



UNIVERSITY OF  
CALGARY

**A tutorial report for SENG 609.22  
Agent Based Software Engineering  
Course Instructor: Dr. Behrouz H. Far**

**A Tutorial on  
Agent Based Software Engineering**

**Qun Zhou**

**December, 2002**

## **Abstract**

Agent oriented software development is one of the most contributions to the field of software engineering. The method of analysis and design is not so mature as structured analysis and design and object oriented analysis and design. In this case, robust and easy-to-use methodologies and tools have to be developed for agent based software engineering. As we know, Analysis is an important phase in agent based software development. This tutorial will introduce an agent based analysis methodology, called Multi-agent Systems Engineering Methodology. It is a high level methodology for analysis and design. In this tutorial we only focus on analysis phase. This tutorial introduces the theory of this method and steps for analyzing agent-based systems. Furthermore, we analyze this method and compare it with Gaia methodology. The object of this paper is to study a comparably new method and try to get a clue on how to improve the effectiveness in practical analysis and design for agent based software development.

## **1. Introduction**

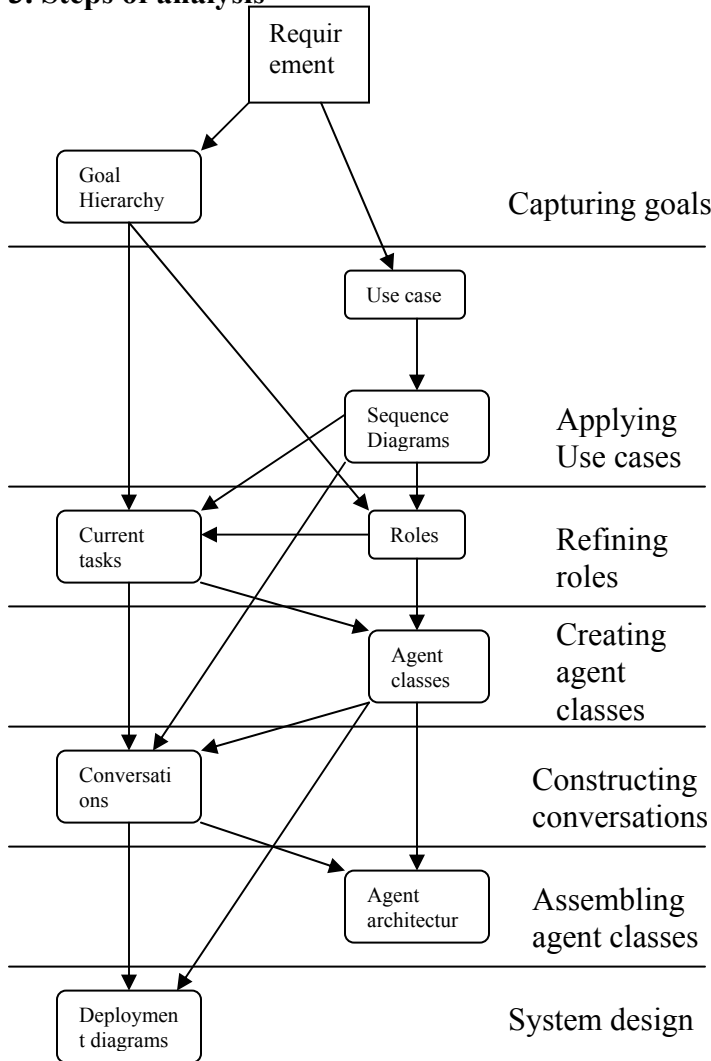
Constructing multi-agent systems is difficult because they have all the problems of traditional distributed, concurrent systems, and the additional difficulties that arise from flexible requirements and sophisticated interactions. There are mainly two difficulties in agent based software development. At first, it is lack of concrete and effective methods to clearly structure applications as multi-agent systems. Second, there are not tool kits that are flexible enough to specify the numerous characteristics of agents. MaSE methodology tries to resolve the first problem by providing a method to constructing multi-agent systems from requirement analysis to implementation. MaSE finally will generate code according to system design. Before we start to learn MaSE we need to realize its application scope. It is assumed that the system being created is closed and that an agent that participates in the system communication protocols encapsulates all external interfaces. This methodology does not consider dynamic systems where agents can be created, destroyed, or moved during execution. And inter-agent conversations are assumed to be one-to-one, as opposed to multicast. The designed system could not be very large. The target is ten or less software agent class.

## **2. General description of MaSE methodology**

Wood and Deloach suggest the Multi-agent Systems Engineering Methodology. It is similar to Gaia with respect to generality and the application domain supported. The goal of MaSE is to lead the designer from the initial system specification to the implemented agent system. The MaSE methodology is divided into seven sections in a logical pipeline. Capturing goals transforms the initial system specification into a structured hierarchy of system specification into a structured hierarchy of system goals. Applying use case creates use cases and sequence diagrams based on the initial system specification. Refining roles creates roles that are responsible for the goals defined in phase one. Creating agent classes, maps roles to agent classes in an agent class diagram. Constructing conversations defines a coordination protocol in the form of state diagrams

that define the conversation state for interacting agents. In assembling agent classes phase, the internal functionality of agent classes are created. Finally, system design creates actual agent instances based on the agent classes; the final result is presented in a deployment diagram.

### 3. Steps of analysis



**Figure 1. Agent based system analysis and design process**

This picture shows two main ideas of MaSE. The first one is each rectangle represents a model used during the analysis. These models could be mapped into the analysis steps as noted by textual. Besides, those arrows indicate how models are related. The model at the back end of an arrow is the input of the model at the front end of an arrow. The final generated model is deployment model. MaSE starts from a requirement specification and ends at the deployment model. In the following part we discuss activities of each step in more detail.

### 3.1 Capturing goals

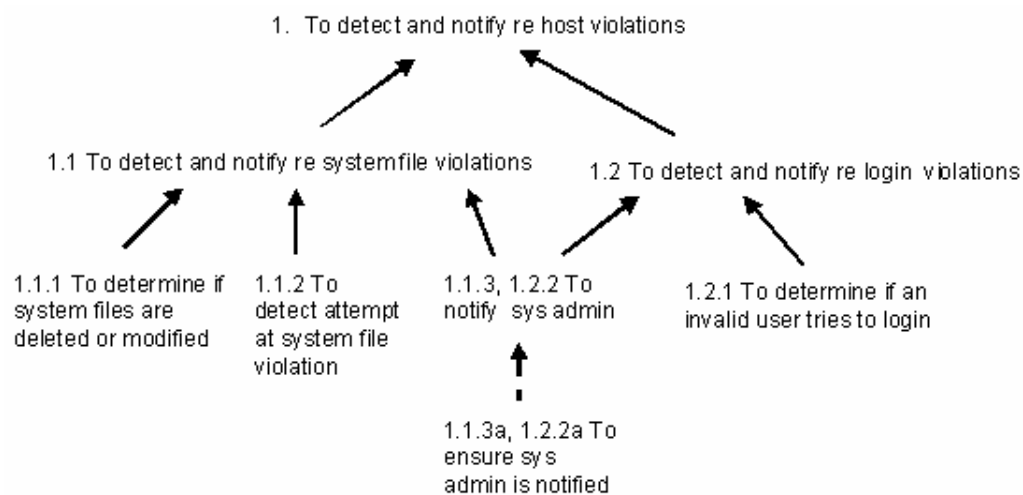
This phase is drawn in a large part from analysis patterns- capturing and structuring goals. A goal is defined as a system-level objective. The objective of this phase is to identify the purpose of each activity, which is stated and hierarchical, and relationships between purposes and a goal hierarchy diagrams is formed. It is the responsibility of the analysts to capture and emphasize what is important. Analysts must also identify a foundation for the requirements model, but the tasks and activities employed by a business are variable. There are two phases in this step; identifying and structuring goals. Identifying goals begins by extracting scenarios. Goal statements for each scenario is ascertained by posing the questions, "what is the objective of this scenario?". Relationships between scenarios and goals must be preserved so that requirements are traceable. Scenarios and related goals should be structured so the main sequence of interaction, variations, conditional exceptions, and subordinate details can be distinguished from one another. Redundant goals and actions should be promoted to a high level object, any repeated goals can be viewed as instances of the same class. Following is an example.

Requirements are stated informally, as follows:

The system is responsible for dealing with host violations, in particular login violations and system file intrusions. The system administrator is notified of suspected or attempted intrusions.

It is necessary to validate the date, time and existence of system files periodically, every few minutes. When a file is not found or a new version appears, the system administrator needs to be notified. When a user tries to modify or delete a system file, the system administrator needs to be notified.

A user tries to login when he or she does not have a valid account. If this occurs once or twice in a short period of time, it is not a violation. Three or more attempts is a violation that needs to be reported. The system administrator may not be available to receive a notification. This can be due to a network failure or the fact



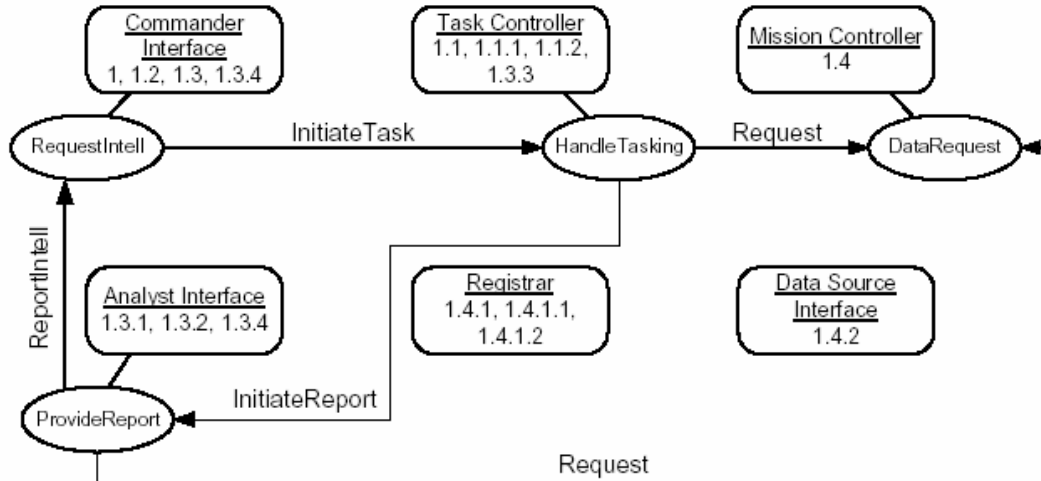
**Figure 2. Goal Hierarchy Diagram**

### 3.2 Applying use cases

The second phase of MaSE looks down the road toward constructing these conversations and creates use cases. The applying use cases phase captures use cases from the initial system requirements and restructures them as a sequence diagram. Use cases are narrative description of a sequence of events that define desired system behavior. Roles could be regarded as the participator in a system behavior. A sequence diagram depicts a sequence of messages between multiple agent roles. A sequence diagram is used to determine the minimum set of messages that must be passed between roles. Typically, we create at least one sequence from a use case. If there are several possible scenarios, multiple sequence diagrams are created.

### 3.3 Refining goals

The third step of MaSE is to transform the structured goals of the Goal Hierarchy Diagram into a form that is more useful for constructing systems: roles. A role is an abstract description of an entity's expected function and encapsulates the system goals that it has been assigned the responsibility of fulfilling. The general case transformation of goals to roles is one-to one; each goal maps to a role. There are many exceptional situations where it is useful to combine goals. Similar or related goals may be combined into single roles. Roles are documented in a role model. Following is an example.



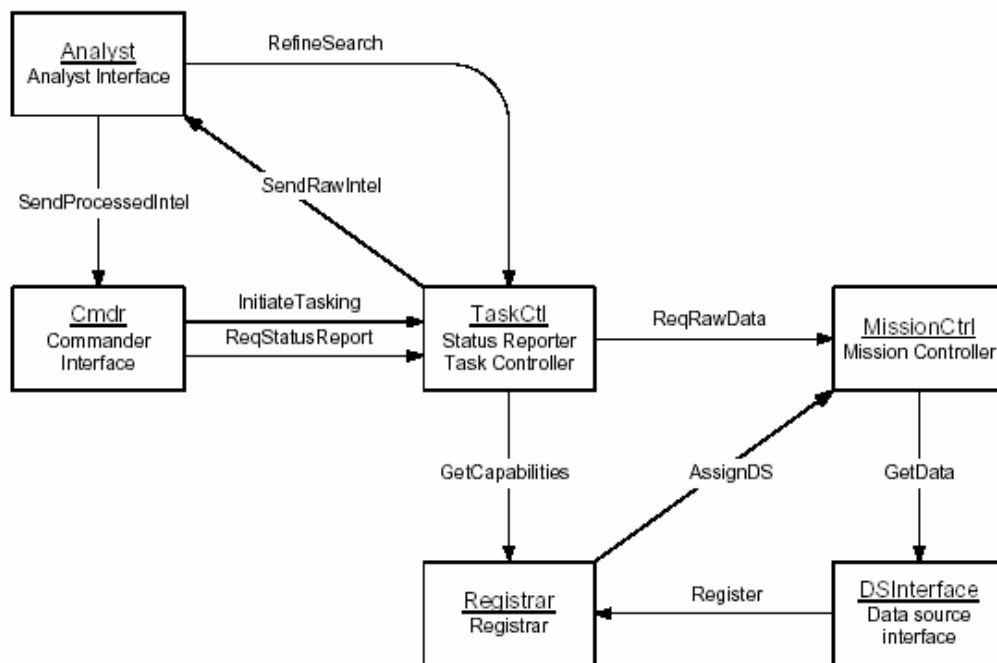
**Figure 3. Roles and Tasks Diagram**

Each rectangle represents a role. The number under role name is associated goal number in Goal Hierarchy Diagram. Each oval represents the task identified for this role. Solid lines indicate peer-to-peer communication between roles. Dashed lines denote communication between concurrent tasks within the same role. A line is dashed if it will only occur within the same instance of the role in the system. Role may not share tasks.

After roles are created, tasks are associated with each role. A task is documented by state chart.

### 3.4 Creating Agent classes

In the creating Agent Classes phase of the MaSE methodology, the agent classes are identified from component roles. An agent class diagram is built in this phase. Each class of agents belongs to an agent class that is much like an object-oriented class. The difference lies in the interface to the agent. In object-oriented classes, each class has attributes and methods. External objects may look at the object's attribute values and invoke its methods. However, each agent has a goal and may or may not provide services to other agents. Following is an example of agent class diagram.



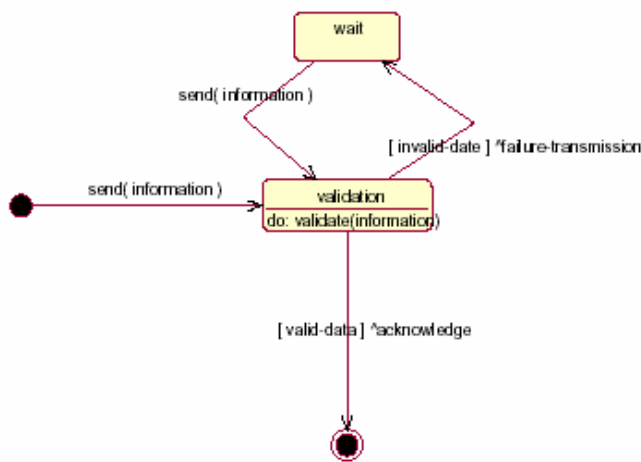
**Figure 4. Agent Class Diagram**

Rectangle is the agent class. Role names are written under class name. Each arrow represents conversations between two classes. The name of conversation is written on each arrow. The conversation of an agent class is those activities it participates in, either as an initiator or responder. The designer may combine multiple roles in a single agent class or map a single role to multiple agent classes. The conversation could be mapped to realize tasks in role model that is produced in previous step.

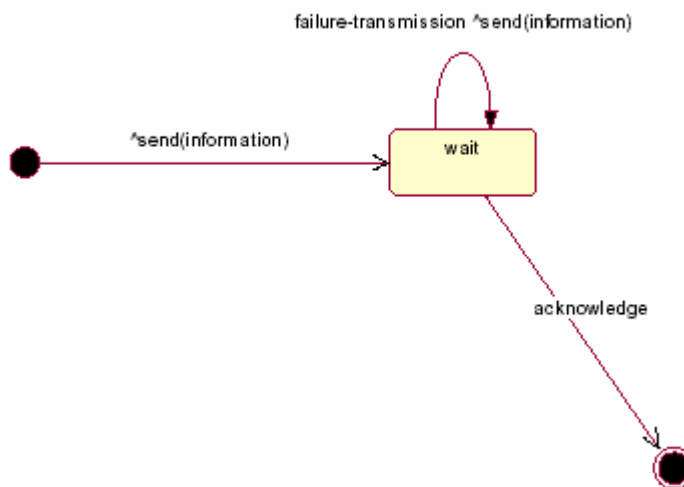
### 3.5 Constructing conversations

Constructing conversations is to define a coordination protocol between two agents. In order to provide services, an agent must interact with other agents in the system. Interactions between agents take place when agents have conversations. These

conversations define a shared convention about message exchanges that the agents use to coordinate their actions and are defined using communication class diagrams. A conversation consists of two communication class diagrams. One is for initiator and the other one is for responder. A communication class diagram is a pair of finite state machines that define the conversation states of the two participant agent classes. The problems encountered in this phase deal with building the finite state automata that define the operation and protocol of conversation. Conversations must support and be consistent with all sequence diagrams derived earlier. Conversations are built by first adding all possible states and transitions that can be derived from the sequence diagrams and tasks. And then designers refine them iteratively with the consideration of previous diagrams. Following are examples of Communication Class Diagrams.



**Figure 5.1. Conversation Class Diagram: Send information conversation-receiver role**



**Figure 5.2. Conversation Class Diagram: Send information conversation-sender role**

Conversation start when one agent sends a message to another agent. The label on the transition from the start state defines the affect of the transition. Once in a non-start state, the conversation waits until a transition occurs that takes it to a new state. In figure 5.2, if the conversation is in the wait, receipt of an acknowledgement or failure-transmission message will cause the conversation to transition to the end state or back to the wait state respectively. Once the conversation transitions to the end state, the conversation is over.

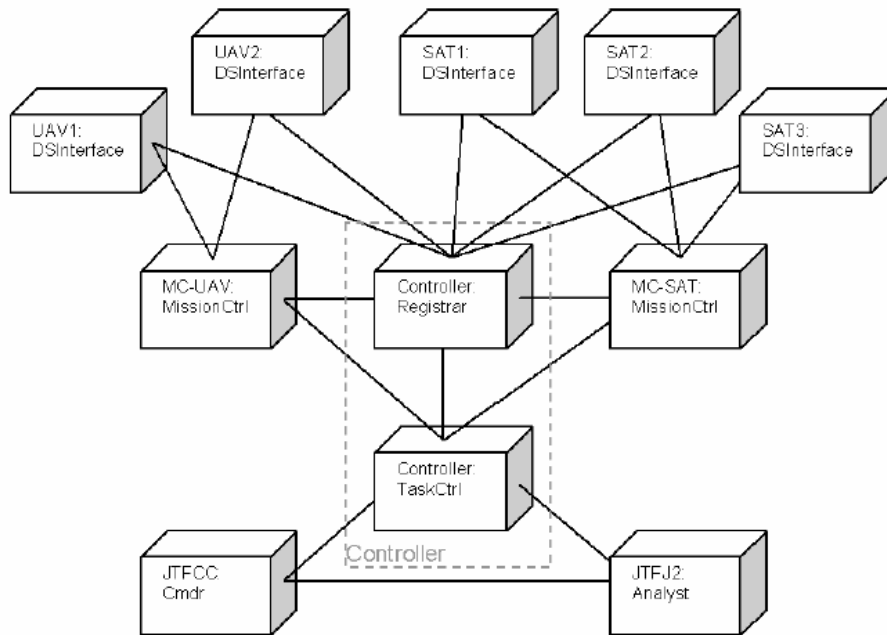
### 3.6 Assembling Agent classes

In this phase, the internals of agent classes are created. A designer can either define components from scratch or use pre-existing components. There some defined architecture styles, such as Belief-Desire-Intention, reactive, planning, knowledge based, and so on.

Constructing conversations and agent class assembly is closely related activities. They should be preceded in parallel way. Which to do first depends on the style of conversations of the system uses. The designer should design conversations first if the system consists of many simple conversations. It is generally better to define the agents first if there are complex conversations.

### 3.7 System Design

In this phase agent classes' instances are realized. MaSE uses a Deployment Diagram to show the numbers, types, and locations of agents within a system. The idea of instantiating agents from agent classes is the same as instantiating objects from object classes in object-oriented programming. Following is an example.



**Figure 6. Deployment Diagram**

The three dimensional boxes are agents, and the connecting lines represent conversations between agents. If there are multiple instances of a class a dashed-line box indicates that agents are housed on the same physical platform. A system must be arranged in a Deployment Diagram before it can be implemented in code. A final element to consider is code generation. This step may be automatically done.

#### **4. Critique of MaSE methodology**

MaSE is a very straightforward method. It is easy to be followed by practitioners. But there are still some uncertainties and weaknesses in this method. These uncertainties will influence how effective MaSE is in practical application. It depends on how experienced analysts and designers are, the characteristics of project or development organization. For software engineering researchers it is necessary to find and resolve these uncertainties by using other methodologies to support the original MaSE. Followings are the main problems and uncertainties in MaSE that have been studied. At first, the processes only concentrate on requirement and system analysis. It is weak in system design. We will discuss about it in analysis of each step. Besides, this method doesn't support the documentation of the analysis and design. Most of the work products are diagrams. In fact, documentation is a good supplementary for explaining the accurate meaning in the diagrams because the analysis results will be discussed within a design group routinely. It is very important to facilitate understanding for others. We can also see that diagrams are based on UML diagramming tools. There may be some training effort needed when this method is introduced newly in an organization or project. MaSE doesn't cover how to implement the agent system in detail. It is still an ongoing research. When we look into each step we can find that some steps are still full of uncertainties. In the applying use cases phase, how to identify use cases is still an ad hoc activity. Analysts have to decide which use cases should be defined and deal with the similarities between different use cases. And in this phase, agent roles have been considered. But this is the task in refining goals phase so that some confused and mistake identification of roles is inevitable as this activity start too early. In the refining goals phase, the problem is what is task and how to find them. This is the most challenging activity in this phase. Tasks should be separated from a use case. Some considerations are roles shouldn't hold the same task and combination of tasks within the system must perform all of the functionalities. There isn't method to deal with this problem in MaSE. In the creating agent class phase, designers may combine multiple roles in a single agent class or map a single role to multiple agent class. There isn't any guidance on how to coordinate roles into one agent. This absolutely depends on experience and potential architecture requirement possibly. Tasks identified in previous step should be related to conversations, the relationship is conversations support part of those tasks. If the relationship is clearly mapped it is benefit to find complete conversation and related agent classes. It also provides a method to verify constructed agent class diagram. It is the same problem in constructing conversations phase. Mapping between conversations and sequence diagram is not existed. In the assembling Agent Classes phase, a scenario should support assigning behaviors to each agent. Roughly speaking, those behaviors in each agent are related to conversational class diagrams. But these do not cover all of them. Tasks in the agent class diagram are also a reference of it. But MaSE does not define these relationships.

## **5. Comparison with Gaia methodology**

### **5.1 Differences**

Gaia and MaSE are different in mainly four aspects. At first, Gaia method includes both macro level and micro level of analysis and design. However, MaSE only address the micro level analysis and design. Second, Gaia method is appropriate to large-scale agent based system development; the upper limit of agent is up to 100. MaSE is fit to small-scale agent system development. The upper limit of number of agent is only 10. Third, Gaia advocates using of textual documentation mixed with graphs. MaSE only uses diagrams to analyze and design. Gaia technique may design an agent system with each agent using different programming language and techniques. MaSE try to generate code for all agents. That indicates all agents should use the same kind of programming language.

### **5.2 Similarities**

On the other hand, they also have some similar characteristics. At first, they all support the analysis and design activities begin from requirement statement to design that is sufficiently detailed that it can be implemented directly. The analyst moves from abstract to increasingly concrete concepts. Both of them borrow some terminology and notation from object-oriented analysis and design. In MaSE, there is state chart that is similar with state transition chart in UML notation. They have very similar analysis and design steps and processes even though they use different notation for documentation. The role model in Gaia is equivalent to goal hierarchy, use case, and sequence diagram in MaSE. Interaction model in Gaia has the same function with refined role and task diagram in MaSE. The agent model in Gaia can be met by Agent class diagram in MaSE technique. The service model in Gaia could be mapped to conversation diagram in MaSE. Acquaintance model in Gaia is right the deployment diagram in MaSE. MaSE models these views of an agent system in concreter style with defined notation and their meaning in the diagram. Gaia models them in a more flexible way with emphasizes on concepts of these models.

## **6. Conclusion**

This tutorial discusses an agent based software analysis and design methodology called MaSE. It is fit to small-scale agent based system development. Analysts may follow the seven phases begin with a requirement statement and end with a documentation of analysis and design. MaSE guides analysts step by step. Diagrams are related with each other. The following diagrams could be built based on previous diagrams. Usually, the steps are solid and they have interaction with each other. What should be imposed is experience will dominate the success of this method, as there are still some uncertainties during the whole analysis and design process. This method is still on the road of improving. Code generation may be added into this method in the near future. We also compare the two most popular agent based analysis and design techniques. One is Gaia and the other one is MaSE. They have very similar basic idea in analysis and design

process. Gaia is more flexible. Gaia could be used in both conceptual level and concrete level analysis and design. It supports multiple programming languages. However, MaSE is more straightforward and concrete. Each of those diagrams has defined notations. In other words, MaSE is easier to be applied in practical situation and fit top small agent system development.

## References

1. Amund Tveit, A Survey of Agent-Oriented Software Engineering, first NTNU CSGSC, May 2001.
2. Scott A. Deloach, multiagent Systems Engineering: A Methodology And Language for Designing Agent Systems, proceeding of Agent Oriented Information Systems, Pages 45-57, 1999.
3. Wood M. F. , Deloach S. A., An Overview of the Multiagent Systems Engineering Methodology. The first International Workshop on Agent-Oriented Software Engineering(AOSE-2000), 2000.
4. Michael Wooldridge, Nicholas R. Jennings, and David Kinny, A Methodology for Agent-oriented Analysis and Design, Proceedings of the Third International Conference on Autonomous Agents (Agents `99).
5. Kendall, Elizabeth A., and Zhao, L.: Capturing and Structuring Goals. Workshop on Use Case Patterns, Object Oriented Programming Systems Languages and Architectures (1998).
6. Deloach. S. A., Wood M. F.: Multiagent Systems Engineering: the Analysis Phase. Technical Report, Air Force Institute of Technology, AFIT/EN-TR-00-02, June 2000.