thus, although there are an increasing number of deployed agent applications for a review, nobody has analysed precisely what makes the paradigm so effective. This is clearly a major gap in knowledge, which this paper seeks to address. Secondly, there has been comparatively little work on viewing multi-agent systems as a software engineering. This shortcoming is rectified by recasting the essential components of agent systems into more traditional software engineering concepts, and by examining the impact on the software engineering life-cycle of adopting an agent-oriented approach.

II. THE NATURE OF COMPLEX SOFTWARE SYSTEMS

Industrial-strength software is complex in nature: it is typically characterised by a large number of parts that have many interactions. Moreover this complexity is not accidental [4]: it is an innate property of the types of tasks for which software is used. The role of software engineering is therefore to provide structures and techniques that make it easier to handle this complexity. Fortunately, this complexity exhibits a number of important regularities:

- Complexity frequently takes the form of a hierarchy1. That is, the system is composed of inter-related sub-systems, each of which is itself a hierarchy. The precise nature of the organisational relationships varies between sub-systems, although some generic forms
- (such as client-server, peer, team) can be identified. Organisational relationships are not static: they can, and frequently do, vary over time.
- The choice of which components in the system are primitive is relatively arbitrary and isdefined very much by the observer's aims and objectives.
- Hierarchic systems evolve more quickly than non-hierarchic ones of comparable size. In other words, complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms, than if there are not.
- It is possible to distinguish between the interactions *among* sub-systems and the interactions *within* sub-systems. The latter are both more frequent (typically at least an order of magnitude more) and more predictable than the former. This gives rise to the view that complex systems are nearly decomposable. Thus, sub-systems can be treated almost as if they are independent of one another, but not quite since there are some interactions between them. Moreover, although many of these interactions can be predicted at design time, some cannot.

III. AGENT ORIENTED METHODOLOGIES

In order to be able to perform a comprehensive literature review for the agent oriented methodologies, "what the meaning of the methodology is" needs to be precisely defined before starting this discussion. A good methodology should provide the models for defining the elements of the multi-agent environment (agents, objects and interactions). A good methodology should also provide the design guidelines for identifying these elements, their components and the relationships between them. Any methodology aims to provide a set of guidelines that covers the whole lifecycle of the system development. The guidelines should cover both the technical as well as the management aspects.

It is also important for a methodology to provide notations which allow the developers to model the target system and its environment. In addition to the methodology, there are also tools that support the use of such methodologies. For instance, diagram editors help developers drawing symbols, models described in the methodology. The Rational Unified Process (RUP) is a good example of a software engineering methodology (Kruchten 2000). It uses the notation described in the Unified Modelling Language (UML) (Booch 1998) and its typical tool support is Rational Rose.

IV. GAIA METHODOLOGY

Gaia methodology (Wooldridge 2000) was developed by Wooldridge et-al. for analysis and design of agent systems. Gaia is enhanced by Zambonelli in 2003. Gaia is a general methodology that supports both levels of Micro and Macro development of agent systems. Micro level relates to agent structure while Marco level relates to agent society and organizational structure. Gaia includes an analysis and design phase and does not explicitly support an implementation phase.

Gaia starts with the analysis phase as is given in figure 1, whose aim is to collect and organize the specification. This is the basis for the design of the computational organization. Then it continues with the design phase, which aims to define the system organizational structure. The definition is in terms of the system's topology and control system to identify the agent model and service model.

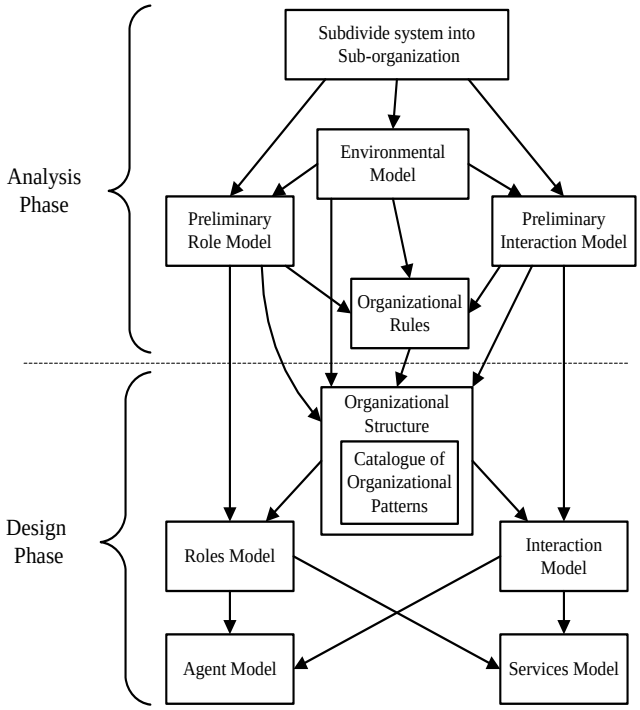

Figure 1 Gaia methodology Models

V. HLIM METHODOLOGY

HLIM methodology is considered to be one of the important agent oriented methodologies. It was developed in 1999 by M. Elammari and W. Lalonde. It allows the development of agent based systems from user requirements. The methodology models the external and internal behavior of agents. It also provides a means for both the visualization of the behavior of systems' agents and the definition of how the behavior is achieved. The methodology provides a systematic approach for generating system definitions from high-level designs which can be implemented. The methodology captures effectively the complexity of agent systems, agents' internal structure, relationships, conversations, and commitments.

Figure 3.3 shows the models of HLIM methodology. The HILM methodology consists of two phases (the **discovery phase** and **definition phase**). The discovery phase is an exploratory phase that leads to the high-level model definition. The agents are discovered and their high-level behaviour is identified. The discovery phase includes only the high-level model.

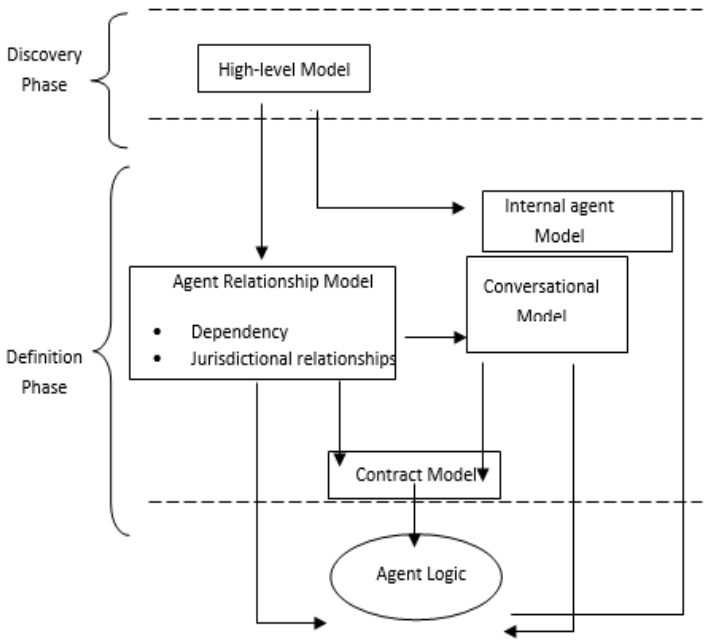

Figure 2 HLIM methodology Models

VI. THE PASSI METHODOLOGY

The PASSI methodology is considered to be one of the important agent oriented methodologies that are object oriented based methodologies. It was developed in 2002 by Cossentino and Potts. PASSI is composed of five models that address different design concerns and twelve steps in the process of building a model.

PASSI uses UML as the modelling language because it is widely accepted both in the academic and industrial worlds. Its extension mechanisms facilitate the customized representation of agent-oriented designs without requiring a completely new language. Extension mechanisms here refer to constraints, tagged values and stereotypes. The models and phases of PASSI are (see figure 3):

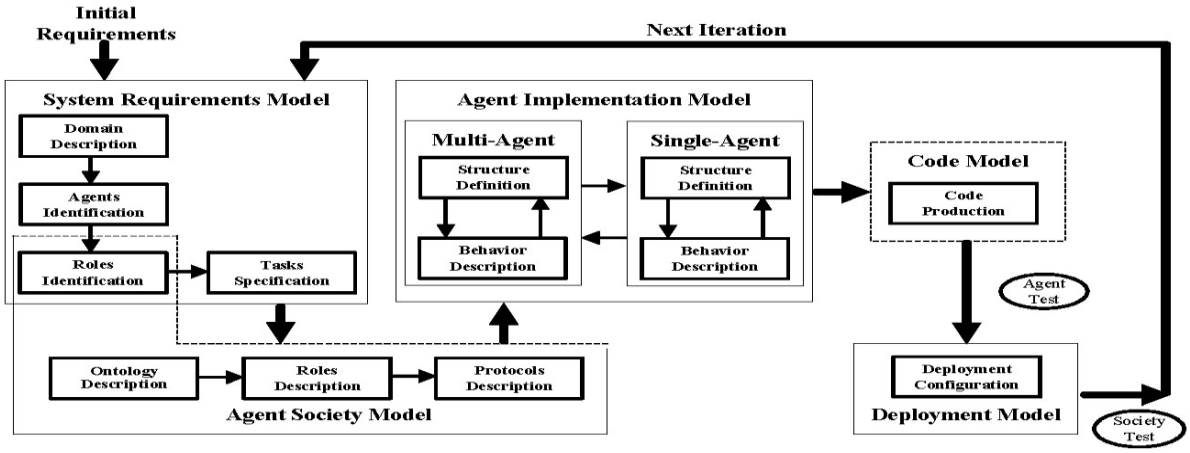

Figure 3 models and phases of the PASSI methodology

VII. TROPOS METHODOLOGY

The Tropos methodology is considered to be one of the important agent oriented methodologies. It was developed in 2003 by J. Castro, M. Kolp, and J. Mylopoulos.

The Tropos methodology is intended to support all analysis and design activities in the software development process, from application domain analysis down to the system implementation. It takes into account both inter-agent and intra-agent issues. In particular, Tropos rests on the idea of building a model of the system-to-be and its environment that is incrementally refined and extended. Tropos methodology provides a common interface to various software development activities, as well as a basis for documentation and evolution of the software. Tropos differentiates between an early and a late requirements phase, and between architectural design and detailed design. Tropos consists of the following three main phases: **Requirement analysis phase, Design phase** and **Implementation phase**. Figure 4 illustrates the models of Tropos methodology.

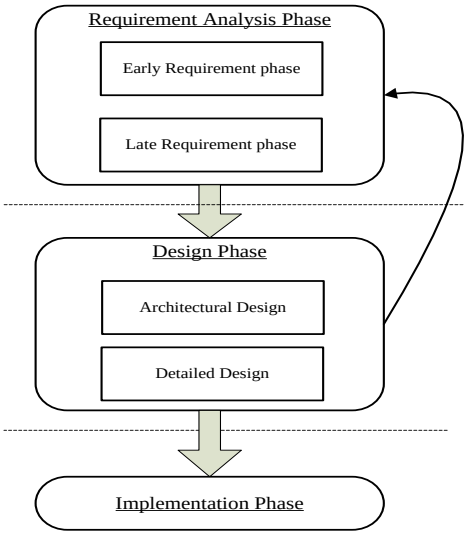

Figure 4 models and phases of the Tropos methodology

VIII. PROPOSED METHODOLOGY

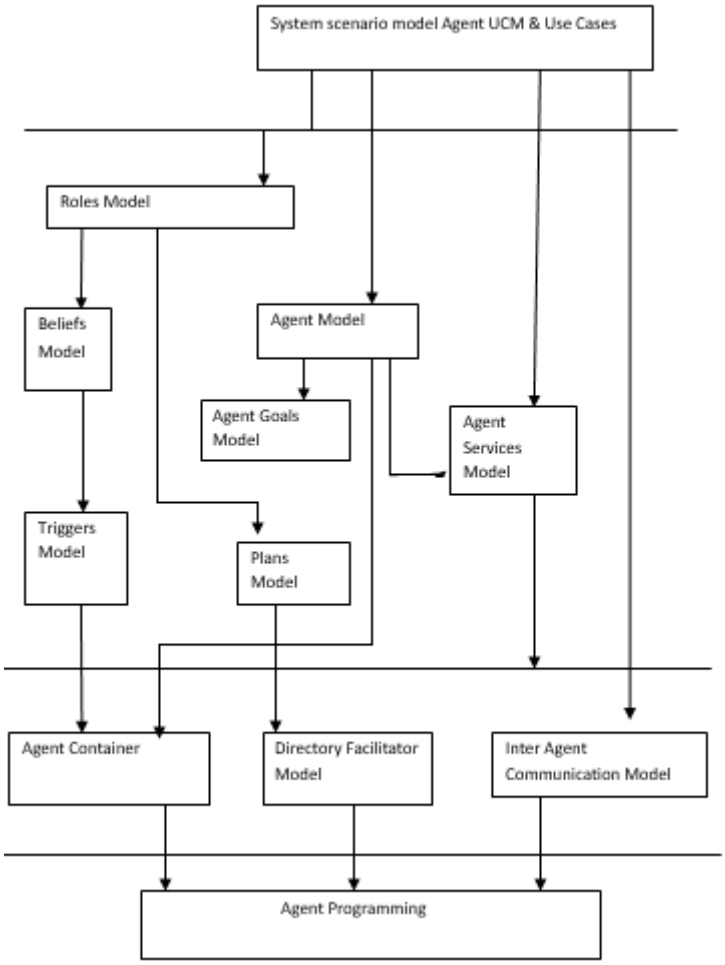

Figure 5 MBA Model

Eliminates the shortcomings faced by most other existing methodologies. The second step is that we demonstrate the new proposed methodology through the use of a case study representing a project clearance system. This case study refers to the collection of detailed information about how the proposed methodology works.

IX. MAB MODEL METHODOLOGY

We represent the proposed methodology MAB model. This methodology is constructed in order to as a reliable systematic approach that proves a milestone for software development lifecycle .The proposed methodology covers the most important characteristics of multi-agent systems.

The new methodology deals with the agent concept as a high level abstraction capable of modeling the complex system. The new methodology also include well known techniques for requirement gathering for user agency communication and links them to domain analysis and design models such as UCMS,UML Use Case Diagrams, Activity diagrams , FIPA-ACL, etc. Furthermore, it supports simplicity and ease of use as well as traceability.

MAB model is composed of four main phases, a System requirements phase, an analysis phase, a design phase and an implementation phase. Figure illustrates the detail of MAB methodology. The next few sections present a more detailed discussion of each of the four phases.

SYSTEM SCENARIO MODEL: the model is designed with the help of the use case map and use case diagrams to define the scenario of the system .In this case study we have designed the project clearance scenario. Where the system scenario is discussed in detailed.

ROLE MODEL: this model defines the role of the agent within the system. In the project clearance system the agents are identified and their rolesdefined for e.g, The role of the DFO within the system is to verify the details of the project and then forward it to the Ministry or return it back to the User agency for further details.

BELIEFS MODEL: here the agent's beliefs are discussed, i.e. the area or jurisdiction of the particular agent and their specific belief in persuing that tasks. i.e. all the agents that are playing an important role in the project clearance.Theymay be the DFO, Local Bodies and many others who have their own perspective of judging it.

AGENT MODEL: All the agents that pay an important role in MAB architecture. Here the agents work is to judge the proposals and forward them to the higher authorities for decision after marking their remarks.

X. CONCLUSIONS

In this article, we have described why we perceive agents to be a significant technology for software engineering. We have discussed in detail how the characteristics of certain complex systems appear to indicate the appropriateness of an agent-based solution: as with objects before them, agents represent a natural *abstraction* mechanism with which to decompose and organize complex systems. In addition, we have summarized some of the key issues in the specification, implementation, and verification of agent-based systems, and drawn parallels with similar work from more mainstream computer science. In particular, we have shown how many of the formalisms and techniques developed for specifying, implementing, and verifying agent systems are closely related to those developed for what are known as *reactive* systems in mainstream computing. Finally, we have described some of the pitfalls of agent-based development. Software engineering for agent systems is at an early stage of development, and yet the widespread acceptance of the concept of an agent implies that agents have a significant future is software engineering. If the technology is to be a success, then its software engineering aspects will need to be taken seriously. Probably the most important outstanding issues for agent-based software engineering are: (i) an understanding of the situations in which agent solutions are appropriate; and (ii) principled but *informal* development techniques for agent systems. While some attention has been given to the latter (in the form of analysis and design methodologies for agent systems, almost no attention has been given to the former

REFERENCES

[1] A. van Lamsweerde, and E. Letier, “Handling Obstacles in Goal-Oriented Requirements Engineering,” IEEE Transactions on Software Engineering vol. 26(10), pp. 978-1005, 2000.

[2] A. Cockburn, "Structuring Use Cases with Goals,” Journal of Object-Oriented Programming, Sep-Oct, 1997 and Nov-Dec, 1997.

[3] 3.S. A. DeLoach and M. Wood, "Developing Multiagent Systems with agentTool," in Y.Lesperance and C. Castelfranchi, editors, Intelligent Agents VII - Proceedings of the 7th

[4] International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000). Springer Lecture Notes in AI, Springer Verlag, Berlin, 2001.

[5] P. K. Harmer, G. B. Lamont, G.B, "An Agent Architecture for a Computer Virus Immune System," in Workshop on Artificial Immune Systems at Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, July 2000.

[6] G. J. Holzmann, “The Model Checker Spin,” IEEE Transactions On Software Engineering, vol. 23(5), pp. 279-295, 1997.

[7] M. Wooldridge (1997) “Agent-based software engineering” IEE Proc. on Software Engineering,144 (1) 26-37.

[8] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens (1989) “Concurrent MetateM: A framework for programming in temporal logic” REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (LNCS Volume 430), 94-129. Springer-Verlag

[9] J. R. Marden and A. Wierman, “Overcoming limitations of game-theoretic distributed control,” in Proc. 47th IEEE Conf. Decision Control, Dec. 2009, pp. 6466–6471.