Workshop: AI

Title: Pre-process for Architecture and Knowledge Co-design of an Intelligent Agent System

Abstract:

An agent based system is a complex software system with some functional constrains, designing and building such a system is a complex task. This article presents a method for agent architecture and knowledge co-design. A use-case approach is used to elicit system requirements from user's point of view, and the related use cases are assigned to corresponding roles. After the role model is refined by an analyzer, the agent architecture and high-level agent knowledge can be derived from the role model. Well-designed agent knowledge plays an important part for agent system to solve problems and make decisions. An ontology approach is used to represent the knowledge of agents and the ontology is implemented by conceptual graph. In a multi-agent system (MAS), the ontology also has the property of knowledge sharing and reuse that facilitate cooperation and negotiation among agents. The contribution of this article is a proposal to a systematic approach for implicit requirement analysis and a possibility for agent architecture and knowledge co-design for agent-based systems.

Keywords: use cases, agent-based system, ontology, requirement analysis

Names, etc.:

Name:	**Chiung-Hon Leon Lee**	**Alan Liu**
E-mail:	**leon_miao@yahoo.com.tw**	**aliu@ee.ccu.edu.tw**
Affliation:	**Dept. of Electrical Engineering, National Chung Cheng University**	
Address	**160 San-Hsing, Min-Hsiung, Chia-Yi, 621, Taiwan**	
Phone:	**886-5-2720411 x23272**	**886-5-2720411 x33220; Fax: 272-0862**

Name of the contact author: Chiung-Hon Leon Lee

Pre-process for Architecture and Knowledge Co-design of an Intelligent Agent System

Chiung-Hon Leon Lee and Alan Liu
leon_miao@yahoo.com.tw aliu@ee.ccu.edu.tw
Department of Electrical Engineering
National Chung Cheng University
Min-Hsiung, Chia-Yi, Taiwan

Abstract:

An agent based system is a complex software system with some functional constrains, designing and building such a system is a complex task. This article presents a method for agent architecture and knowledge co-design. A use-case approach is used to elicit system requirements from user's point of view, and the related use cases are assigned to corresponding roles. After the role model is refined by an analyzer, the agent architecture and high-level agent knowledge can be derived from the role model. Well-designed agent knowledge plays an important part for agent system to solve problems and make decisions. An ontology approach is used to represent the knowledge of agents and the ontology is implemented by conceptual graph. In a multi-agent system (MAS), the ontology also has the property of knowledge sharing and reuse that facilitate cooperation and negotiation among agents. The contribution of this article is a proposal to a systematic approach for implicit requirement analysis and a possibility for agent architecture and knowledge co-design for agent-based systems.

1. Introduction:

An agent-based system [1][2] is a complex software system, and designing and building such a system is non-trivial. Jennings and Wooldridge view an agent based system as a software engineering paradigm [3,5]. The development of an agent-oriented methodology [6] and agent-oriented software engineering (AOSE) [7,8] encourages agent based system developers to think of building agent-based systems as a process of organizational design.

There are many AOSE methodologies having been proposed. Wooldridge, et al. proposed the Gaia methodology for agent-oriented analysis and design [9,10]. Zambonelli et al. improved the Gaia methodology for supporting development of Internet applications [11]. The SODA methodology proposed by Omicini [12] is also an agent-oriented methodology for Internet application. Wood and DeLoach proposed the Multi-agent Systems Engineering (MaSE) methodology for creating agent-based system [13,14]. MaSE is similar to Gaia and supports more general application domain. Giunchiglia et al. suggested a novel agent-oriented software development methodology that called Tropos, focusing on comprehensive software engineering for multi-agent system from early requirement analysis to system implementation [15].

An agent-based system consists of software entities called agents, which interact with themselves and other resources to perform problem solving. In such setting, each agent plays some roles in the environment. As indicated by the related work, analyzing an agent-based system by role and interaction among roles is a good point to start. A role model concept is widely used in AOSE methodologies. However, we have to identify the role and to know the requirements of each role for creating the role model, before we start to analyze the role model.

The Unified Modeling Language (UML) [16,17] is becoming widely adapted for the representation of engineering artifacts in agent-oriented software [7]. In UML, a use case diagram is used for representing requirements of a system. The use case approach [18] is a valuable technique in eliciting, analyzing, and documenting requirements. The use case diagram defines a set of use case and actors, where each use case is a sequence of actions performed by the target system that yields an observable result of value to a particular actor. Use-case-driven analysis focuses on one specific usage aspect at a time and looks at the interactions of a single category of users at a time. A use-case approach can reduce the complexity of requirements determination.

After the role model is refined by an analyzer, the agent architecture and high-level agent

knowledge can be derived from the role model. Well-designed agent knowledge plays an important role for agent system to solve problems and make decisions. In MAS, the well-design knowledge also has the properties of knowledge sharing and reuse that facilitate cooperation and negotiation among agents. For knowledge sharing and reuse purpose, an ontology [19,26] approach is widely used to represent the knowledge of agents. The agent ontology is domain dependent and how to analyze the ontology is difficult.

We construct the agent ontology to be a hierarchy model from abstract to physical views. The abstract agent ontology can be derived from the use case based role model. The abstract ontology will be a guideline to help the analyzer to find the detail ontology. The physical ontology part can be derived from the system analysis and design.

In our approach, the ontology is implemented by conceptual graph [27]. The conceptual graph hierarchy can be divided into three parts: the environment knowledge, shared knowledge, and agent-self knowledge. The environment knowledge includes the information of acquaintance, and other resource to facilitate the cooperation and negotiation among agents. The shared knowledge is the reused knowledge between agents and consists of canonical basis. When one agent has the canonical basis of another, it can predict the behavior of other agent and facilitate the cooperation of agents. The agent-self knowledge is the domain knowledge for problem solving or decision-making.

Many AOSE researchers focus their view on agent architecture analysis and design, but we believe that the relationships between agent architecture and knowledge are as important as agent architecture. For example, the knowledge parts of a rule-based agent and a case-based reasoning (CBR) agent are different. A problem solved by a rule based system may not be always solved by a CBR system. How to construct a well-designed knowledge depends on the analysis and design model of the agent architecture.

Figure 1 shows the processes and artifacts of the agent system architecture and knowledge co-design. At the beginning, the use case approach is used to elicit system requirement from user's point of view. An original use case diagram will be generated in this step. Then, related use cases are assigned to a role. The use case diagram will be reorganized into a role model and the system requirements will be performed by cooperation between roles. Use cases represent requirements of the system and will be refined by system analyzer. Roles will be identified as internal actors to find more system specific use cases. At the end of requirement analysis, we will obtain a refined role model. All the steps that we have discussed above can be performed in an iterative way. Following the requirement analysis phase, relationships between use cases and roles also can be identified to optimize the role model.

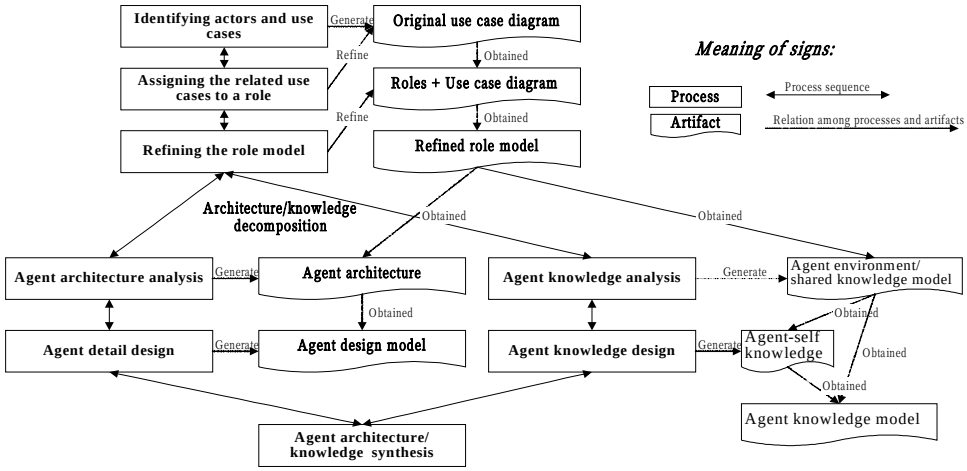

Figure 1: The process and artifact of the agent system architecture and knowledge co-design

Until the role model is constructed, the analyzer can derived the agent analysis and design model, and abstract knowledge model from it. The analysis and design model are domain dependent. The engineer can choose a proper architecture to implement the system requirements such as a rule-based architecture, CBR based architecture, or other architectures constructed by object-oriented approach such as Rational Unified Process (RUP) [28]. The agent knowledge can be constructed by ontology model from abstract to physical.

The agent architecture and knowledge co-design is a more challenging task, because each phase of co-design, such as co-partition and co-synthesis must consider the physical restriction imposed by the application domain characteristics. We believe that designing the agent architecture and knowledge at the same time can enhance the system performance and reduce the development time.

In this paper, we focus our aim in the pre-process **for Architecture and Knowledge Co-design of an Intelligent Agent System** to identify and analyze the role model and agent knowledge model.

To illustrate our methodology, a meeting scheduling system is described in Section 2. Use-case based requirements analysis is discussed in Section 3. Using the requirements analysis result, we can analyze the knowledge part of agents. The idea is detailed in Section 4. How to map the role model into an agent model is an important phase in AOSE, and this is discussed in Section 5. We give our conclusions and future works in Section 6.

2. Meeting scheduling system

We choose the meeting schedule problem mentioned in [29] as an example to illustrate our method. The meeting schedule problem could be solved by simple programming or complexly implemented by a distributed agent system [30]. Such a problem can help us address challenges of role-based requirements analysis, which is to elicit the user's requirements, map those requirements into a role model, turn a vague and contradictory mission statement into a detailed specification, and analyze role relationships.

For demonstration, we simplify the problem. We assume that the user only has a basic idea of the system. The main property of the system is to help meeting initiator to plan a meeting and give an interface for the meeting participant to register a meeting.

3. Use-case based requirement analysis

The targets of this phase are to find the whole software system's requirements and find roles to serves those requirements. Three steps are discussed: identifying actors and use case, assigning the related use cases to a role, and refine the role model with internal actor analysis. To obtain requirements of a system is a gradual process. In many cases, the user and system analyzer do not clearly know their requirements. Through the use-case-based requirement analysis phase, the system requirements can be found from coarse to delicate.

3.1 Identifying actors and use cases

This step is like general a use case approach to find the system actors and use cases. Identifying the actors is to identify the information provider, system function user, and system cooperator, etc. Different user types are represented as actors. An actor can be a real user, hardware, or virtual software program. An actor represents a particular class of user rather than an actual user. The mapping between users and actors can be one to one, one to many, many to one or many to many. In our example, we can identify two actors: meeting initiator and meeting participant.

A use case is a sequence of actions performed by the system that yields an observable result of value to a particular actor. Identifying the use case is to identify the tasks, which the system would be involved. It is like system initiation/termination, system maintenance, data maintenance, and functionality needed to modify behavior in the system, etc. In our example, we can find the main use cases includeing planning a meeting and registering a meeting. These are the minimum requirements of the system.

The original function of the system is simply described below. The meeting initiator input the meeting requirements like conceivable number of attendance, presentation need (projector), and date range. The system gives the suggestion of a meeting location, a several meeting time for participant select, and register due date for all members of the team. Then, the participant

registers the meeting before the due date. Finally, the system will announce the meeting initiator meeting planning result.

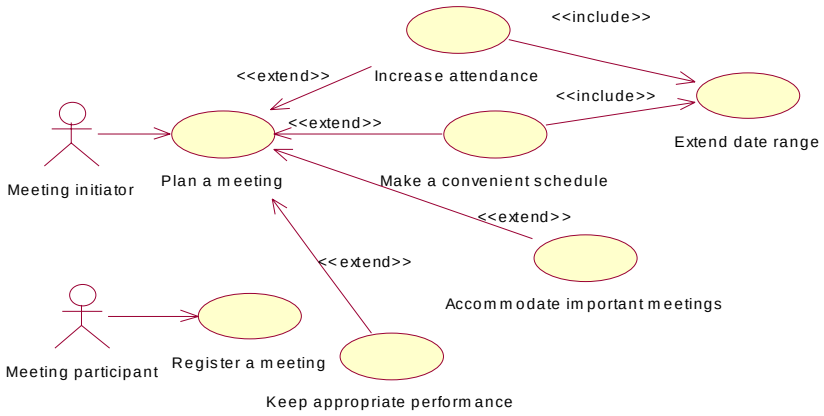

Figure 2: Meeting schedule system original use case diagram (adapted from [29]).

We assume that the meeting initiator find some extended requirements of the system. It includes maximizing the number of participant, making a convenient schedule, accommodating important meeting, and giving an appropriate performance. We also identify that extending the date range can satisfy the purpose of increasing the number of participants or making a convenient schedule. The use case diagram of the meeting schedule system is shown in Figure 2.

3.2 Assigning the related use cases to a role

When we want to accomplish a large and complex project in the real world, it needs many people who have their specific skills to work together and collaborate to complete the project. Everyone in this project plays a certain role and undertakes some jobs. The use cases that we identified in Section 3.1 can be seen as jobs that the system needs to do. We group the related use cases and assign them to a role. The role is an active class to cooperate with other roles for complete task requests from actors. When we think of what kind of roles we need, and assign the jobs to roles, and there are some considerations that we should make:

1. If the jobs need to be done in different places, they are not suitable to be performed by one role.

2. If the system achieves different jobs that may share common data or perform similar action, these jobs should be assigned to one role.

3. If the jobs assigned to different roles make the communication and interaction among roles frequently, they should be assigned to one role.

4. A job should not be grouped with others if that job works as a daemon (ever running background process). It should be assign to one role.

5. We group the jobs if there are large amount of data transmitting occurred when those jobs are performed.

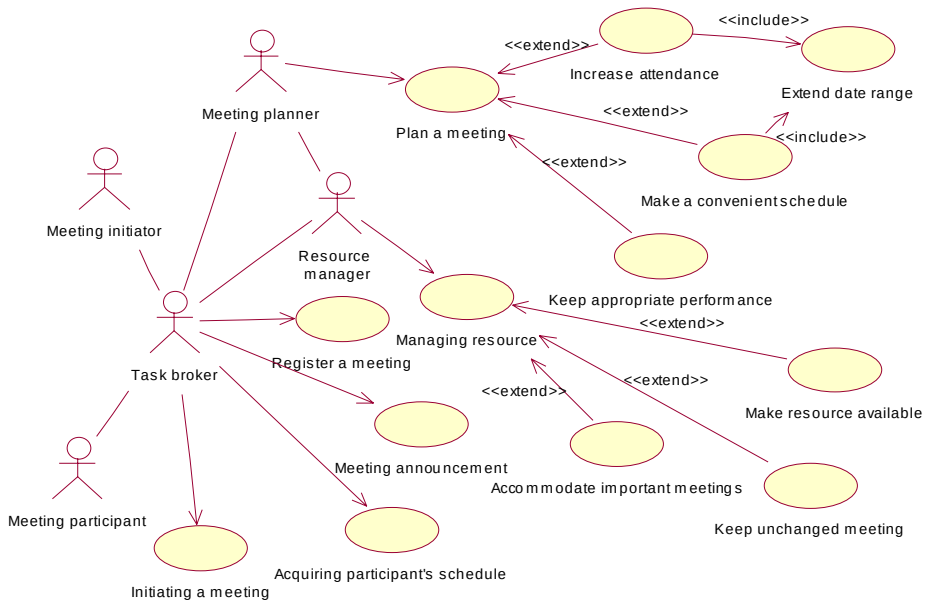

Figure 3: Grouping related use cases into a role.

A role model example is shown in Figure 3. Three roles are identified to elicit more requirements. Meeting planner offers the original task of planning a meeting. We find that the regular schedule of participants can be acquired before the initiator plans a meeting. Both meeting

information and participant's schedule information are the resources of the system. A resource manager role is specified to manage the system resource. Because accommodate important meetings have to compare the existed meeting schedule, we assign this task to resource manager. Other nonfunctional requirements of resource manager has been found such as keeping unchanged meeting or making meeting resource available. Since we need the participant's schedule information, the system have to acquire the information from participants automatically. When a meeting is decided to be held or to be canceled, the system also has to announce the meeting information to participants and the initiator automatically. A task broker is specified for these tasks. Both initiating a meeting and registering a meeting are also performed by this role.

By transferring from the original use case diagram to the role based use case diagram, we can find that the design of a system is making a change from passive (receiving the user's input) to active (dealing the information). Thus, the system has better abilities to serve the user as an agent-based system.

3.3 Refining the role model with internal actor analysis

After assign the related use cases to roles, we can find two kinds of actors in Figure 4: external actor and internal actor. An external actor is the original actor found in use case diagram and is the same as mention in section 3.1. It is a part of environment and is an external part of agent system. An internal actor is the role that identified in Section 3.2. It is an internal part of the system and has the active property to cooperate with other roles for complete task requests from external actors. Using the internal actor concept, we can find more implicit requirements of the system from internal actor's point of view and elicit more user requirements of the system from external actor's point of view.

4. Analyzing the knowledge part of agents

The agent knowledge contains the information about how to cooperate and negotiate with

other agents, how to solve the problem and make the decision, etc. In MAS, an agent can play different roles to perform different tasks. In our approach, we determine the agent's knowledge in the role model and implemented it with ontology concept for knowledge sharing and reuse purpose. The ontology also facilitates the agent knowledge level communication, cooperation and negotiation.

We implement ontology with the idea of conceptual graph [22] in this paper. A conceptual graph is a finite, connected, bipartite graph in which there exist two kinds of nodes, concepts and concepts relation. Concept is enclosed by square brackets with the concept relation by parentheses. The arc connects concepts and concept relations that do not use label. The difference between the concept node and concept relation node is that a concept node contains a type label and referent field, a concept relation node contains a type label only. There are two functions to map the concept into its labels. The function *type* maps concepts into a set of type labels T and the function *referent* maps concept into a set S of referent types. The label of concept c, is a pair type(c): referent(c).

Three components are important in a concept graph: type hierarchy, type relationship hierarchy and canonical basis. Both the type hierarchy and the type relationship hierarchy of conceptual graph is a lattice, a common form of multiple inheritance systems. For example, consider two types x and y, if $y \leq x$ that y is said to be a sub-type of x and x is said to be a super-type of y. A type may have more than one super-type and more than one sub-type. A common sub-type or super-type might exist. However every pair of types must have a minimal common super-type and a maximal common sub-type. Two special types exist. A super-type of all types is called universal type, and a sub-type of all types is called absurd type. A sample type hierarchy is shown in Figure 4.

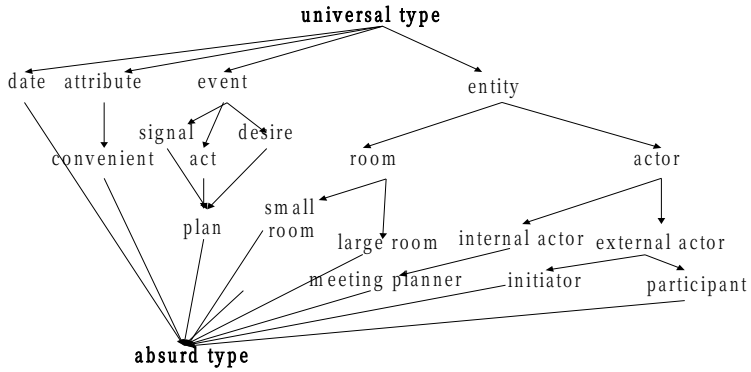

Figure 4: a sample type hierarchy.

The canonical basis is a finite set of conceptual graph B which contains all type labels in T and all referents either * or markers in S. Figure 5 shows a sample canonical basis.

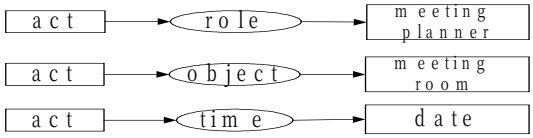

Figure 5: A sample canonical basis.

By the concept graph approach, we divide our role knowledge into three levels including role's ontology level, canonical basis level, and concept graph level. A role's ontology consists of many canonical bases which is shared, reused and cannot refined among roles. From these canonical graphs we can derive the concept that we want to indicate. Figure 6 shows this approach. Each role has their ontology and the canonical bases are shared with acquaintance to implement environment knowledge. The concept graphs are derived from canonical bases, and different agents have different concept graphs to implement the domain knowledge.

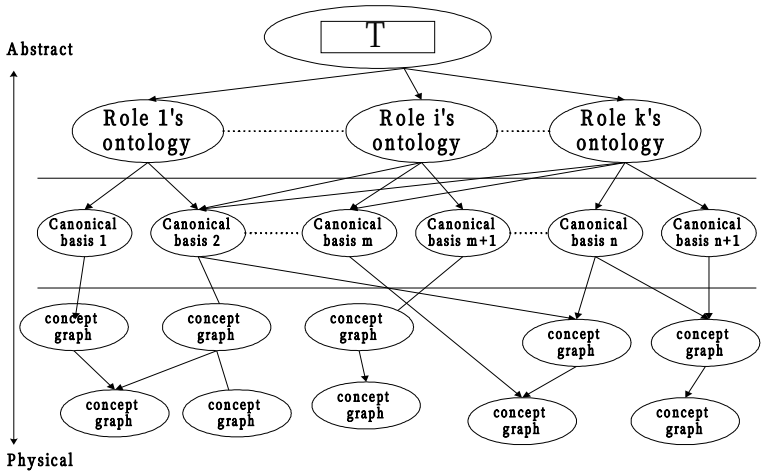

Figure 6: The hierarchy of agent's ontology.

Transforming an existing graph to a new graph, there are a number of operations in the theory of conceptual graphs (cg). The operation is *copy, restrict, fuse, join,* and *simplify* [31]. The *copy* operation allows us to form a new graph cg, that is the exact copy of cg1 and cg2. That is cg = copy (cg1) = cg1 or cg = copy (cg2) = cg2. The *restrict*-operation allows us to replace a concept type to its sub-type if *referent*(c1) conforms to *type*(c2). The *fuse*-operation allows us to combine two disjoint graphs if a concept in one graph is identical to a concept in the other. The *join*-operation allows us to delete two duplicate concepts in a graph. The *fuse, join,* and *restrict* operation allows us to implement inheritance. The *simplify*-operation allows us to delete two duplicate relations in a graph.

5. Mapping the role model to agent model

In general, an agent can play more than one roles. But to map the role model to agent model is very application specific. The mapping between the role and the agent can be one to one, one to many, many to one, or many to many. When we consider about what kind of agents we need, and the mapping between agents and roles, there are some considerations proposed by Chen [32]:

1. If the roles are distributed in different places, they are not suitable to be played by one

agent. Although there are some mobile agent technologies in MASs research, it still has its limits, so we argue that agent should not play different roles in such situation.

2. We can make an agent to play roles to decrease the communication load of the whole system when below situations occur.

The communications and interactions frequently occur between roles and that may seriously increase the communication load of the whole system.

There are quantities of data transmitting between roles.

3. For a basic mechanism in implement stage to resolve the competition for public resource. Because an agent can only play a role at one time, the roles that would be played by an agent would not have competition with the resource. However, if the roles that the agent plays are frequently required to serve in the system or they have to take a lot of time to accomplish their tasks, this mechanism would not be suitable. The system would have a bottleneck in the agent, and the performance would be reduced seriously.

4. For a simple mechanism in implement stage to resolve the conflict with goal between agents. Especially, for the conflict that can be resolved through controlling actions of agents in timing.

For mapping between roles and agents, some guideline considered below:

5. A role should not be grouped with others if that role works as a daemon (ever running background process).

6. We group the roles that have the similar action or use the same information resource.

7. We group the roles that the communications and interactions frequently occur between them.

8. We group the roles that there have a large amount of data transmitting between them.

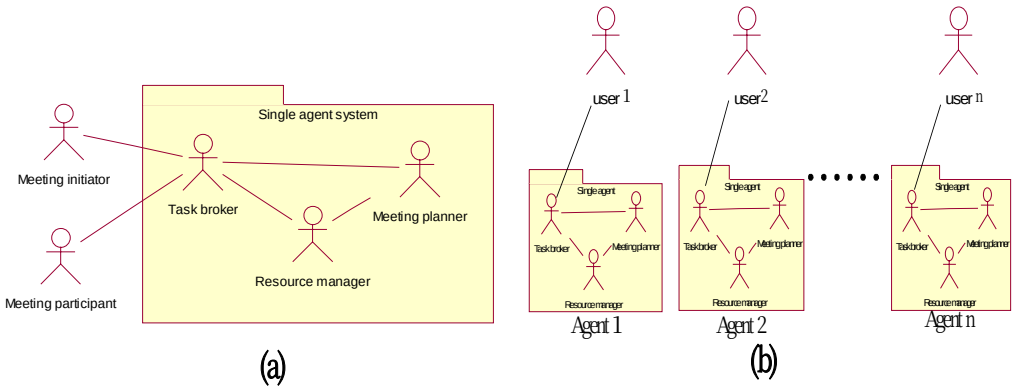

Figure 7: Different implementations of meeting schedule system.

Figure 7 shows two different implementation of meeting schedule system: a single system (a) and a multiagent system (b). In Figure 7(a), all functions of the roles are implemented in one agent. The task broker is the interface of the agent, the meeting planner is the control module of the agent, and the resource manager is the data unit of the agent. All users use the same agent for their task such as initiating a meeting and register a meeting. The other implementation is shown in Figure 7(b), the roles are grouped into one agent, one agent serves one user, and the tasks are completed by agent's cooperation. In this example we can find that mapping the role model to agent model is highly application dependent and we have to analyze the system requirement carefully. Through the construction of role model, our method can give a guideline to facilitate the mapping from role model to agent model.

Understanding the relationship among roles can help the system analyzer to refine and optimize the role model. Moreover the implicit conflict among roles can be identified. Identifying roles and mapping the role model to an agent model are essential phases in many proposed AOSE methodologies like Gaia, MaSE, and Tropos. By our methodology, the system analyzer can capture more implicit requirement, construct clear role model, and make better guideline mapping role model to the agent model.

6. Conclusions and future works

We proposed a possibility for guiding the agent based system designer to elicit system requirement, refine the role model, and construct the ontology from role model. A meeting scheduling system is used for illustrating our points. We focus our view on requirement phase of AOSE. The benefits of this approach are to facilitate the derivation of role model, to elicit more functional and nonfunctional requirements of role model, and to make easy the acquisition of ontology of role model.

The agent knowledge sharing and reuse is another important issue and challenge in this paper. We proposed an approach which implements the role ontology using conceptual graph to offer the knowledge sharing and reuse. These advantages will facilitate the cooperation of agents and reducing the agent communication load. We also can reuse the existed conceptual graph theories like searching/retrieving algorithm [31] and universal data structure [33], etc.

There is still much work that needs to be done to improve our work including agent architecture detail design, agent knowledge detail design, and agent architecture and knowledge synthesis etc. Although our methodology is complete yet, we believe we propose a new possibility for AOSE.

There are many issues in the agent knowledge detail design that includes the issues of combinatorial explosion, retrieving efficiency, and adapting correctness. As the time goes by, the agent knowledge may become large, and the combinatorial explosion problem will appear. We need to filter the useless information, store the useful agent knowledge, and retrieve the agent knowledge efficiently. How to adapt the correct knowledge is another challenge. These mechanisms need a well-designed agent architecture to support it. We can try the methodology proposed in [30] to reduce our agent knowledge. The agent communication problem introduces the issues of problem allocation, result synthesis, negotiation, and cooperation. These problems are ad-hoc and there are many people researching in these problems. Although we find the role

model, we do not perform a method to optimize the role model. We will explore the possible method to face this challenge. Finally, because this methodology can be executed in a systemic way, a software tool can be developed to facilitate the requirement evaluation and role knowledge analysis.

Reference:

[1] K. Sycara, K. Decker et al., “Distributed Intelligent Agents”, IEEE Expert, pp. 36-45, December 1996

[2] M. Wooldridge, “Agent-based computing” In Interoperable Communication Networks, 1(1), pages 71-97, January 1998

[3] M. Wooldridge. “Agent-based Software Engineering”, In IEE Proceedings on Software Engineering, 144(1), pp. 26-37, February 1997

[4] N. Jennings and M. Wooldridge, “Agent oriented software engineering”, in Proceedings of MAAMAW-99

[5] M. Wooldridge. “Agents and software engineering.” In AI*IA Notizie XI(3), pages 31-37, September 1998.

[6] C. A. Iglesias, M. Garijo, and J. C. Gonzalez. “A survey of agent-oriented methodologies.” In J. P. M¨uller, M. P. Singh, and A. S. Rao, editors, Intelligent Agents V, Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1999.

[7]A. Tveit, “A Survey of Agent-Oriented Software Engineering”, Report, Norwegian University of Science and Tech., May 2000

[8] M. Wooldridge and P.Ciancarini. “Agent-Oriented Software Engineering: The State of the Art” In P. Ciancarini and M. Wooldridge, editors, Agent-Oriented Software Engineering. Springer-Verlag Lecture Notes in AI Volume 1957, January 2001

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

[10]Wooldridge M. J., Jennings N. R. and Kinny D. “The Gaia methodology for agent-oriented analysis and design.” Journal of Autonomous Agents and Multi-Agent Systems, 3(3):285–312, September 2000.

[11] F. Zambonelli, etc. “Agent_Oriented Software Engineering for Internet Applications” Coordination of Internet Agents:Models, Technologies and Applications, chapter 13. Springer-verlag, 2000.

[12] Andrea Omicini. “Soda: Societies and infrastructures in the analysis and design of agentbased systems.” In P. Ciancarini and M. Wooldridge, editors, Agent-Oriented Software Engineering Proceedings of the First International Workshop (AOSE-2000). Springer-Verlag: Berlin, Germany, 2000.

[13]DeLoach S. A. “Multiagent Systems Engineering A Methodology and Language for Designing Agent Systems.” In Proc. of Agent Oriented Information Systems, pages 45–57, 1999.

[14] Wood M. F. and DeLoach S. A. “An Overview of the Multiagent Systems Engineering Methodology.”, The First International Workshop on Agent-Oriented Software Engineering (AOSE-2000), 2000.

[15] Giunchiglia et al. “The Tropos: Software Development Methodology Processes, Models And Diagrams”, Technical Report No. 0111-20 ITC-IRST, Nov. 2001, Submitted to AAMAS ‘02

[16] G. Booch, J. Rumbaugh, and I. Jacobson “The Unified Modeling Language User Guide”, Addison Wesley Longman, Inc. 1998

[17] J. Rumbaugh, I. Jacobson, and G. Booch, “The Unified Modeling Language Reference Manual”, Addison Wesley Longman, Inc. 1999.

[18] I. Jacobson, Object-Oriented Software Engineering, Addison Wesley Longman, Reading, Mass., 1992.

[19] M. N. Huhns and M. P. Singh "Ontologies for Agents, " *IEEE Internet Computing*, Vol. 1 pp. 81-83 Nov./Dec. 1997

[20] N.J.I. Mars, "The Role Of Ontologies In Structuring Large Knowledge Bases, " *Knowledge Building and Knowledge Sharing*, K. Fuchi and T. Yokoi, eds., pp240-248, *Tokyo: Ohmsha*, 1994.

[21] B. Chandrasekaran and J. R. Josephson, "What are ontologies, and why do we need them? " *IEEE Intelligent systems*, Jan./Feb., pp. 20-26, 1999

[22] J.F. Sowa, *Conceptual structures: information processing in Mind and Machine*, Addison-Wesley, Reading, Mass., 1984

[23] P. E. van der Vet and N. J.I. Mars, "Bottom-Up Construction Of Ontologies, " *IEEE Trans. on knowledge and data engineering*, Vol. 10, No. 4, pp. 513-525, July/Aug. 1998

[24] http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[25] N. Guarino, "Formal ontology and information systems," *Proc. of FOIS'98*, Trento, Italy, 6-8 June 1998. Amsterdam, IOS press.

[26] M. Gruninger and J. Lee, "Special Issue on Ontology Application and Design: Editors' Introduction", Communications of the ACM, Feb 2002, p.39.

[27] J.F. Sowa, *Conceptual structures: information processing in Mind and Machine*, Addison-Wesley, Reading, Mass., 1984

[28] I. Jacobson, G. Booch,, and J. Rumbaugh, "The Unified software development process", Addison Wesley Longman, Inc. 1998.

[29] Jonathan Lee and Nien-Lin Xue, "Analyzing User Requirements by Use Cases: A Goal-Driven Approach", IEEE Software Volume: 16, July-Aug. 1999, pp. 92 -101

[30] L. Garrido and K. Sycara. "Multi-agent meeting scheduling: Preliminary experimental

results.” In proceedings of the 2nd International Conference on Multi-Agent System(ICMAS-96), pp.95-102, 1996

[31] G. Ellis, "Compiling conceptual graphs, " *IEEE Tran. on knowledge and data engineering,* Vol. 7, No. 1, pp. 68-81, Feb. 1995

[32] Chia-Wei Chen “An analysis method for cooperation issues in Multi-Agent Systems” MS. Thesis, CCU, EE, 2000

[33] Robert Levinson, "UDS: A universal data structure, " *Proc. ICCS ’94*, pp. 231-250, Aug. 1994