Submitted to: The International Journal of Multiagent and Grid Systems, Nov. 2005.

A Methodology to Evaluate Agent Oriented Software Engineering Techniques

Chia-En Lin, Krishna M. Kavi
Department of Computer Science and Engineering
University of North Texas
Denton, TX 76203
cl0065@cse.unt.edu, kavi@cse.unt.edu

Frederick T. Sheldon and Thomas E. Potok
Computational Sciences and Engineering
Oak Ridge National Laboratory[1]
Oak Ridge, TN 37831 USA
SheldonFT@ornl.gov | PotokTE@ornl.gov

Abstract[1]

Systems using Software Agents (or Multi-Agent Systems, MAS) are becoming more popular within the development mainstream because, as the name suggests, an Agent aims to handle tasks autonomously with intelligence. To benefit from autonomous control and reduced running costs, system functions are performed automatically. Agent-oriented considerations are being steadily accepted into the various software design paradigms. Agents may work alone, but most commonly, they cooperate toward achieving some application goal(s). MAS's are components in systems that are viewed as many individuals living in a society working together. Currently however, there is no universal agreement on how to build a comprehensive Agent-oriented system. Development of MAS's is a non-trivial task especially without the necessary support provided by software engineering environments. From a software engineering perspective, solving a problem should encompass the steps from problem realization, requirements analysis, architecture design and implementation. These steps should be implemented within a life-cycle process including testing, verification, and reengineering to proving the built system is sound. Agent-oriented software engineering techniques must be evaluated and compared to gain a better understanding of how Agent systems should be engineered and evolved.

In this paper, we explore the various applications of Agent-based systems categorized into different application domains. We describe what properties are necessary to form an Agent society with the express purpose of achieving system-wide goals in MAS. A baseline is developed herein to help us focus on the core of Agent concepts throughout the comparative study and to investigate both the Object-Oriented and Agent-oriented techniques that are available for constructing Agent-based systems. In each respect, we address the conceptual background associated with these methodologies and how available tools can be applied to provide to specific domains.

[1] This manuscript has been authored by UT-Battelle, a contractor of the U.S. Government (USG) under Department of Energy (DOE) Contract DE-AC05-00OR22725. The USG retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

1 Introduction

Within the last decade, Information Technology (IT) has played an ever-increasing role in the everyday aspect of life. The power of IT has increasingly influenced software development in every field of application from personal computing, to critical infrastructures and industrial systems. Developing a software system involves the challenge of coping with embedded computational complexities of distribution, multi-tasking, and real-time. Moreover, it is nearly impossible for users to manually command/control a software systems that is even mildly complex. Clearly, the need for sophisticated, automatic intelligence must be brought into the life cycle and development environment. Accordingly, agent-oriented computing, which provides such intelligence, has become an important research topic. An "Agent", is a programmed system that can handle tasks autonomously with intelligence under some prescribed rules. Systems using Agents are becoming more popular in software development.

How can existing methodologies be extended to ensure sophisticated and autonomous intelligence be ordained from the first step forward? Agent methodologies can inherit properties of simple functional systems, reactive systems and distributed concurrent systems. Jennings and Wooldridge (1998) suggest three classes of systems utilizing the Agent-oriented computing model: Open Systems, Complex Systems, and Ubiquitous Computing Systems. An Open System is comprised of components and structures that change dynamically. For example, the Internet is a highly open software environment with its size and complexity increasing exponentially. Software systems live on it; hence, mechanisms to adapt software are needed. Complex Systems require integration of large number of components without *a priori* knowledge of the interactions or results of the interactions. Usually, modularity and abstraction are used to cope with this complexity. Agents can be used to monitor the interactions among components of complex and dynamic systems, particularly when human-interactions in such systems is either unavailable or becomes a bottleneck. The need for autonomous control systems, reducing cost of running such systems and dynamic nature of complex systems are driving the adaptation of Agent-oriented considerations into software designs.

Agents can be delegated to perform simple tasks or to provide autonomous services to assist a user without being "commanded." Agents play roles in terms of different expertise and functionalities while supporting services to an application. They are responsible for making intelligent decisions to execute such activities and to provide valid and useful results. Agents can work alone, but most importantly, they can cooperate with other agents. Agents are software components in the system that are viewed as many individuals living in a society working together. Examples of Agents include personal assistance devices, application assistants, tutoring agents, agents controlling smart home management and agents involved in critical infrastructure protection.

1.1 Agent Background

An Agent is an entity that is not only designed to run routine tasks commanded by its users, but also to achieve a proposed setting (or goal) within the context of a specific environment. The difference between an Agent and a traditional software entity is that the latter just follows its designed functions, procedures or macros to run deterministic codes. The former incorporates the ability to practice intelligence by making (autonomous/semi-autonomous) decisions based on dynamic runtime situations.

There is no universally accepted definition of an Agent. According to N. R. Jennings (2000), autonomy is the central notation of an Agent. More generally, an Agent is an encapsulated system, which is situated in some environment and is capable of flexible, autonomous action in the environment to meet its designed objectives.

There are several application domains of software system functions that are appropriate for Agent techniques. Jennings and Wooldridge (1998) proposed a catalogue to classify Agent-based application domains as shown in Table 1.

Table 1: Agent Applications Catalogs

Application Domain	Application Fields
Industrial	Critical Process Control, Manufacturing, Air Traffic Control, Network and Telecommunication Management, Transportation System
Commercial	Information Management, Electronic Commerce, Business Process Management
Medical	Patient Monitoring, Health Care
Entertainment	Games, Interactive Theater and Cinema

Building industrial-strength applications in a robust, fault-tolerant and flexible way seems a demanding requirement. Current Object-Oriented software engineering processes provide methods and tools for developing traditional software systems. Standards exist for modeling, analyzing, designing and testing Object-Oriented software. However, there is no consensus agreement on how to build a comprehensive Agent-oriented system. Because of the intrinsic properties of Agents, development of Agent systems is a non-trivial task without the necessary infrastructure. Agent-Oriented Software Engineering requires methodologies and tools to encompass steps from problem realization, requirements analysis, architecture design and implementation. These steps must be implemented within a life-cycle process including testing, verification, and reengineering to prove the built system is sound.

1.2 Related Research

Agent-Oriented Software Engineering (AOSE) is a nascent but active field of research (Tveit, 2001). A comprehensive methodology that plays an essential role in software engineering must be robust but easy-to-use. More importantly, it should provide a roadmap to guide engineers in creating Agent-based system. Recently several Agent-oriented methodologies have been proposed to address the Agent oriented software engineering process. Thus far, however, software developers have not embraced any single methodology, primarily because AOSE lacks industrial strength tools and standards (Sturm & Shehory, 2003). Understanding the limitation of existing AOSE methodologies can permit researchers to develop better solutions. Exploration into building blocks for Agents, fundamental theories and methods as well as available assistance from notations of analysis, architecture design and implementation toolkits should is needed. The ultimate hope is the development of practical AOSE methodologies for building robust, industrial-strength Multi-Agent Systems (MAS). However, given the divergent directions currently being pursued by researchers, the road towards *mature* Agent software development may be reached most effectively by first gaining a better understanding of the competing approaches.

A few frameworks for comparing Agent-oriented methodologies have been suggested. Sabas, Badri, and Delisle (2002) suggest a multi-dimensional framework containing criteria within each of the following aspects: methodology, representation, organization, cooperation, and technology. These criteria are used as differentiators for comparison purposes. The results of comparisons are described by a two-dimensional array containing criteria (row wise) and methodological names (column wise). Each intersection is marked: "Y" for Yes, "N" for No, "P" for possible, or simply blank (' ')[2]. In the work by Strurm and Shehory (2003), four dimensions are proposed as four separate divisions of the issues being examined. They are (1) concepts and properties, (2) notations and modeling, (3) process, and (4) pragmatics. In addition to these, support for software engineering and marketability are added into the comparison framework by Dam and Winikoff (2003).

A different approach using goal-question-metric (GQM) is proposed by Gernuzzi and Rossi (2002) to determine what factors are important to measure for comparing methodologies. An attribute tree is created according to the objectives of advanced GQM questions to identify comparison criteria with each tree root representing a specific attribute. Traditional software engineering and Agent-based system characteristics are proposed by Shehory and Strurm (2001) as an evaluation/comparison approach methodology. In each field, corresponding criteria are inspected. Textual statements provide comments on each issue. The final evaluation is graded incrementally as good (+), satisfying (*), or not supported (NS). Consequently, in this paper we build on these previous attempts and develop a framework for comparing different Agent-Oriented Software Engineering methodologies. We begin by first introducing several AOSE techniques and frameworks.

[2] "Y" (Yes) represents the methodology that takes criteria into account. "N" (No) represents that it does not take criteria into account. "P" (Possible) means the methodology could take criteria into account based on available information. Finally, the blank (' ') means that it is unable to make conclusion based on available information.

2 AOSE Techniques and Tools

Often, software engineers use Object-Oriented programming to implement Agent systems. There are similarities between an Object-Oriented (OO) entity and an Agent-oriented (AO) entity. Both are distributed instances that can communicate with peer entities and have state and member functions to support their behavior.

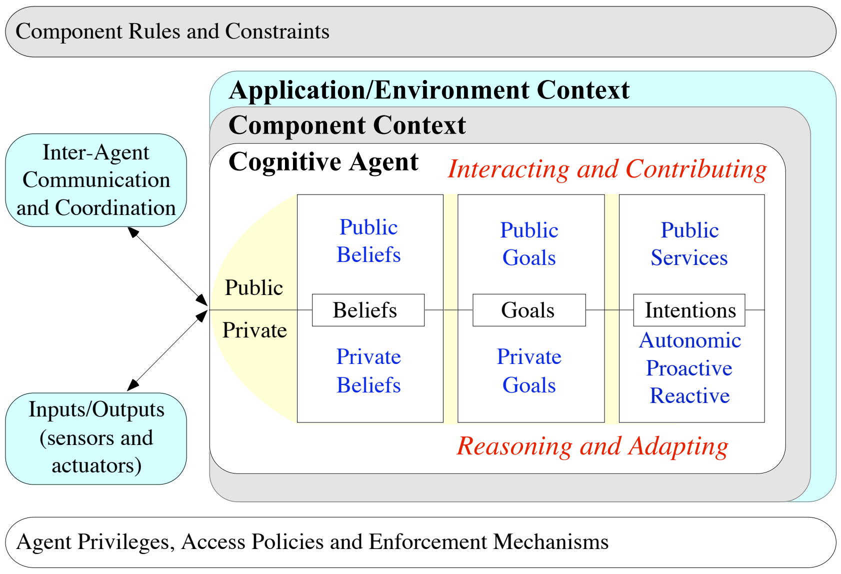
Component Rules and Constraints
Application/Environment Context
Component Context
Cognitive Agent
Interacting and Contributing
Inter-Agent Communication and Coordination
Public Beliefs
Public Goals
Public Services
Public
Private
Beliefs
Goals
Intentions
Private Beliefs
Private Goals
Autonomic Proactive Reactive
Inputs/Outputs (sensors and actuators)
Reasoning and Adapting
Agent Privileges, Access Policies and Enforcement Mechanisms

Figure 1. Example of how the BDI model is a useful Agent paradigm for endowing intelligent adaptive characteristics

Using well-tested Object-Oriented Software Engineering (OOSE) methods can make the Agent software process more adaptable to a majority of systems. However, some differences exist between Agents and Objects. For example, Agents are autonomous, self-contained and should act to achieve goals without external influence or initiation (i.e., an Agent decides when and how to execute its functions). Objects, however, are passive entities because their behavior depends on their member functions being triggered. In addition, Objects lack an explicit mental state concept; they do not have mechanisms to act proactively toward goals. On the other hand, objects bring maturity to the design. Although lacking major support features for the Agent paradigm, software engineers depend on their experiences and intuition while using pure Object-Oriented paradigms for modeling and implementing Agents. In Object-Oriented modeling, design patterns have been mostly used to increase flexibility, and reusability of software entities. Practitioners can apply patterns to fit object components into applications thus gaining a clearer view of design that are justified through comprehensive detailed scenarios within a given pattern. Patterns can provide a template for addressing some aspects of Agents. In short, an Agent not only preserves Object properties but also extends them with specific capabilities. These features must be considered while contemplating Object-Oriented patterns for Agents.

2.1 BDI Based Methods

A crucial capability for autonomous Agents is reasoning. Modeling how Agents "think" can help designers observe micro-activities in Agents. A well-known method to describe rational Agents is the BDI model, proposed by A. Rao and M. Georgeff (1991). The motivation of BDI (Belief, Desire and Intention) is the recognition that dynamic factors from the system and the environment should be considered when modeling the behavior of an

Agent. BDI describes an Agent's beliefs about the system and the environment, the Agent's goals (or desires) to achieve as well as, expressing the Agent's intentions by way of executable plans. Agents can reason about what is the best plan for achieving desires under specific beliefs about the environment. An Agent can review its goals and respond with revised plans, if necessary, as system or environmental parameters change. Figure 1 illustrates these concepts which convey that intelligent (or cognitive) adaptive systems may comprise three types of processes: 1) *reactive*, for producing timely responses to external stimuli, 2) *deliberative*, for possessing learning and reasoning abilities, and 3) *reflective*, for the ability to continuously monitor and adapt based on introspection. Although useful, the BDI model has limitations for use as an Agent-Oriented Software Engineering methodology, particularly for the design of multi-agent systems.

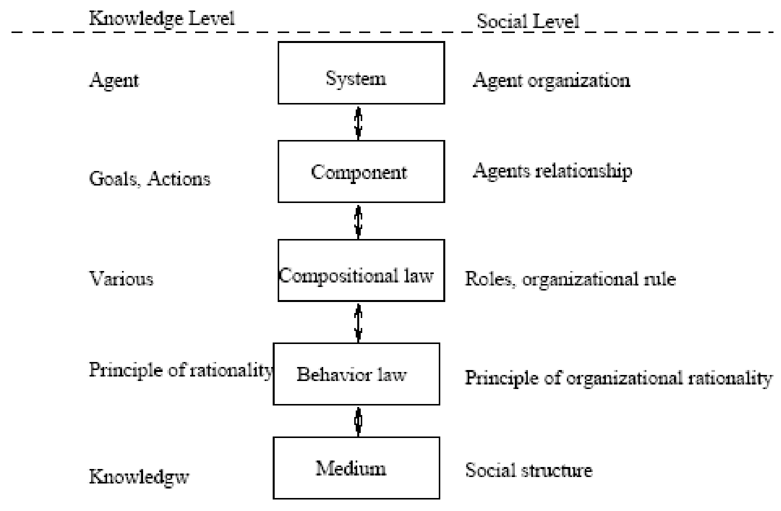

Figure 2. Social level model

2.2 Role/Society Based Model

A software process consists of a set of steps (or stages) used to assure the successful development of systems. Some easily usable models, techniques and tools have to be offered to developers to handle complex problems within these steps. In the context of Agent systems, the software processes must deal with traditional issues as well as the added properties of Agents. For Object-Oriented approach, Booch suggested decomposition, abstraction, and organization are principles to handle software complexity. N. R. Jennings (2000) states that Agent approaches can also use these principles. However, these are not adequate for Agents' interactions or BDI (mental) abilities. Jennings suggests the following approach for Agents, inspired by social interactions among humans.

There are two ways to describe this model. The first is the social-level viewpoint and the second is knowledge-level viewpoint. Figure 2 shows the social level model in a summary diagram, which describes how a system is modeled as an organization or society made up of components, the majority of which are agents. Their communication channels include content and mechanisms, dependencies between Agents, and organizational relationships such as the concepts of peers and competitors. In the society, compositional laws are used as guidelines that describe how components in the system are organized under the regulation of the society.

Behavioral laws regulate how components (i.e., members in the organization) meet both their roles and societal commitments. From the social level viewpoint, units of the system are different organizations in the society. Different organizational mechanisms and structures can influence the behavior of the constituent components. The way organizational structures change can also significantly affect role relationships, especially by adding/removing roles. The Medium describes how to accomplish these changes, and from the knowledge level side, Agents are central to a system. An Agent perceives its goals and accomplishes them by actions. These goals and actions are governed by rational rules, which are provided as laws. All laws are based on the knowledge of their environment. Utilizing this knowledge, Agents continue working toward their goals.

The social level conceptual model stands on top of the knowledge level and provides social concepts as a foundation for Agent-oriented systems. In doing so, the social level facilitates abstraction (i.e., high-level system components are conceptualized without getting into the details of greater complexity). Moreover, laws in the society help to regulate an Agent's behavior and make the results of the system more predictable.

N.R. Jennings (2000) also proposed a methodology that aids in conceptual modeling, analysis and design. In analysis, the main task is to identify the overall goals of computing organizations, basic skills, interactions between organizations, and rules of behavior these organizations must follow. This analysis leads to the identification of roles, protocols of interaction and rules of conduct. The design phase consists of readjustment of roles, organizational patterns, and definition of organizational structure. The central metaphor generated from organizational concepts can be roles in the society. Thus, role modeling becomes a subsystem in the architecture. Goals are implicit behind roles, and Agents are important elements during analysis. In social concepts, goals are mostly mentioned as organizational aspects. Goal structure analysis can help refine them in the organizational structure. Most agent-oriented systems can be designed using this methodology if goals can be properly modeled with roles.

2.3 UML Extensions

The Unified Modeling Language (UML) is widely used in the Object-Oriented paradigm. Since UML is inherited from the Object-Oriented methodology, UML is limited in its application to multi-Agent systems. However, UML has been a successful modeling language in Object-Oriented design. It would be beneficial if Agent oriented design can use UML with extensions to model Agents. There have been several proposed extensions to UML. Some extensions propose to build models of different Agent properties as an infrastructure. Others include new UML diagrams to address specific Agent features. Currently, both FIPA (Foundation of Intelligent Physical Agents) and OMG (Object Management Group) are securing proposals to extend UML to accommodate Agent features.

One example of a framework and UML extension was proposed by a research team lead by Kavi and Kung (Kavi et al., 2003). Rooted in BDI, this UML extension provides for integrated modeling of Agents making objects Agent-capable. Mental states are central to Agent modeling. The BDI architecture has been adapted broadly as an Agent behavioral reasoning mechanism in constructing multi-Agent systems; however, it seems that engineers often use the model at a conceptual level. Implicit modeling of the BDI philosophy often generates unclear semantics in analysis and design. The framework by Kavi and Kung extends UML to model specific Agent scenarios. New notations are used to model Agent mental states, such as beliefs, goals and plans. Using these notations, a model for BDI architecture has been introduced as the Agent Domain Model. In the architecture, beliefs represent the state of a changing environment where Agents are situated. Beliefs are updated when events occur changing the current state of the system. Belief changes affect goals, which are evaluated to reflect the update. Finally, updated plans according to goals are delegated to threads responsible for execution.

The following is a summary of the framework.

- In the framework, goals and Agent classes are keys to modeling the whole system. They are represented using the Agent Domain Model to express BDI architecture in detail.
- Relationships among Agent activities and goals are modeled throughout the whole process. Goals are also represented clearly in the newly introduced diagrams, such as the Agent Goal Diagram and the Agent Sequence Diagram. It promotes goal preservation in the complete development process.
- The blackboard communication mechanism provides effective interaction among Agents. It also facilitates clear expressiveness in the Agent communication diagram.
- The modeling provides great flexibility in dealing with dynamic environments mainly due to adapting the BDI architecture and providing a clear way to depict mental schemes in UML.

2.4 Examples

There are an increasing number of AOSE methodologies that try to encompass the software issues and compete in being the main approach. Here we will describe the more popular AOSE technologies.

2.4.1 Tropos

Tropos (Bresciani, Giorgini, Hiunchiglia, Mylopoulos, & Perini, 2004) was originally developed at the University of Toronto, Canada, and is being updated and maintained by a number of universities in Europe. Tropos adopts Yu's i* framework (Castro, Kolp, & Mylopoulos, 2002) as the base theory of requirement analysis. The i* offers concepts such as actors, goals, and dependencies intended to model social structures and describe detailed relationships between them. Tropos provides a method for engineers to design multi-Agent systems that can take advantage of the societal model throughout the design process (Bresciani et al., 2004).

There are four phases in the Tropos design process. The early requirements analysis phase is the first step for identifying basic stakeholders. During the late requirement analysis phase, a potential system actor is introduced. The purpose of this "system" actor is to provide system operational services to actors depending on services from the last analysis phase. In the detailed design phase, more explicit scenarios of Agents are depicted. Finally, in the implementation phase, Tropos adapts JACK for its execution because they both rely on the BDI architecture. Notations used in Tropos can be seen as mental ones, such as goals and tasks (plans) (Perini, Bresciani, Giunchiglia, Giorgini, & Mylopoulos, 2001). The notations used throughout the analysis and design phase help preserve the semantic mapping.

Critique of Tropos Methodology:

1. The exploration of stakeholders and the dependencies between them mainly rely on the developers' experience. The realization of goals and their dependencies among stakeholders can be derived based on different points of view, which could lead to different results. The discretion is left wide open.
2. During the analysis phase, when a new sub-goal is generated from goal refinement, dependencies between the new goal and every actor in the system have to be recalculated. The iterated algorithm to run this process becomes a non-deterministic concurrent algorithm (Bresciani et al., 2004). It is non-trivial for engineers to conduct this analysis in an efficient way.
3. Dependency analysis plays an important role in the analysis phase. Statements of goals and dependencies are prone to be unclear depending on an engineer's interpretation. There are no standard guidelines to follow while decomposing goals or tasks into sub-goals or sub-tasks, and depends on the interpretations of the software engineer.
4. Although traceability is available through notation diagrams, it is still difficult to trace all the dependencies backward. A system actor is added in the middle of the analysis. Dependencies among actors are rearranged to accommodate the new actor. There is no formal rationale to support this approach.
5. Tropos rests on the uniform use of small sets of intentional notations (Perini & Susi, 2003) throughout the whole development process; however, it is difficult to reflect on a changing environment to adaptively revise beliefs and plans.

2.4.2 Gaia

Gaia, proposed originally by M. Wooldridge et al. (2000), where the foundation of analysis is based on a Object-

Oriented design method called Fusion, from which it borrows terminology and notations. Gaia is rooted in conceptual organizational modeling (Zambonelli, Jennings, & Wooldridge, 2000) and suggests that developers think about building Agent-based systems as a process of organizational design. The Agent computational organization is viewed similarly to human organization consisting of interacting roles and functions.

Under the organizational metaphor, role is the key template to be modeled. Agents play designated roles and are aware of resources modeled by environmental variables. Roles and resources are regulated by organizational rules. The developing process consists of analysis, architecture design and detailed design. Preliminary models are abstracted from requirements, which help to postulate implicit goals about organizational divisions, environment, roles and interaction rules. Explicit decisions about the desired structure are made at the architecture design stage to finalize role modeling. The detailed design stage takes roles and interactions to develop Agent classes and services.

Critique of Gaia Methodology:

1. Goals implicitly coincide with subdivisions of the system, which potentially increase the modeling complexity. There is also no clear guideline on how to derive roles from the organizational model.
2. It is difficult to model Agents entering and exiting sub-organizations; or, adapting to the evolution of organizational structure. There is a lack of dynamic reasoning (Juan, Pearce, & Sterling, 2002)
3. Organizational metaphor is a strongly embedded abstraction coded in the Gaia methodology.

2.4.3 MaSE

MaSE, proposed by Deloach et al. (2001), stands for Multi-agent System Engineering. MaSE methodology aims to provide developers guidance from requirements to implementation. The development process consists of two main phases: analysis and design. In each phase, a series of steps are provided to model the system. In each step, related models are created. Models in one step produce outputs that become inputs to the next step, which supports traceability of the models across all of the steps.

The analysis phase consists of three steps: capturing goals, applying use cases, and refining roles. 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. The second step generates use-cases and their corresponding sequence diagram. 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.

There are four steps in the design phase: creating Agent classes, constructing conversations, assembling Agent classes, and system design. The first step is a process which creates Agent classes and their interactive behavior. After each Agent class is recognized, constructing a conversation is the next step. In this step, designers construct conversation models used by Agent classes. The assembling Agent class step creates Agent class internals. The final step of design is system design, where the Agent classes are instantiated into actual Agents.

Critique of MaSE Methodology:

1. Goal analysis, conducted at the beginning of a MaSE process, reinforces goal preservation through analysis and design phases. It facilitates role and Agent class modeling to focus on clear goal delegation, where every role is responsible for a particular goal to be accomplished. There are tasks that belong to the dedicated goals of roles.
2. In a role refinement step, it is crucial to match goals with roles. Every goal has to be associated with a role. With these roles defined, the design of communication between roles and their corresponding tasks become fixed, lacking dynamic adaptability of goals (and hence roles).

3 Evaluation Methodology Defined

Often, it is unclear which methodology would be the most effective for the design of a Multi-Agent system. In this paper, we have defined a process for evaluating the methodologies, comparing strengths, weaknesses and identifying ways to improve on a particular methodological improvement. AOSE methodologies should be compared for their software engineering and agent-oriented potential/capabilities. Our evaluation includes criteria for both software processes and agent-oriented properties. Four major divisions similar to (Sturm & Shehory, 2003) are adopted in the comparison framework. We form the framework with an "aspect overview" and "detailed rationale" to provide a comprehensive evaluation. A summarized checklist is provided at the overview level based on criteria suggested in each of four divisions (i.e., Concepts and Properties, Notations and Modeling Techniques, Process, and Pragmatics). The results are compiled in Table 3, with criteria as rows and methodologies as columns. Each cell of the matrix contains an "Yes" representing that the criteria is supported by the AOSE methodology, or otherwise "No." Textual descriptions are provided as appropriate. Logical inferences of concerns are evaluated at the detailed level by providing questions and answers. These questions are derived both from emphases on displaying logical relationships within methodological issues as well as from experiences obtained from case studies. As we address these questions and gather data, we have gained a deeper insight into the comparisons and better understand the rationale of each methodology.

3.1 Criteria A: Concepts and Properties

Concepts and properties collect all the basic building blocks of Agents. Primitive capabilities or characteristics of

Agents are covered in this division. This category deals with questions on whether or not a methodology adheres to the basic notions of Agents.

Table 2: Concepts and Properties Comparison

Criteria	Description
Autonomy	An Agent can make decisions on its own based on inner states without external supervision.
Mental Mechanism	An Agent has mechanisms to realize its intentions by achieving goals.
Adaptation	An Agent is flexible enough to adjust its activities according to dynamically changing environments.
Concurrency	An Agent may need to perform multiple tasks concurrently.
Communication	There are protocols or mechanisms defined for Agent interactions.
Collaboration	An Agent has methods to cooperate with other Agents to achieve goals.
Agent Abstraction	Methodology has theory to describe Agents using high level abstractions.
Agent-oriented	The design of methodologies originates from the consideration of Agent-oriented approaches[3].

Table 3 summarizes our comparison of the methodologies based on the *Concepts and Properties* criteria identified in Table 2.

Table 3: Comparison using Concepts and Properties

	Tropos	Gaia	MaSE
Concepts and Properties (A)			
Autonomy	Yes	Yes	Yes
Mental mechanism	Goal, softgoal, task	No	Goal, task
Adaptation	Yes	Yes	No
Concurrency	Yes	Yes	Yes
Communication	Yes	No details	No Details
Collaboration	Yes	Yes	Yes
Agent Abstraction	Social actors	Roles in organization	Roles
Agent-oriented	Yes	Yes	Yes

3.1.1 Detailed Level Questions

1. What concepts are at the root of the methodology and what are the advantages?
2. How is an Agent created in a methodology?
3. How well constructed is the design that deals with Agent mental mechanisms?
4. How well does a design deal with an Agent's perception of its environment, and how does it react based on the perception?

[3] The consideration is primarily focused on whether or not the methodology addresses Agent-based features during the analysis and design.

5. How efficient are Agents in achieving their goals?

A detailed analysis and comparison of methodologies based on these *Concepts and Properties* questions above is provided in the Appendix (also see Lin, 2003).

3.2 Criteria B: Notations and Modeling Techniques

Notations and modeling techniques are key to representing elements and activities in a system. During the software development process, consistent expressive constructs help to clearly address an Agent's behavior. Good modeling can ease the complexities of understanding and implementing systems from concepts to realizations. This criteria deals with notations and models that are manipulated in a methodology.

Table 4: Notations and modeling techniques Comparison

Criteria	Description
Expressiveness	Notations are used in this methodology to help the design processes.
Complexity	There are abstract levels from low to high that help manage a complex problem with modeling.
Modularity	Uses components or modules in the methodology to model in an incremental fashion.
Executable	Models used in this methodology are capable of generating or simulating prototypes.
Refinement	A modeling technique permits refinement of goals into subgoals or roles into sub-roles.
Traceability	Traceability across the refinement boundaries is provided.

Table 5 provides a comparison of the methodologies for the *Notations and Modeling Techniques* criteria identified in Table 4.

Table 5. Comparison using Notations and Modeling Techniques

	Tropos	Gaia	MaSE
Notations and Modeling (B)			
Expressiveness	Yes	Yes	Yes
Complexity	Decomposition of goals, tasks	Role	Goal, role refinement
Modularity	Yes	Yes	No
Executable	No	No	No
Refinement	Yes	No	Yes
Traceability	Yes	Yes	Yes

3.2.1 Detailed Level Questions

1. How well are notations and models formed to address Agent-based system scenarios?

2. How consistent and unambiguous are models while running the process?
3. How well does the modeling technique address traceability and reuse?
4. How well does the modeling technique represent Agents?

A detailed analysis and comparison of methodologies based on these *Notations and Modeling Techniques* questions is provided in the Appendix (also see Lin, 2003).

3.3 Criteria C: Process

A process is a series of steps that guide practitioners to construct a software system from the beginning to the end. It serves as a detailed guideline of all activities throughout subsequent phases. This criteria deals with the investigation of development processes for a methodology.

Table 6: Process Comparison

Criteria	Description
Specification	This methodology provides ways of forming a system specification from scratch.
Life-cycle Coverage	This methodology covers steps from analysis, design, implementation, and testing throughout system development.
Architecture Design	This methodology provides mechanisms to facilitate design by using patterns or modules.
Implementation Tools	This methodology provides suggestions on how to implement Agents in the system
Deployment	This methodology provides support for practical deployment of Agents.

Table 7 compares the methodologies using the *Process* criteria described in Table 6.

Table 7. Comparison using Process criteria

	Tropos	Gaia	MaSE
Process(C)			
System specification	Stakeholders analysis	Role analysis	Use-cases goal and role analysis
Life-cycle coverage	Yes	Yes	Yes
Architecture Design	Yes	No	Yes
Implementation	Yes	No	Yes
Deployment	No	Yes	Yes

3.3.1 Detailed Level Questions

1. How well does the methodology define the system domain?
2. How well does the process cover the whole lifecycle development?
3. How well do the transitions between phases preserve goals?

A detailed analysis and comparison of methodologies based on the *Process* questions above is provided in the

Appendix (also see Lin, 2003).

3.4 Criteria D: Pragmatics

Pragmatics refers to real use scenarios as developers apply methodology in building Agent-based systems. This provides reviews in real situations from instituting concepts, building models, and implementing details. This division deals with the exploration of practical deployment while using a methodology.

Table 8: Pragmatics Comparison

Criteria	Description
Tools Available	There are resources and tools ready to use.
Required Expertise	There is a required level of background or expertise to apply the methodology.
Modeling Suitability	This methodology is based on a specific architecture.
Domain Applicability	This methodology is suitable for a specific application domain.
Scalability	This methodology is able to handle a large number of Agents in an application.

Table 9 compares the methodologies using the *Pragmatics* criteria from Table 8.

Table 9. Comparison using Pragmatics

	Tropos	Gaia	MaSE
Pragmatics(D)			
Tools available	No	No	Yes
Required expertise	No	No	No
Modeling suitability	BDI	No	No
Domain applicability	Yes	Yes	Yes
Scalability	Yes	Yes	Yes

3.4.1 Detailed Level Questions

1. Is the methodology easy to use?
2. Do Agent concepts and properties evolve easily?
3. Is the Agent-oriented methodology flexible enough in reengineering?
4. Are paradigms and architectures suitable in general cases?

A detailed analysis and comparison of methodologies based on the *Pragmatics* questions above is provided in the Appendix (also see Lin, 2003).

3.5 Summary of Observations

Based on the evaluation using the above criteria to compare Tropos (Bresciani et al., 2004) , Gaia (Wooldridge et al., 2000), MaSe (Wood & DeLoach, 2001), Extending UML (Kavi et al., 2003), and Object-Oriented frameworks (Garcia, Silva, Chavez, & Lucena, 2002), we observe that a good AOSE methodology for MAS should include the following:

- A good mental mechanism to support an Agents' autonomy, adaptation and collaboration,
- Communication protocols for interactions among Agents,
- Description and management of goals,
- Practical conceptual models to ease the management of design complexity,
- Notations for clearly and concisely expressing key processes and properties,
- An executable and reliable life-cycle software engineering process,
- Modular and refinement capabilities are needed to analyze and integrate elements in the system.

4 Conclusions

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. The more complex an Agent-based system is, the more sophisticated the methodology to design such systems must be. At present there are no consensus standards on how to create Agents or model them in the development process. A study of various proposals for creating Agent-based systems is under way to gain insights on what attributes are useful in leading to better design methodologies.

In this work, we described agent-based system as they are used in a variety of application domains. Since there is no single definition of agents, we described the more commonly accepted properties of agents. We then described some of the available methodologies and software engineering processes for designing agent-oriented software systems. We created a framework for comparing the available AOSE methodologies. Our framework is based on both software engineering process principles and agent characteristics. The evaluation framework is composed of two levels. At the overview level we evaluate AOSE methodologies to determine whether a criteria have been met by the methodology. We then proposed questions at the detailed level concerning logical relationships among these criteria, and provide answers as statements for comparison.

5 References

Bresciani, P., Giorgini, P., Hiunchiglia, F., Mylopoulos, J., & Perini, A. (2004). Tropos: An agent-oriented software development methodology, technical report #dit-02-0015. *AAMAS Journal, 8*(3), 203-236.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: The tropos project,information systems.Elsevier, Amsterdam, The Netherlands.

Dam, K. H., & Winikoff, M. (2003). *Comparing agent-oriented methodologies.* Paper presented at the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003), Melbourne, Australia.

Garcia, A., Silva, V., Chavez, C., & Lucena, C. (2002). Engineering multi-agent systems with patterns and aspects. *Journal of the Brazilian Computer Society, SBC, Special Issue on Software Engineering and Databases*.

Gernuzzi, L., & Rossi, G. (2002). *On the evaluation of agent oriented modeling methods.* Paper presented at the Proceedings of Agent Oriented Methodology Workshop, Seattle, Washington.

Jennings, N. R. (2000). On agent-based software engineering. *Artifical Intelligence*(177), 277-296.

Jennings, N. R., & Wooldridge, M. J. (1998). Applications of intelligent agents. *Agent Technology: Foundations, Applications, and Markets*, 3-28.

Juan, T., Pearce, A., & Sterling, L. (2002). Roadmap: Extending the gaia methodology for complex open systems. *Autonomous Agents and Multi-Agent Systems*.

Kavi, K., Kung, D. C., Bhambhani, H., Pandcholi, G., Kanikarla, M., & Shah, R. (2003, May 3-10, 2003). *Extending uml to modeling and design of multi agent systems.* Paper presented at the 2nd Intl Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS2003), held in conjunction with the International Conference on Software Engineering, Portland, OR.

Lin, Chia-En (2003. A comparison of agent-oriented software engineering frameworks and methodologies, MS Thesis, Dept of CSE, University of North Texas, Denton, TX 76203.

Perini, A., Bresciani, P., Giunchiglia, F., Giorgini, P., & Mylopoulos, J. (2001, 28 May - 1 June 2001). *A knowledge level software engineering methodology for agent oriented programming.* Paper presented at the Fifth International Conference on Autonomous Agents, Montreal, Canada.

Perini, A., & Susi, A. (2003). *Discussing strategies for software architecting and designing from an agent-oriented point of view.* Paper presented at the ICSE '03, Portland, Oregon.

Rao, A. S., & Georgeff, M. P. (1991). *Modeling rational agents within a bdi-architecture.* Paper presented at the Second International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA.

Sabas, A., Badri, M., & Delisle, S. (2002). *A multidimentional framework for the evaluation of multiagent system methodologies.* Paper presented at the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2002), Orlando, Florida.

Shehory, O., & Sturm, A. (2001, February 11-13, 2001). *Evaluation of modeling techniques for agent-based systems.* Paper presented at the AGENTS '01, Montreal, Quebec, Canada.

Sturm, A., & Shehory, O. (2003). *A framework for evaluating agent-oriented methodologies.* Paper presented at the Fifth International Bi-Conference Workshop on Agent-Oriented Information System (AOIS-2003).

Tveit, A. (2001). *A survey of agent-oriented software engineering.* Paper presented at the NTNU Computer Science Graduate Student Conference, Norwegian University of Science and Technology, Trondheim, Norway.

Wood, M. F., & DeLoach, S. A. (2001). An overview of the multiagent systems engineering methodology in agent-oriented software engineering. *P. Ciancarini, M. Wooldridge, (Eds.) Lecture Notes in Computer Science, 1957*.

Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems, 3*, 285-312.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2000). *Organizational abstractions for the analysis and design of multi-agent systems.* Paper presented at the AOSE 2000.

Zambonelli, F., Jennings, N. R., & Woolridge, M. (2003). *ACM Transactions on Software Engineering and Methodology, 12*(3), 317-370.

Appendix: Detailed Evaluation of AOSE Methodologies

(A) Concepts and Properties:

[QA1.] *What concepts are at the root of the methodology and what are the advantages?*

[Ans:] A role concept is being used in most of the methodologies. Roles can have specific responsibilities in a social or organizational setting. Agents are designed to represent abstract concepts while running corresponding tasks that can fulfill their goals. Governed by a hierarchy, an Agent's communication can be regulated by the rank of roles. Role modeling provides designers a way to comprehend an Agent's activities at run-time, and to effectively describe them. Gaia and MaSE are the main representatives that use role modeling. The BDI architecture is also a good tool to use in modeling an Agent's behavior with mental states. Without mental notation, it is difficult to describe an Agent by active interactions only.

[QA2.] *How is an Agent created in a methodology?*

[Ans:] In Tropos, Gaia, and MaSE, Agent classes are usually transformed from role concepts. As roles are refined into sub-roles, corresponding tasks are refined into sub-tasks and roles are assigned with proper responsibilities. It is then that roles are realized by Agent types. Agent classes are implemented for Agent types, and an Agent capability is created within an Agent class.

[QA3.] *How well constructed is the design that deals with Agent mental mechanisms?*

[Ans:] The earlier a mental state mechanism is analyzed, the easier it is to model an Agent-based system in following processes. Tropos and MaSE use goals and intentions from the beginning of an analysis. By analyzing relationships between roles, these intentions are implicitly modeled by interactions among Agents. Although goals are accomplished by tasks, it is difficult to handle beliefs and desires in a flexible way, because no precise mental mechanism is mentioned. In Gaia, mental mechanisms are represented by a liveness property in a role schema. An Agent type can change its plan by applying different liveness equations. Goals are fixed in the design so MaSE lacks flexibility in dealing with a changing environment.

[QA4.] *How well does a design deal with an Agent's perception of its environment, and how does it react based on the perception?*

[Ans:] As mentioned in QA3, a design process with explicit mental mechanisms will have a better modeling and performance in answer to its environment.

[QA5.] *How efficient are Agents in achieving their goals?*

[Ans:] Goals are the main reasons that Agents exist. In most of the designs of comparison methodologies, specific tasks are implemented to fulfill goals. There is no difference in running time. In a dynamic environment, however, Agents must contend with the possibilities of goal conflicts. Mental reasoning and negotiations among Agents play *decisive* roles. Agent design should be embedded with mental mechanisms as well as efficient Agent-based communication to excel in accomplishing goals.

(B) Notations and Modeling Techniques:

[QB1.] *How well are notations and models formed to address Agent-based system scenarios?*

[Ans:] Tropos defines specific notations to assist designers in developing Agent-based systems with analysis and visualization tools. These notations represent goals, tasks, and Agents in their relationship of dependencies. Using these notations, designers can gain a clearer idea on Agent interactions within the system. In Gaia and MaSE, models are used to present Agents with their functionalities, such as in models of roles and interactions. Normally, models are used to present an overview of Agents in the system.

[QB2.] *How consistent and unambiguous are models while running the process?*

[Ans:] Generally speaking, the modeling used in the methodologies we compared is consistent and unambiguous. Only minor problems exist in the transition between phases. For example, in Tropos late requirement analysis, may adversely impact analysis diagrams due to a newly added system-to-be actor. In such cases, the consistency of analysis may be called into question causing the analysis to be restarted. Another example, is the transformation from role model to agent model using Gaia. Designers must decide on what and how many agent types are needed and there is no support provided from the agent role schema design phase. Furthermore, there is a potential risk from using multiple same agent types that could lead to resource contention and deadlock troubles.

[QB3.] *How well does the modeling technique address traceability and reuse?*

[Ans:] MaSE is the one that emphasizes models that can be traced back and forth in each analysis and design layer. Modeling in each layer is smoothly derived from its upper layer with explicit rules.

[QB4.] *How well does the modeling technique represent Agents?*

[Ans:] Methodologies using Agent-based features, such as goals or mental states, to analyze and model the system are better for describing and modeling Agents. For a counter-example, Gaia did not use any explicit goals or mental states to model the system. As a result, the overall system has less explicit Agent-oriented features and flexible Agent management properties.

(C) Process:

[QC1.] *How well does the methodology define the system domain?*

[Ans:] In the Tropos, Gaia, and MaSE, methodologies role concepts are used to explore stakeholder interaction, which depicts the main system specification. In MaSE, Use-cases and UML modeling are used for the system specification. Each provides a way to explore the system domain by extracting roles or agents from a requirements statement.

[QC2.] *How well does the process cover the whole lifecycle development?*

[Ans:] Most Agent-based methodologies cover the analysis and design phases in their respective design/development process. In the implementation phase, some suggest applying agent-based implementation toolkits. For example, Tropos adopts JACK as the implementation toolkit because it easily maps to a BDI architecture. In most cases, Agents are implemented as object classes by recognizing Agent types analyzed from design phases. Also, reuse and maintenance of design is seldom available.

[QC3.] *How well are transitions between phases in a process to preserve goals?*

[Ans:] In each Agent-based methodology, Agents should be managed to bear their goals and achieve them successfully. Keeping goals in each design process for each Agent type is crucial to a successful process.

(D) Pragmatics

[QD1.] *Is the methodology easy to use?*

[Ans:] From an empirical study of these methodologies, Gaia is the simplest to use (this is based on constructing a *prototype* Agent-based system). MaSE provides improved layered steps and phases while building models. Tropos emphasizes the notations used throughout the design process; but rationale analysis introduces complexity.

[QD2.] *Do Agent concepts and properties evolve easily?*

[Ans:] Deriving an analysis from a requirement statement using abstract concepts is not difficult to do; however, in most methodologies there is no explicit rule for developers to follow. Because experiences and emphases differ with developers, the exploration of initial stakeholder interactions could alter concepts and/or properties. In Tropos, especially, exploring dependencies between stakeholders is not an easy task. Practitioners must assure that the analysis proceeds in the "right" direction.

[QD3.] *Is the Agent-oriented methodology flexible enough in re-engineering?*

[Ans:] Tropos is problematic because designers must refine all the analyses from the initial stakeholder models/specifications. Gaia may also lead to thorny problems when communication bottlenecks cause the redesign of Agent roles and interactions starting from the beginning.

[QD4.] Are the paradigms and architectures of those methodologies compared suitable for general applications?

[Ans:] All of the methodologies studied provide good support for most general purpose applications.