Review on Multi-Agent Oriented Software Engineering Implementation

N.R. Genza*
Department of Computer Science, College For Legal Studies Yola, Adamawa State, Nigeria
*Nubiya2001 {at} yahoo.co.uk

E.S. Mighele
Department of Computer Science,
Delta State School of Marine Technology Burutu.
Delta State, Nigeria

Abstract— **Agent Oriented Software Engineering (AOSE) is an exciting and promising approach for solving complex and real world problems. It is crucial for industrial and commercial application as these systems are required to operate in increasingly complex, open, dynamic, unpredictable and inherently high interactive environments. This work aims to engineer complex systems with autonomous entities and also to determine the core phases of software engineering development and the different methodology. It has several benefits compared to existing development approaches in modeling, designing and implementing computer systems. Thus a robust and scalable software system requires an autonomous agent.**

Keywords- Agent-Oriented Software Engineering; Information Communication Technology (ICT); Object Oriented Programming; Multi-Agent Systems (MASs); Multiagent, Generic Architecture for Information Availability (Gaia), Multiagent Systems Engineering Methodology (MaSE)

I. INTRODUCTION

Agent and Multiagent Systems (MASs) are being described as a new paradigm for the research field of Software Engineering to face the complexity of today's Information Communication Technology (ICT) scenarios such as agent based air traffic control system (known as OASIS) [1], network control, transmission and switching, service management and network management[2], agent-based system for monitoring and diagnosing faults in nuclear power plants [3], spacecraft control [4], and climate control [5]. For instance, several industrial experiences already testify to the advantages of using agents in manufacturing processes [6, 7], Web services and Web-based computational markets [8] and distributed network management [9]. However, to become a new paradigm for software industry, a robust, easy to use methodologies and emergent tools have to be developed to meet these challenges.

MASs represent a general purpose paradigm for software development. One emergent technique is the development of multiagent systems. However, the academic community as well as industry, are still trying to determine which problem call for a multiagent approach.

A number of methodologies exist for building multiagent system. The methodology ranges from extensions of existing object oriented methodologies to new agent oriented techniques which offer a new perspective for developing multiagent systems by increasing the level of abstraction the developer uses to analyse and design the system. Agent based computation promotes designing and developing applications in terms of autonomous software entities (Agents), situated in an environment that can flexibly achieve their goals by interacting with one another in terms of high level protocols and languages.

Thus, agent orientation can serve as a useful paradigm in software engineering which has been referred to as agent oriented software engineering (AOSE). The core phases of software engineering are analysis and design phase, these two phases are mostly affected with the evolution of software engineering paradigm.

The rest of the article is structured as follows. Section II discusses on related research, section III Software Engineering Environment, Section IV Nature of Complex System, Section V Motivation for Software Engineering, Section VI Software Engineering Methodology Classification, section VII Multi-Agent Oriented Software Engineering Methodology, Section VIII Issues in Multi-Agent Oriented Software Engineering Methodology the Software Engineering Methodology Criteria (Section 3.2 discusses on the multiagent system engineering methodology (MASE) in comparison to Gaia; and section 4 discusses on issues in multiagent oriented software engineering methodology.

II. RELATED RESEARCH

There have been several proposed methodologies for analyzing, designing and building multiagent system [10]. The majority of this research is centred on existing object oriented or knowledge-based methodologies. In fact, most syntax adopted by the model was taken from the Unified Modeling Language. However, only few brought up the idea of using Agent Oriented approach for solving complex problems.

Actually, Agent-Oriented Software Engineering Methodology builds upon the work of many agent based approaches; it takes many ideas and combines them to form a complete methodology. MASs, more than an effective technology, represent a novel general-purpose

paradigm for software development and its main advantage over other methodologies is its scope of completeness [11].

The validation of agent-oriented techniques, come from a qualitative analysis of how well the techniques addresses the principle that allow software engineering techniques to deal with complex problems proposed by Booch: abstraction, decomposition and hierarchy [12, 13]. They leave "understanding of the situations in which agent solutions are appropriate" as an outstanding issue [13].

The European Institute for Research and Strategic Studies in Telecommunications (EURESCOM) used a different strategy in 1999 when they began a project to explore the use of agent technologies with the European telecommunications industry. One of the main objectives of the project is to "define guidance for the identification of application areas where an agent-based approach is more relevant than other approaches". They came up with the following guidelines to evaluate agent-oriented approach [14].

1. An agent-oriented approach is beneficial in situations where complex/diverse types of communication are required.
2. An agent-oriented approach is beneficial when the system must perform well in situations where it is not practical/ possible to specify its behavior on a case-by-case basis.
3. An agent-oriented approach is beneficial in situations involving negotiation, co-operation and competition among different entities.
4. An agent-oriented approach is beneficial when the system must act autonomously
5. An agent-oriented approach is beneficial when it is anticipated that the system will be expanded, modified or when the system purpose is expected to change. However, these guidelines provides good understanding of the criteria to decide upon but there are still no clear answer.

III. SOFTWARE ENGINEERING ENVIRONMENT

The environment in which a multiagent system is situated is of fundamental importance in the analysis, design and operation of the system. Despite the fact, only few multiagent methodologies include modeling of the environment or the agent's interactions with it environment [10]. In situated multiagent systems, the *environment* is the entity in which agents exist and communicate [6]. Communication is a critical factor that enables agents to interact and coordinate. Typically, this interaction and coordination is modeled using direct communication through the social environment; however, it can also be modeled indirectly through the physical environment. A *social environment* is the entity that provides the principles, processes and structures that enable the agents to communicate while the *physical environment* provides principles and processes that affect objects within an environment [6]. In [4], Ferber defines a multiagent system as having six basic entities:

- An environment, E
- A set of objects, O, that exist in E
- A set of agents, A, which are active objects (i.e., a subset of O)
- A set of relations, R, that define relationships between objects in O
- A set of operations, O, that agents can use to sense and affect objects in O
- A set of universal laws that define the reaction of the environment to agent operations Based on Ferber's definition, we have identified *five requirements* for specifying agent-environment interaction model. Essentially, an AEI should define:
  1. A unique entity called the environment
  2. The set of objects in the environment (which includes agents)
  3. Specific types of relations that may exist between objects in the environment
  4. The set of operations that agents may perform upon objects in the environment
  5. The laws that govern the effect of those operations on objects in the Environment

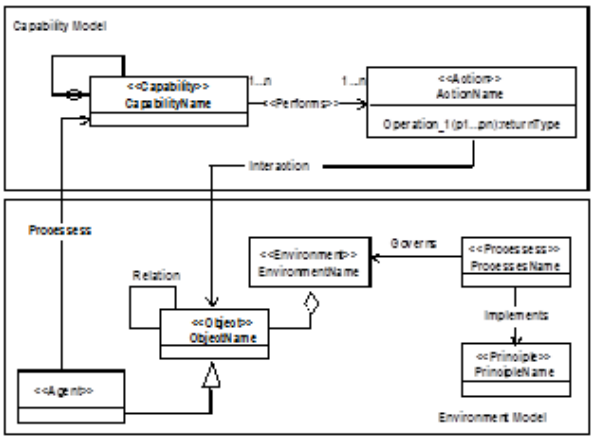

Figure 1: Agent- Environment Interaction Model.

IV. NATURE OF COMPLEX SYSTEM

Industrial-strength software is complex in nature. It is typically characterized by a large number of parts that have many interactions [18]. Moreover this complexity is not accidental. It is an innate property of the type 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 [18]. The diagram below represents a canonical view of a complex system [19, 20].

- Complexity frequently takes the form of hierarchy. That is, the system if composed of inter-related sub-systems, each of which is itself a hierarchy. The precise nature of

the organizational relationship varies between sub-systems, although some generic forms (such as client-server, peer and team) can be identified. Organizational relationship 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 is defined 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 frequently (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.

Drawing these insights together, it is possible to define a canonical view of a complex system (Figure 2). The system's hierarchical nature is expressed through the "frequent interaction" links and interactions between components are expressed through "infrequent" links. The variable notion of primitive components can be seen in the way that atomic components at one level are expanded out to entire sub-systems at subsequent levels.

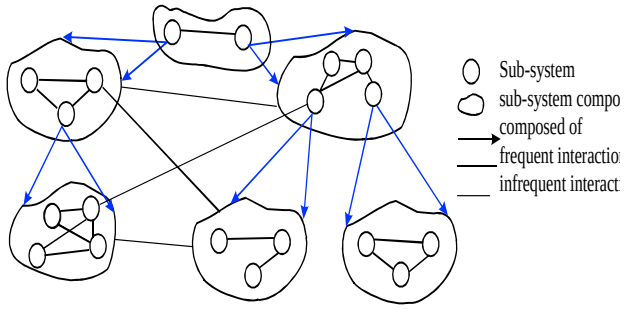

Figure 2. Canonical view of a Complex System

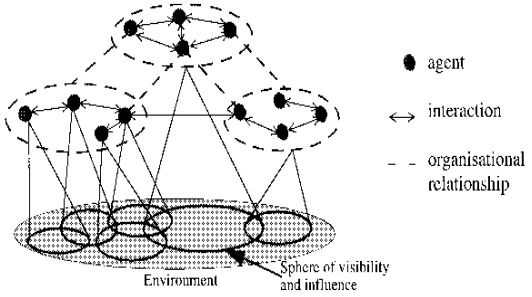

Figure 3: Canonical view of an Agent-based System

V. MOVITATION FOR SOFTWARE ENGINEERING

The core concept of agent-based computing is, of course, that of an agent. However, the definition of an agent comes along with a further set of relevant agent-specific concepts and abstractions. Generally speaking, an agent can be viewed as a software entity with the following characteristics.

- **Autonomy**: an agent is not passively subject to a global, external flow of control in its actions. That is, an agent has its own internal execution activity (whether a Java thread or some other sort of goal-driven intelligent engine, this is irrelevant in this context), and it is pro-actively oriented to the achievement of a specific task.
- **Situatedness**: an agent performs its actions while situated in a particular environment, whether a computational (e.g., a Web site) or a physical one (e.g., a manufacturing pipeline), and it is able to sense and affect (portions of) such an environment.

- **Sociality**: in the majority of cases, agents work in open operational environments hosting the execution of a multiplicity of agents, possibly belonging to different stakeholders (think, e.g., of agent-mediated market places). In these MASs, the global behaviour is derived from the interactions among the constituent agents. In fact, agents may communicate/coordinate with each other (in a dynamic way and possibly according to high-level languages and protocols) either to achieve a common objective or because this is necessary for them to achieve their own objectives. Figure 4 below depicts a multiagent system architecture and traditional software architecture [21].

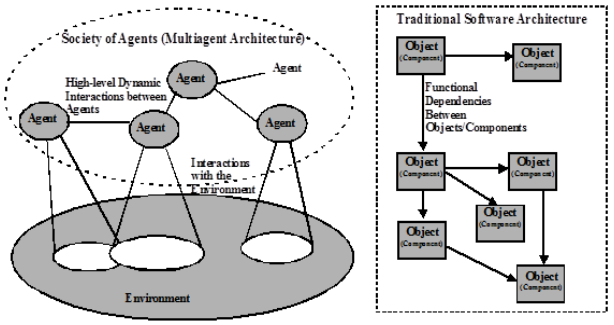

Figure 4. Multiagent Systems Architecture vs. Traditional Software Architecture.

Looking at the above definition, it is clear that MAS cannot be simply reduced to a group of interacting agents. Instead, the complete modeling of a MAS requires explicitly focusing also on the environment in which the MAS and its constituent agents are situated and on the society that a group of interacting agents give rise to.

Modeling the environment implies identifying its basic features, the resources that can be found in the environment, and the way via which agents can interact with it [22]. Modeling agent societies [23] (or agent organizations [24], or agent ecologies [25], the specific metaphor to be adopted depending on the specific characteristics of the application goals, and of the operational environment as well) implies identifying the overall rules that should drive the expected evolution of the MAS and the various roles that agents can play in such a society. All of the above considerations lead to the very general characterization depicted in Figure 4-left, whose basic abstractions and overall architecture totally differ from that of traditional software engineering approaches (Figure 4right).

When considering the traditional object-oriented perspective [26], the differences between the object-oriented and the agent-oriented perspective on modeling and building a software system are sharp. An object, unlike an agent, is in principle neither autonomous nor proactive, in that its internal activity can be solicited only by service requests coming from an external thread of control. In traditional object applications, there is not any explicit modeling of external ‘‘environment’’: everything is modeled in terms of objects, and objects either wrap environmental resources in terms of internal attributes, or perceive the world only in terms of other objects’ names/references. In addition, traditional object-based computing promotes a perspective on software systems in which components are ‘‘functional’’ or ‘‘service oriented’’ entities. A global system architecture is conceived as a static functional decomposition, where interactions between components/objects are simply an expression of inter-dependencies [27,28,29], and where concepts such as society or roles simply do not make any sense.

The above considerations make us claim that agent-based computing represents a brand new software engineering paradigm for a new discipline of agent-oriented software engineering (AOSE). However objects and components in today’s distributed and concurrent systems are somewhat removed from the historical definition and are starting to approach our view of agents. Aspect-oriented programming explicitly aims at overcoming the intrinsic limitations of functional decomposition [30]. Active objects and reactive components exhibit at least some degree of autonomy [31]. Context-dependencies in component-based applications, together with the explicit distinction between active and passive objects, approach the distinction between agents and their environment [32]. The possibility, promoted by modern middleware [33,34,35], of both establishing open interactions and modeling interactions that are more articulated than simple request-response ones, makes complex object/component based systems appear more like a dynamic society than a static software architecture. In any case, the fact that traditional object- and component-based systems are abandoning traditional abstractions and are starting to adopt others, approaching those of agent based computing, is the body of evidence that (i) a novel software engineering paradigm is needed and (ii) the agent-oriented one is the right one.

The main purpose of Agent Oriented Software Engineering is to create methodologies and tools that enable inexpensive development and maintenance of agent-based software. In addition, the software should be flexible, easy-to-use, scalable [36] and of high quality in relation to Object Oriented Software Engineering.

Agent-Oriented Programming can be seen as an extension of Object Oriented Programming (OOP), OOP on the other hand is a successor of structural programming.

VI. SOFTWARE ENGINEERING METHODOLOGY CLASSIFICATION

The software Engineering Institute (SEI) in 1988 presented a set of guidelines for assessing software development methods for real time systems[37]. The guidelines define a five-step process for evaluating different methodologies. These five steps are:-

1. **Needs Analysis** – Determine the important characteristics of the system to be developed and how individual methods help developers deal with those characteristics;

2. **Constraint Identification** – Identify the constraints imposed on the permitted solutions and determine how individual methods help developers deal with those constraints;
3. **User Requirements** – Determine the general usage characteristics of the individual methods;
4. **Management Issues** – Determine the support provided by the method to those who must manage the development process as well as the costs and benefits of adopting and using the method; and
5. **Introduction Plan** – Develop an understanding of the issues that the method does not address and a plan to augment the method in those areas where it is deficit.

The Multiagent System Engineering (MaSE) methodology, takes an initial system specification, and produces a set of formal design documents in a graphically based style. The primary focus of MaSE is to guide a designer through the software life cycle from a prose specification to an implemented agent system. MaSE is independent of a particular multiagent system architecture, agent architecture, programming language, or message-passing system. A system designed in MaSE could be implemented in several different ways from the same design. MaSE also offers the ability to track changes throughout the process. Every design object can be traced

forward or backward through the different phases of the methodology and their corresponding constructs. MaSE is described in more detail in [38, 39]. An overview of the methodology and models is shown in Figure 5.

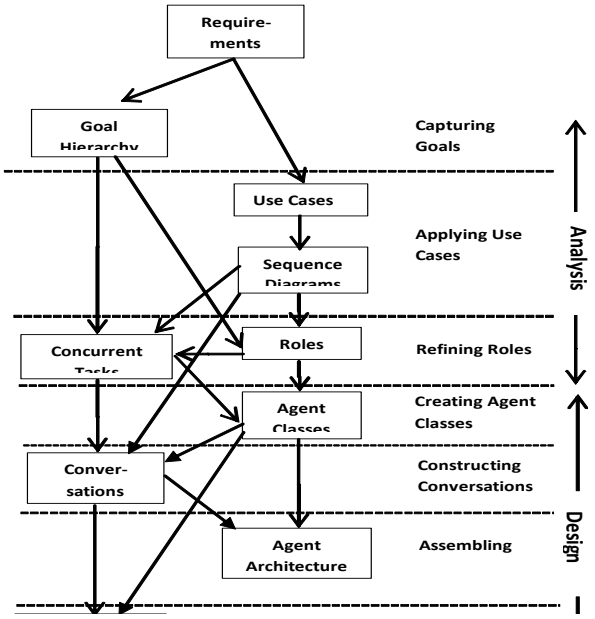

Figure 5: The MaSE Methodology

VII. DEFINING DECISION CRITERIA

The challenge of selecting an appropriate methodology for a software development project is in understanding the difference between the methodologies. The ability to classify these methodologies is crucial to the understanding. With the characteristics developed in [40] in mind, a set of criteria was developed which combined the management and usage characteristics into one category, called *Management Issues [41]*. The technical characteristics of the methodology are captured in the *Program Requirements category*. Each of these categories is discussed in detail below.

A. Management Issue

As indicated above, this category is closely related to the management and usage characteristics as defined [41]. Because of their universal applicability, many of the issues addressed that pertain to this category [17] are used as the management and usage issues for selecting a software development method for real time systems. Below is the initial set of issues selected for this category.

- Cost of Acquiring the Methodology (Meth)
- Cost of Acquiring Support Tools (Tool)
- Availability of Reusable Components (Reuse)
- Effects on Organizational Business Practices (Org)
- Compliance with Standards (Stan)
- Traceability of Changes (Chan)

The first two issues deal with costs involved with selecting the methodology. Specifically, *Cost of Acquiring the Methodology* involves the costs associated with adopting the methodology for use. Factors that influence this issue include the costs incurred by sending personnel to available training, the purchase of reference material, etc. Additionally, *Cost of Acquiring Support Tools* deals with the costs incurred by purchasing tools that support the methodology. The tools include CASE tools as well as programming development tools. Further, the cost of factors such as additional hardware/software to operate the tools, maintenance costs for the tools, and training, should be included.

Another issue that indirectly deals with cost is *Availability of Reusable Components.* The incorporation of previously developed software into a new system reduces the overall design, implementation, and testing phases for software development. This category is used to measure the methodology's ability to incorporate predefined components into the system.

The final three issues reflect usage issues. First, *Effects on Organizational Business Practices* measures the impact the adoption of a methodology will have on the existing business practices of the organization. The business practice includes ideas such as tracking development progress through milestones, reports, and customer interactions. Next, *Compliance with Standards* is proposed to determine how well an alternative is able to meet standards, whether local to the organization or outside the organization such as national or international. Finally, the last issue in this category, *Traceability of Changes*, measures the methodology's support to trace changes throughout the development lifecycle.

B. Project Requirements.

The second category of criteria, Project Requirements, is related to the technical characteristics. For this category, the criteria for real-time systems are not directly relevant. In order to derive a set of criteria, we turned to current research and identified a number of technical issues that relate to complex software systems [42,7]. The issues selected are:-

- Legacy System Integration (Leg)
- Distribution (Dis)
- Environment (Env)
- Dynamic System Structure (Struc)
- Interaction (Int)
- Scalability (Scal)
- Agility and Robustness (Agi)

The first three issues in this category relate to constraints of the problem. First, *Legacy System Integration* is a measurement of the methodology's ability to support for the incorporation of previously developed systems, commonly called legacy systems, with the new project requirement. Next, *Distribution* focuses on the ability to support the modeling of distributed aspects of the problem. Distribution can occur in the form of processors, resources, or information. Then, *Environment* measures the methodology's support of developing software systems for environments that have heterogeneous hardware or software.

The next three issues in the category are *Dynamic System Structure*, *Scalability*, *Agility and Robustness*. *Dynamic System Structure* represents the methodology's ability to develop software capable of handling the introduction and removal of system components in a manner that is not detrimental to the users of the system is considered in this category. *Scalability*, similar to *Dynamic System Structure*, measures the methodology's ability to develop software capable of handling the introduction and removal of system-level resources while minimizing the impact on users. Last, *Agility and Robustness* focuses on the methodology's ability to create flexible software systems that will be resilient to dynamic changes in the environment.

The final issue in the Project Requirements category is *Interaction.* This category determines the methodology's ability to handle the interaction between system-level components as well as entities outside the system such as human users and other systems.

C. Specification

The predominant approach to specifying agents involved treating them as intentional systems that may be understood by attributing to them mental states such as beliefs, desires, and intentions [43,44]. Following this idea, a number of approaches for formally specifying agents have been developed, which are capable of representing the following aspects of an agent-based system:

- the *beliefs* that agents have—the information they have about their environment, which may be incomplete or incorrect;
- the *goals* that agents will try to achieve;
- the *actions* that agents perform and the effects of these actions;
- the *ongoing interaction* that agents have—how agents interact with each other and their environment over time.

We call a theory that explains how these aspects of agency interact to achieve the mapping from input to output an *agent theory*. The most successful approach to (formal) agent theory appears to be the use of a *temporal modal logic [45]*. Two of the best known such logical frameworks are the Cohen-Levesque theory of intention, and the Rao-Georgeff belief-desire-intention model [45]. The Cohen-Levesque model takes as primitive just two attitudes: beliefs and goals. Other attitudes (in particular, the notion of *intention*) are built up from these. In contrast, Rao-Georgeff takes intention as primitives, in addition to beliefs and goals. The key technical problem faced by agent theorists is developing a formal model that gives a good account of the interrelationships between the various attitudes that together comprise an agent's internal state [43]. Comparatively few serious attempts have been made to specify real agent systems using such logics.

D. Implementation

Once given a specification, we must implement a system that is correct with respect to this specification. The next issue we consider is this move from abstract specification to concrete computational system. There are at least two possibilities for achieving this transformation that we consider here:

- somehow directly execute or animate the abstract specification; or
- somehow translate or compile the specification into a concrete computational form using an automatic translation technique. In the sub-sections that follow, we shall investigate each of these possibilities in turn. The figure below depicts a methodology selection value hierarchy [46].

TABLE I. SUMMARY OF DECISION CRITERIA

CATEGORY	DESCRIPTION
Management	These are the cost involve in acquiring the methodology and support tools, availability of Reusable support tools, components, effects on organizational Business Practices, Compliance with standards and traceability of change
Project Requirement	These includes the Complex system such as Legacy System Integration, Distribution, Environment, Dynamic System Structures
Specification	Approach to specifying agents is attributed to their mental states such as beliefs, desires, intensions and interaction.
Implementation	This implies testing the system to ensure it meet the desire requirement and laid out principle of user's need.

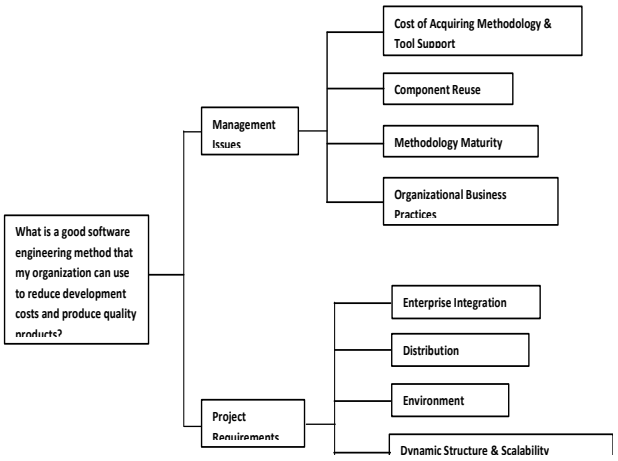

Figure 6: Methodology selection value hierarchy

VIII. MULTI-AGENT ORIENTED SOFTWARE ENGINEERING METHODOLOGY

This section describes methodologies that provide a top-down and iterative approach towards modeling and developing agent-based systems.

A. The Gaia. Methodology

Wooldridge, Jennings and Kinny [10,8] present the Gaia methodology for agent-oriented analysis and design. Gaia (Generic Architecture for Information Availability) distinguishes between analysis and design phases and associates deferent models with these two phases. It is a general methodology that also supports both the micro-level (agent structure) and macro-level (agent society and organization structure) of agent development, it is however no "silver bullet" approach since it requires that inter-agent relationships (organization) and agent abilities are static at run-time. 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 first step in the Gaia *analysis* process is to find the *roles* in the system, and the second is to model *interactions* between the roles found. Roles consist of four attributes: responsibilities, permissions, activities and protocols. *Responsibilities* are of two types: *liveness properties* - the role has to add something good to the system and *safety properties* - prevent and disallow that something bad happens to the system. *Permissions* represents what the role is allowed to do, in particular, which information it is allowed to access. *Activities* are tasks that a role performs without interacting with other roles. *Protocols* are the specific patterns of interaction, e.g. a seller role can support different auction protocols, e.g. "English auction". Gaia has formal operators and templates for representing roles and their belonging attributes; it also has schemas that can be used for the representation of interactions.

In the Gaia *design* process, the first step is to map roles into *agent types*, and then to create the right number of *agent instances* of each type. The second step is to determine the *services model* needed to fulfill a role in one or several agents, and the final step to is create the *acquaintance model* for the representation of communication between the agents.

Due to the mentioned restrictions of Gaia, it is of less value in the open and unpredictable domain of Internet applications, on the other hand it has been proven as a good approach for developing closed domain agent-systems. As a result of the domain restrictions of the Gaia method, Zambonelli, Jennings et al [49]. proposes some extensions and improvements of it with the purpose of supporting development of Internet applications. Other sources for the discussion of micro and macro aspects of agent modeling include work by Chaib-draa [50]

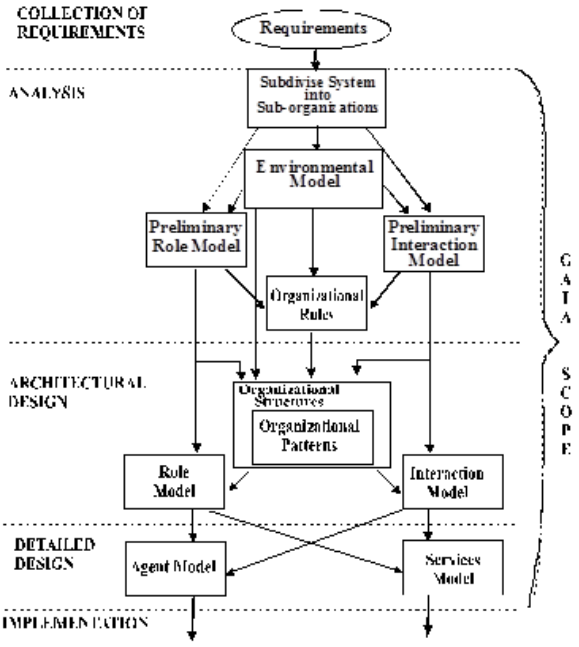

Figure 7: Developing Multiagent System: The Gaia Methodology

B. The Multiagent. Systems Engineering Methodology

Wood, M.F and DeLoach [3,31] suggest the Multiagent Systems Engineering Methodology (MaSE). MaSE is similar to Gaia with respect to generality and the application domain supported, but in addition MaSE goes

further regarding support for automatic code creation through the MaSE tool. The motivation behind MaSE is the current lack of proven methodology and industrial-strength toolkits for creating agent-based systems. The goal of MaSE is to lead the designer from the initial system specification to the implemented agent system. Domain restrictions of MaSE are similar to those of Gaia's, but in addition it requires that agent-interactions are one-to-one and not multicast.

The MaSE methodology is divided into seven sections (phases) in a logical pipeline. *Capturing goals*, the first phase, transforms the initial system specification into a structured hierarchy of system goals. This is done by first identifying goals based on the initial system specification's requirements, and then ordering the goals according to importance in a structured and topically ordered hierarchy. *Applying Use Cases*, the second phase, creates use cases and sequence diagrams based on the initial system specification. Use cases present the logical interaction paths between various roles in and the system itself. Sequence diagrams are used to determine the minimum number of messages that have to be passed between roles in the system. The third phase is *refining roles*; it creates roles that are responsible for the goals defined in phase one. In general each goal is represented by one role, but a set of related goals may map to one role. Together with the roles a set of tasks are created, the tasks defines how to solve goals related to the role. Tasks are defined as state diagrams. The fourth phase, *creating agent classes*, maps roles to agent classes in an agent class diagram. This diagram resembles object class diagrams, but the semantic of relationships is high level conversation as opposed to the object class diagrams' inheritance of structure. The fifth phase, *constructing conversations*, defines a coordination protocol in the form of state diagrams that define the conversation state for interacting agents. In the sixth phase, *assembling agent classes*, the internal functionality of agent classes are created. Selected functionality is based on five different types of agent architectures: Belief-Desire-Intention (BDI), reactive, planning, knowledge based and user-defined architecture. The final phase, *system design*, create actual agent instances based on the agent classes, the final result is presented in a deployment diagram. Visions of the future for MaSE are to provide completely automatic code generation.

IX. ISSUES IN MULTI-AGENT ORIENTED SOFTWARE ENGINEERING METHODOLOGY

Although MASs provide many potential advantages, they also present many difficult challenges. Here, I present problems inherent in the design and implementation of MASs. The list of challenges includes problems first posed in by Bond and Gasser (1998), but I have added some:

First, how do we formulate, describe, decompose and allocate problems and synthesize results among a group of intelligent agents?

Second, how do we enable agents to communicate and interact? What communication language and protocols do we use? How can heterogeneous agents interoperate? What and when can they communicate? How can we find useful agents in an open environment?

Third, how do we ensure that agents act coherently in making decisions or taking action, accommodating the nonlocal effects of local decisions and avoiding harmful interactions? How do we ensure the MAS do not become resource bounded? How do we avoid unstable system behavior?

Fourth, how do we enable individual agents to represent and reason about the actions, plans, and knowledge of other agents to coordinate with them; how do we reason about the state of their coordinated process (for example, initiation and completion)?

Fifth, how do we recognize and reconcile disparate viewpoints and conflicting intentions among a collection of agents trying to coordinate their actions?

Sixth, how do we design technology platforms and development methodologies for MASs?

Solutions to these problems are intertwined (Gasser 1991). For example, different modeling schemes of an individual agent can constrain the range of effective coordination regimes; different procedures for communication and interaction have implications for behavioural coherence. Different problem and task decompositions can yield different interactions.

X. CONCLUSION

The area of AOSE is definitely at a very early stage. However an increasing number of research groups were involved in this topic to explore the implications of developing complex software systems according to agent-oriented paradigm. The reasons for software engineering methodologies are to develop a high quality software product at the least cost and to use the methodology that best fits the problem. However, new methodologies are being developed that specifically address new technologies, such as the development of multiagent systems. AOSE provides a different way of looking at the same problem by raising the level of abstraction. The solution to this is to be able to classify different software engineering methodologies quantitatively based on the software requirement at hand.

This paper aims to give the concepts and recent progress of agent-oriented software engineering methodologies. Firstly, some of the outstanding issues in regard to software engineering are understanding the situations in which agent solutions are appropriate and the development techniques for agent systems. Significantly, Multiagent Software Engineering tends to guide system designer through the entire software development lifecycle, beginning from system representation to implementation. MaSE combines several preexisting models into a single structured methodology. Secondly,

agent-based techniques are the ideal computational model for developing software for open, networked systems (such as the Internet).

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

REFERENCES

.

[1] M. Ljunberg and A. Lucas, "The OASIS Air –Traffic Management System". In Proceedings of the Second Pacific Rim International Conference on AI (PRICAI-92. 15-18 September, 1992. Seoul, Korea.

[2] R. Weihmayer and H. Velthuijsen, "Application of Distributed AI and Cooperative Problem Solving to Telecommunications and Network Management", eds. J. Liebowitz and D. Prereau. Amsterdam, the Netherlands: IOS Press, 1994.

[3] H. Wang and C. Wang, "Intelligent Agents in the Nuclear Industry". IEEE Computer 30(11)28) 28-34. 1997.

[4]. U.M. Schwuttke and A.G. Quan. "Enhancing Performance of Cooperating Agents in Real-Time Diagnosis Systems". In Proceedings of the Thirteen International Joint Conference on Artificial Intelligence (IJCAI-93), 332-337. Menlo Park, Calif,: International Joint Conferences on Artificial Intelligence, 1993

[5] B. Huberman and S.H. Clearwater. "A Multiagent System for Controlling Building Environments". In Proceedings of the First International Conference on Multiagent Systems, 171-176. Menlo Park, Calif.: AAAI Press. 1995

[6] S. Bussmann, "Agent-oriented programming of manufacturing control tasks," in Proceedings of the 3rd International Conference on Multi-Agent Systems, IEEE Comput., CS Press: Paris (F), 1998, pp 57-63.

[7] W. Shen and D. Norrie, "Agent-based systems for intelligent manufacturing: a state of the art survey," Int. J. Knowl. Inform. Syst., vol.1, no.2, pp. 129-156, 1999. 129-156.

[8] J. Kephart, "Software agents and the route to the information economy," Proc, Natl. Acad. Sci. vol. 99, no.3, pp. 7207-7213, 202.

[9] A. Bieszczad, B. Pagurek, and T. White, "Mobile agents for network Management," IEEE Commun. Surv. Vol. 1, no.1, pp 2-9.

[2.2(100)] H.A. Simon (1996) "The sciences of the artificial" MIT Press.

[10] Iglesias, C., Garijo, M., Gonzalez., J.: A Survey of Agent-Oriented Methodologies. In: Muller, J.P., Singh, M.P., Rao, A.S., (Eds.): Intelligent Agents V. Agents V. Agents Theories, Architectures, and Languages. Lecture Notes in Computer Science, Vol. 1555. Springer-Verlag, Berlin Heidelberg (1998) 185-198.

[11] N.R. Jennings, "An agent-based approach for building complex software systems," Commun. ACM, vol. 44, no. 4, pp. 35-41

[12] N.R ,Jennings, and M.J. Wooldridge, "Agent-Oriented Software Engineering,". To appear in Bradshaw, J. (ed.): Handbook of Agent. AAI/MIT Press (2001)

[13] G. Booch, "Object-Oriented Analysis and Design with Applications". The Benjamin/Cummings Publishing Company, Redwood City, CA (1994).

[14] MESSAGE: Methodology for Engineering Systems of Software Agents-Initial Methodology: EURESCOM Participants in Project P907-GI (2000).

[15] D. Weyns, H. Parunak, F. Michel, T. Holvet and J. Ferber. "Environments for Multiagent Systems State-of-the-Art and Research Challenges." LNAI Vol. 3373, Springer (2005) 1-47

[16] J. Odell, H. Parunak, M. Fleischer, S, Bruckner. "Modeling Agents and their Environments." LNCS Vol. 2585, Springer (2002) 16-31

[17] J. Ferber, "Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence." Addison-Wesley, Harlow (1999)

[18] H.A. Simon(1996) "The sciences of the artificial" MIT Press.

[19] N.R. Jennings and M. Wooldridge, "Agent-Oriented Software Engineering." Department of Electronic Engineering Queen Mary & Westfield College University of London, London E1 4NS, United Kingdom. 1999

[20] N.R. Jennings, "On agent-based software engineering", Department of Electronics and Computer Science, University of Southampton, Southampton SO17 IBJ, UK . 1999.

[21] F. Zambonelli and A. Omicini, "Challenges and Research Directions in Agent-Oriented Sofware Engineering." Autonomous Agents and Multi-Agent Systems, 9, 253-283, 2004 @ 2004 Kluwer Academic Publishers. Manufactured in the Thailand.

[22] A. Omicini, "SODA: Societies and infrastructure in the analysis and design of agent-based systems," in P. Ciancarini and M.J. Wooldridge(eds.), Agent-Oriented Software Engineering, vol. 1957 of LNCS, Springer-Verlag. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers, 2001, pp. 185-193.

[23] Y. Moses and M. Tennenholtz, "Artificial social systems," Comput.
Artif. Intell. Vol. 14, no.3, pp. 533-562, 1995.

[24] M.S. Fox, "An organizational view of distributed systems," IEEE Trans.
Syst. Man Cyber. Vol. 11, no 1, pp. 70-80, 1981.

[25] H.V.D. Parunak, "Go to the ant: Engineering principles from natural agent systems," Ann. Oper. Res., vol. 75, pp. 69-101, 1997.

[26] G. Booch, Object Oriented Analysis and Design, (2nd edn.), Reading (MA): Addison-Wesley, 1994.

[27] L.Bass, P. Clements, and R. Kazman, "Software Architectures in Practice.", (2nd edn.). Addision Wesley: Reading (MA), 2003.

[28] M.Shaw, R. DeLine, D. Klein, T. Ross, D. Young, and G. Zelesnik, "Abstractions for software architecture and tools to support them," IEEE Transactions on Software Engineering, vol.21, no.4, pp 314-335, 1995.

[29] M. Shaw and D. Garlan, Software Architectures: Perspectives on an Emerging Discipline, Prentice Hall, Englewood Cliffs (NJ), 1996.

[30] G. Kiezales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin, "Aspect oriented programming," in Object-Oriented Programming, vol.1241 of LNCS, Springer-Verlag, 1997, pp. 220-242.

[31] P. Eugster, P.A. Felber, , R. Guerraoui and A. Kermarrec, "The many faces of publish/subscribe," ACM Comput. Surv. Vol. 35, no.2, pp. 114-131, 2003.

[32] G. Cabri, L. Leonardi, and F. Zambonelli; "Engineering mobile agent applications via context-dependent coordination," IEEE Trans. Software Eng. Vol. 28, no. 11, pp. 1034-1051, 2002.

[33] P. Ciancarini, A. Omicini, and F. Zambonelli, "Coordination technologies for Internet agents," Nordic J. Comput. Vol. 6¸ no.3, pp. 215-240.

[34] P. Eugster, P.A. Felber, , R. Guerraoui and A. Kermarrec, "The many faces of publish/subscribe," ACM Comput. Surv. Vol. 35, no.2, pp. 114-131, 2003.

[35] OMG, "COBRA 2.1 Specifications," 1997, www.corba.org.

[36] B. Wood and R. Pethia, L.R. Gold and R. Firth, "A Guide to the Assessment of Software Development Methods." Software Engineering Institute Technical Report 88-TR-8. Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA (1988).

[37] S.A. DeLoach and M.F. Wood, "Multiagent Systems Engineering: the Analysis Phase." Technical Report, Air Force Institute of Technology, AFIT/EN-TR-00-02, June 2000.

[38] M.F. Wood, "Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems." MS thesis, AFIT/GCS/ENG/00M-26. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFT Ohio, USA (2000).

[39] R. Firth, et al.: A Classification Scheme for Software Development Methods. Software Engineering Institute Technical Report 87-TR-41. Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA (1987)

[40] O'Malley, S.A.: Selecting a Software Engineering Methodology using Multiobjective Decision Analysis, AFIT/GCS/ENG/01M-08. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH (2001).

[41] S.A. O'Malley, "Selecting a Software Engineering Methodology using Multiobjective Decision Analysis," AFIT/GCS/ENG/01M-08. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH (2001).

[42] Lesser, V.R.: Cooperative Multiagent Systems: A Personal View of the State of the Art. IEEE Trans. on Knowledge and Data Engineering. 11(1) (1999) 133-142.

[43] D.C. Dennett (1987) "The International Stance" The MIT Press.

[44] M. Wooldridge and N.R. Jennings (1995) "Intelligent agents: theory and practice" The Knowledge Engineering Review 10(2) 115-152.

[45] A.S. Rao and M. Georgeff (1995) "BDI Agents: from theory to practice" Proc. of the 1st Int. Conf. on Multi-Agent Systems, 312-319, San Francisco, CA.

[46] A. Scott, O'Malley and Scott, A. Deloach (2001) "Proeedings of the Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), Montreal, Canada, May 29th 2001.

[47] M.J. Wooldridge, N.R. Jenning and D. Kinny, "The Gaia methodology for agent-oriented analysis and design." Autonomous Agents and Multi-Agent Systems, 1(1): 7-38, 1998.

[48] M.J. Wooldridge, N.R. Jenning and D. Kinny, Amethodology for agent-oriented analysis and design. In Proc. of the third international conference on Autonomous agents, pages 69-76, 1999.

[49] M. Young, The Technical Writer's Handbook. Mill Valley, CA: Science, 1989.

[50] B. Chaib-draa, "Connection between micro and macro aspects of agent modeling". In proc. of the First International Conference on Autonomous agents, pages 262-267,1997.

[51] A.H. Bond and , L. Gasser, "Readings in Distributed Artificial Intelligence." San Francisco, Calif.: Morgan Kaufmann. 1988.

[52] L.Gasser, "Social Conceptions of Knowledge and Action. Artificial Intelligence." 47(1-3):107-138. 1991