

Architecture-Centric Object-Oriented Design Method for Multi-Agent Systems

Hongsoon Yim*, Kyehyun Cho*, Jongwoo Kim **, and Sungjoo Park*

* Graduate School of Management, KAIST (Korea Advanced Institute of Science and Technology), 373-1 Kusong-dong, Yousong-gu, Taejon 305-701, Korea
{hsyim, potin, sjpark}@kaist.ac.kr

** Department of Statistics, Chungnam National University, 220 Kung-dong, Taejon 305-764, Korea

jwkim@stat.chungnam.ac.kr

Abstracts

This paper introduces an architecture-centric object-oriented design method for MAS (Multi-Agent Systems) using the extended UML (Unified Modeling Language). The UML extension is based on design principles that are derived from characteristics of MAS and concept of software architecture which helps to design reusable and well-structured multi-agent architecture. The extension allows one to use original object-oriented method without syntactic or semantic changes which implies the preservation of OO productivity, i.e., the availability of developers and tools, the utilization of past experiences and knowledge, and the seamless integration with other systems.

Keywords: multi-agent systems, architecture, object-oriented development methods

1. Introduction

Software agents provide a new way of analyzing, designing, and implementing complex software systems. Currently, agent technology is used in wide variety of applications with range from comparatively small systems such as personalized email filters to large, complex, and mission critical systems such as air-traffic control [3, 14]. Due to the popularity and complexity of MAS (Multi-Agent Systems), they require systematic development methodology, but support of them still lacks in practice [3, 23].

For MAS development, although object-oriented methodologies are seemed to be natural, they have limitations to properly represent MAS [4, 12, 15, 16, 24]. These limitations include the representing collaboration among agents and mental states of an agent. Several agent-oriented design methods have been suggested to cover limits of the past system development methodologies [4, 12, 15, 16, 24]. These methodologies have been extended based on the past development methodologies such as object-oriented methodologies, knowledge engineering methodologies [12], and enterprise integration methodologies [16]. They, however, have own special purpose notations and techniques. So, these extensions cause the semantic and syntactic gaps between them and original object-oriented methodologies. This implies lacks of developers and tools for the development of MAS.

In addition, object-oriented methodologies have been evolved from past object-oriented methodologies that the agent-oriented methodologies have referred such as OMT [20]. Past object-oriented methodologies have limits such as functional decomposition and poor representing collaboration among objects [5]. The gap between agent-oriented methodologies and original object-oriented methodologies prevents from evolution of agent-oriented methodologies.

In MAS, multi-agent architecture plays important role in defining relationships and collaborations among agents [3]. Actually, most of MAS have been suggested multi-agent architecture, but it was just implementation-oriented, focused on internals of a single agent, or was provided with informal diagrams [3, 14]. In particular, as the size and complexity of systems increase, the main design problem is specifying overall system structure rather than the algorithms and data structure [8]. Even though multi-agent architecture is important for the MAS, current agent-oriented methodologies do not provide models and modeling elements to analysis, design, and evaluate it.

This paper suggests an architecture-centric object-oriented design method for MAS. It is based on considering MAS as a software system [23]. It uses object-oriented methods as a core development methodology, and extends them to properly represent collaborations among agents. The extensions are based on design principles that are derived from characteristics of MAS and concepts of software architecture. They have roles that are kinds of syntactic or semantic constraints to restrict on models or modeling elements and are formalized. Concepts of software architecture, which software system consists of software components and their connectors [8], are introduced to properly represent multi-agent architecture. These formalized restrictions contribute to accuracy and verification of models in principled-way.

The suggested method uses semantics and notations of the UML (Unified Modeling Language) that has adopted as an industry standard object-oriented design method by OMG (Object Management Group) [2, 19]. The use of standard design methods has several benefits such as availability of developers and tools, utilization of past experience and knowledge, and more direct comparison and evaluation than special-purpose notations. In addition, the UML has the extension mechanism without semantic and syntactic change of original models such as constraints, stereotypes, and tagged values [19]. It implies that the availability of developers and tools, the utilization of past experiences and knowledge, and the seamless integration of other systems. The suggested method is illustrated with a real case [21].

2. Background

As a result of the popularity and complexity of agent-based systems, agent-oriented methodologies (AOM) and modeling techniques have been suggested in last few years [4, 12, 15, 16, 24]. They take object-oriented methodologies (OOM) as their basis, extend the OOM to cover limits of the OOM, and suggest new kinds of AOM. A survey of current agent-oriented methodologies is found in [11].

There are similarities among AOM in concept of differences between agents and objects. Agents do not just of attributes and methods, but also have mental state and concepts such as plans or goals. Based on OMT [20], the approach of Kinny et. al [16] is the most strong extension case of AOM, but it restricts the scope on domain of methodology to BDI (Belief, Desire, and Intention) agents.

Another difference is the type of communication between agents and objects. Agents communicate with each other by structured or meaningful messages and use protocols to collaborate, while message passing for objects is just method invocation

or simple data passing. Behavior models of AOM approaches depend on their basis model. Approaches that their basis are classical OOM such as the OMT [20], the Booch Method [1], and Object-Oriented Software Engineering (OOSE) [13] use behavior model of the OOM with slight extension of modeling elements. There is an approach to propose the combination of the OOM and enterprise modeling methodology IDEF (Integration Definition for Function modeling) for agent system modeling [15]. The classical OOM, however, take concept of functional decomposition and are poor to represent collaboration among agents [5]. For example, the event flow model of the OMT for the behavior among objects is lack in concepts and constructs such as communication of instance level, message sequence, concurrency, and dependency among objects in terms of dynamic state. The functional model of the OMT and the IDEF0 (functional model) clash with object structure of the implementation.

The AOM have own special purpose notations and techniques. So, their extensions cause the semantic and syntactic gaps between them and original OOM. In addition, approaches of the AOM are based on the classical OOM, while the classical OOM have been evolved. For example, the UML (Unified Modeling Language) have been suggested to cover limits of the classical OOM. This implies lacks of developers and tools for the development of MAS.

All these AOM support the development of MAS in limited value without considering multi-agent architecture. There are two kinds of approaches to research agent architecture: internals of a single agent and multi-agent architecture. Only the AOM approach of Kinny et.al [16] reflects agent architecture to logical model, but the scope is restricted on single agent architecture. In MAS, multi-agent architecture plays important role in defining relationships and collaborations among agents [3]. Actually, most of MAS have been suggested multi-agent architecture, but it was just implementation-oriented, focused on internals of single agent, or was provided with informal diagram [3, 21]. In particular, as the size and complexity of systems increase, the design problem is specifying overall system structure rather than the detail algorithms and data structure [8]. Even though multi-agent architecture is important for the MAS, current AOM do not provide models and modeling elements to analysis, design, and evaluate it.

Concept of software architecture supports to properly represent multi-agent architecture. Software architecture description is a high-level model of software systems. It is recognized as a collection of computational components together with a description of the interactions between these components – the connectors such as

procedure call, event broadcast, database query, and pipes [8]. An architectural style defines a family of such systems in terms of a pattern of structural organization. It determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined. In MAS, it provides clean separation between individual agents and their interactions in overall systems. The separation is natural for the description of multi-agent architecture, because agents should communicate with each other without dependency on other agent. It is called connection problem in MAS [6].

Architectural styles for the multi-agent architecture have been already researched such as blackboard system, agency, and facilitator [3, 24]. Although current AOM do not reflect multi-agent architecture, for analysis and design of multi-agent architecture, there are ADL (Architecture Description Languages) [18] and techniques such as design pattern [7]. The ADL, however, are inappropriate to adopt as a development method, because they focus on only architecture description without other development artifacts such as development models and activities [18]. Design pattern provides a useful technique for the design of multi-agent architectural style. According to research of Gamma et. al [7], design patterns are defined as descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. Even though concepts of software architecture and design pattern support to effectively develop MAS, there is no AOM to reflect them.

Although a development methodology need to include a development process that describes activities, results, and guidelines along development phases, the development process depends on development environments such as scale of projects, experience and knowledge level of the development teams [2, 19]. This paper focuses on notations and techniques for the design of MAS and development process is beyond the scope of this paper.

3. Conceptual Framework

3.1 Overview of Proposed Framework

The suggested design method is an integrated method from existing technologies rather than another new design method. These existing technologies include object-oriented methodology, design patterns, and software architecture in software engineering. Although artificial intelligence and knowledge engineering have

influenced to knowledge processing in singular agent or among agents, the development of MAS as a software system should utilize experience and knowledge from past decades in software engineering [24]. It is intended that the suggested method is appropriate for development of large-scale multi-agent systems that necessarily require design methods for the effectiveness such as the productivity and quality of the system development.

The UML is adopted as a core design notation. Main reasons of using the UML are that it has integrated progresses from past object-oriented methodologies, that it has extension mechanism underlying original semantics of the method, and that it has been adopted industry standard object-oriented design methods in the OMG (Object Management Group) [2, 19]. The use of standard design methods has several benefits such as availability of developers and tools, utilization of past experience and knowledge, and more direct comparison and evaluation than special-purpose notations.

The extensions of the UML are based on design principles that are derived from agent characteristics and concepts of software architecture. They have roles that are kinds of syntactic or semantic constraints to restrict on model or modeling elements. Agent characteristics are consolidated on structure and behavior of multi-agents rather than those of single agent because current object-oriented technologies provide sufficient techniques to consider agent as a primitive building block. For example, an agent can be manipulated as a single class in logical model by abstraction mechanism such as aggregation. In addition, behavior abstraction of an agent with interfaces makes logical models are free from internal implementation of agents.

Concept of software architecture, which software system consists of software components and their connectors [8], is utilized to properly represent multi-agent architecture. It provides clean separation between individual agents and their interactions in overall systems. The separation makes large applications more tractable, global analysis feasible, and software reusable.

Design principles are extended using the extension mechanism of the UML and formalized by the OCL (Object Constraints Language) of the UML. The extension mechanism guarantees to keep original syntax and semantics of model and modeling elements. This implies seamless integration of legacy object-oriented system, reusability of past experience and knowledge, and availability of developers and tools. The formalizations contribute to accuracy and verification of models in principled-way.

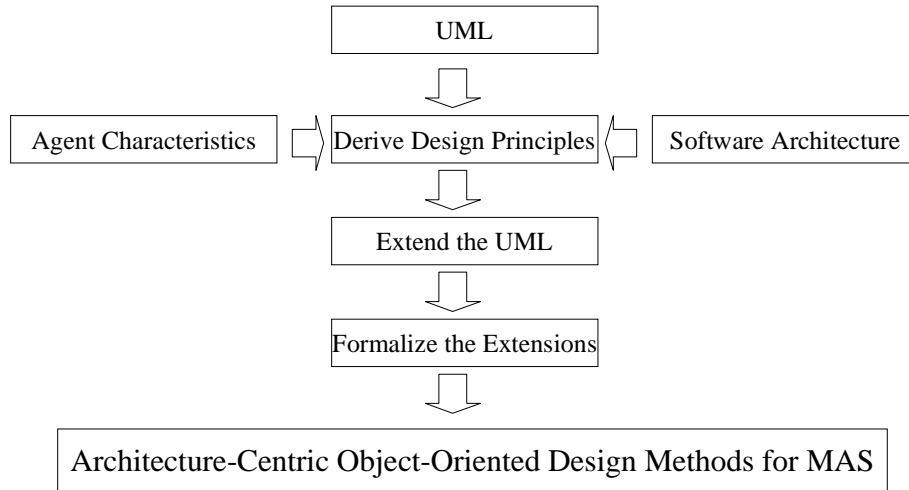


Figure 1. Conceptual Frameworks

3.2 The UML As a Core Design Methods and Its Extension Mechanisms

The UML is an object-oriented graphical language with well-defined syntax and semantics. The syntax and semantics are specified semi-formally through underlying semantic models (meta-model), descriptive text, and constraints. The UML consists of nine diagrams and each diagram represents various aspects to develop a target system: class diagram, object diagram, use case diagram, statechart diagram, sequence diagram, collaboration diagram, activity diagram, component diagram, and deployment diagram.

The UML is an extensible language so that new constructs may be added to address new issues in software development without changing the original syntax and semantics. (1) Constraints place semantic restrictions on particular design elements. (2) Tagged values allow new attributes to be added to particular elements of the model. (3) Stereotypes allow to add new elements representing a subclass of an existing element. These mechanisms may be written by the OCL that is a formal language to express constraints based on first-order predicate logic.

Since, basically, the UML covers most of aspects to design MAS as a software system, the suggested method extends the UML to satisfy additional requirements for

MAS in principled way. Design principles are derived from agent characteristics and software architecture and formalized to verify models whether or not the model user specified satisfies the principles.

3.3 Deriving Design Principles

Multi-agent system design encourages the distribution of behavior among agents. Such distribution can result in agent structure with many connections between agents. Lots of interconnections make it less likely that an agent can work without the support of others, while one of agent characteristics is that an agent has capability to follow its goal autonomously. In fact, as the number of agents is increasing, it can not be avoided that the more dependencies among agents exist. In classical object-oriented modeling techniques, objects should have references to communicate with each other, while agents communicate with other agent in keeping independence. Concept of software architecture, which software systems consist of components and connectors, supports the design to minimize the interconnection among agents. Introduction of connectors reduces structural dependencies between agents in structural model. This leads to the following principle of architecture-centric object-oriented design:

Multi-agent structure consists of agents having roles of components and connectors.

Although the internal structure of an agent consists of functional components and also a number of behavior patterns, agent is a primitive building for the design of MAS. In fact, an agent has more similarity with a subsystem of classical object-oriented methods than a single object. It, however, is only of limited use to structure agent-oriented systems in the form of subsystems. Rather, the modeling should be performed taking into consideration an agent as an abstracted agent class within overall systems, because an agent can be directly designed by decomposing without re-structuring in object-oriented model. This leads to second principle of architecture-centric object-oriented design:

An agent is a primitive building block for the design of multi-agent systems.

This abstracted modeling technique accommodates reuse. One of the strengths in object-oriented techniques is the inheritance mechanism to reuse or abstract

components, but it discourages to use this mechanism in MAS. Agent normally has a very special form and also the existence of knowledge based components. It is a reason that current agent-oriented methodologies do not permit or discourage to use this mechanism. An architecture-centric design, however, provides a generic system structure and behavior patterns. This implies that once successfully designed architecture can be reused in the same specific domain. The following principle is derived to accommodate reuse in the design of MAS:

Architecture-centric design favors pattern-based mechanism over inheritance.

All these above principles support to effectively design MAS, but they are not enough to design complete MAS. They are just principles at least for the design. There are more general principles in development methodology community [2]. They can be summarized following as: it is important what kinds of models are selected, model should be expressed at different levels, the best models are connected to reality, and no single model is sufficient.

In this paper, we provide the extensions of the UML based on these principles and then, the extensions are formalized by OCL. The formalizations have roles that are syntactic or semantic constraints to restrict on model or modeling elements.

4. CASE Study

4.1 Overview of Case

As an illustrative example, the visitor hosting system (VHS) example is used in this paper [21]. It had been developed based on reusable multi-agent computational architecture RETSINA (Reusable, Task Structure-based Intelligent Network Agent). It has three types of agents. Interface agents interact with users. Task agents help users perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other agents. Information agents provide intelligent access to a heterogeneous collection of information sources.

The RETSINA architecture was suggested in the form of figure 2. This form is typical to suggest agent architecture in agent-based applications, but it is difficult to represent the architecture in traditional object-oriented model. Figure 3 shows the problems when the architecture is designed by traditional object-oriented model. As the number of agent increases, dependency among agent is to be profound. In

traditional object-oriented techniques, agents have references to each other to communicate with them in natural way. At worst case, agents know all other agent. To resolve these problems, we have suggested architecture centric design methods to constrain modeling.

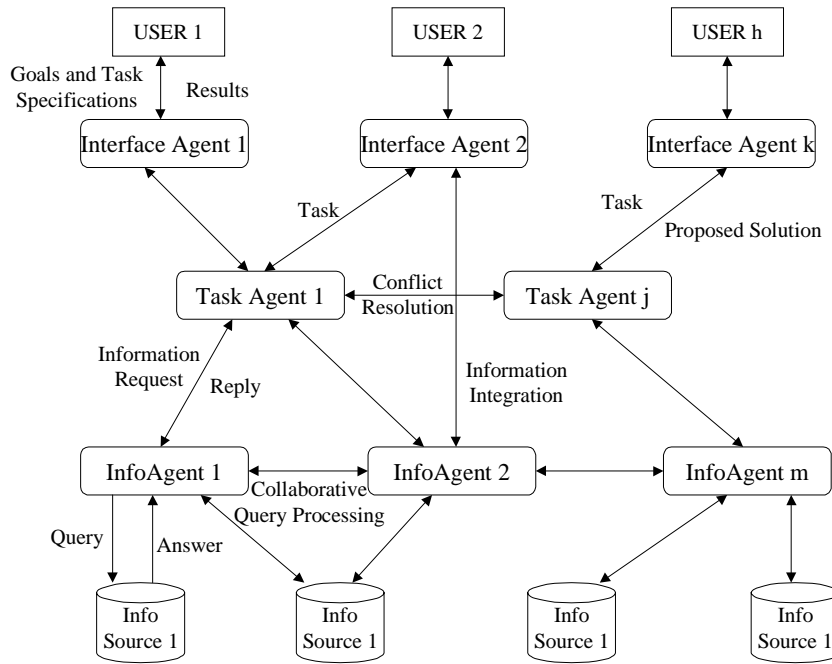


Figure 2. Agent Architecture in RETSINA [21]

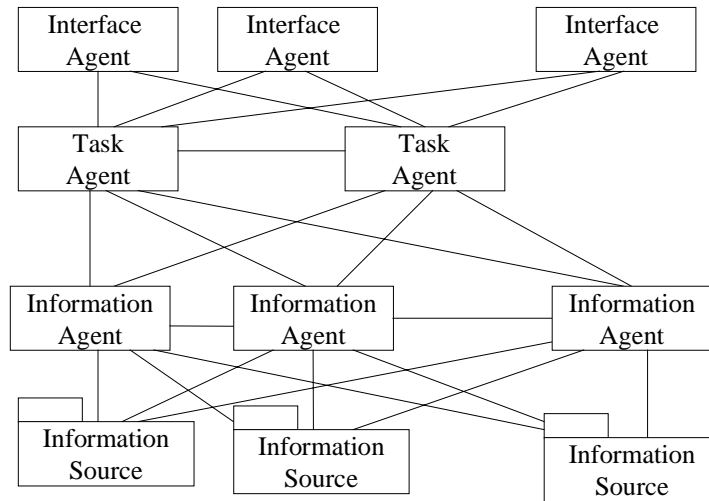


Figure 3. Traditional Object-Oriented Model of the RETSINA Architecture

4.2 The Visitor Hosting System

The VHS is a system that is to arrange the visitor's schedule with faculty whose research interests match the interests that the visitor has expressed in his/her visit request. The VHS has an interface agent, referred to as the Visitor Hoster, which interacts with the person hosting the visit. It also has the following task agents: a personnel finder task agent, who finds detailed information about the visitor, and also finds the faculty he/she meets, (2) the visitor's scheduling task agent and (3) various personal calendar management task agents that manage calendars of various faculty members. In addition, the VHS has a number of information agents that (1) retrieve information from a CMU database that has faculty research interests (Interests agent), and (2) retrieve personnel and location information from various university databases. For the brevity, it is assumed that agents have been already identified and the KQML is used for the communication among agents

4.3 Architecture-centric Extensions for MAS

In order that agents are primitive building blocks for the design of MAS,

behaviors of agents and their internal objects are abstracted to interfaces and the agent has operations corresponding to the interfaces.

Stereotype ArMessage for instance of meta-class Message

[1]ArMessages are tagged identifying protocol types

arMsgType : enum {advertise, unadvertise, ask, ask-all, reply, ... }

Stereotype ArOperation for instance of meta-class Operation

[1]ArOperations are tagged for identifying corresponding ArMessages

arOprType : enum {advertise, unadvertise, ask, ask-all, reply, recruit, ... }

[2]ArOperations have no return values

self.parameter → not exists(p / p.kind = return)

Stereotype ArInterface for instance of meta-class Interface

[1]All ArInterface operation corresponds to stereotype ArOperation.

self.oclType.operation → forall (o / o.stereotype = ArOperation)

Figure 4. Interface Restrictions

Modeling elements for design of MAS consist of components and connectors. Agents are components that are abstracted to ignore their internal details. Agents have state values for representing mental state. In fact, agents do not reply until their mental state reach *agreement* state although they take request message.

Stereotype ArAgent is instance of meta-class Class

[1]Mental state of agent has a tagged value either agreement or disagreement

ArMtlState : enum { agreement, disagreement }

[2]ArAgent has ArOperations to recognize all of communication protocols.

self.oclType.operation → exists (o / o.arOprType = advertise) and

self.oclType.operation → exists (o / o.arOprType = unadvertise) and

self.oclType.operation → exists (o / o.arOprType = ask) and

self.oclType.operation → exists (o / o.arOprType = ask_all) and

self.oclType.operation → exists (o / o.arOprType = reply)

[3]ArAgent is associated to its internals with composition relationship.

Let wholes = self.oclType.assocEnd → select (a / a.aggregation = composite),

wholes.association.oclType → select (o / o.ocllsKindOf (class)) and select (o /

o.stereotype <> ArAgent) and select (o / o.stereotype <> ArConnector)

Figure 5. Component Restrictions

In architecture-centric design, connector plays a role that is a bridge between agents. In this approach, agents communicate with each other maintaining independence through connectors. In fact, a connector is an agent, but it abstracts interactions among other agent.

Stereotype ArConnector is instance of meta-class Class
[1 – 2] Same as constraints 1-2 on ArAgent

Figure 6. Connector Restrictions

In architecture-centric design, there is only one kind of relationships between components and connectors. An agent is not associated with other agents except connectors. It leads a topology rule that agent can not directly connect to other agent without connector. This fact guarantees independence of agents

Stereotype ArAttachment is instance of meta-class Association.
[1]ArAttachment is binary association
self.oclType.assocEnd → size = 2
[2]The first end of the association must be to an ArAgent
self.oclType.assocEnd[1].class.stereotype = ArAgent
[3]The second end of the association must be to an ArConnector
self.oclType.assocEnd[1].class.stereotype = ArConnector
[4]Multiplicity of ArConnector that participate ArAttachment is at minimum one and at maximum one
self.oclType.role → forall (r / r.multiplicity = “1..1”)
[5]Multiplicity of ArAgent that participate ArAttachment is at minimum one and at maximum one
self.oclType.role → forall (r / r.multiplicity = “1..1”)

Figure 7. Relationship Definition

Architecture-centric design means that all of components are organized in

structured way. As described in previous section, only few kinds of model elements are used for architectural model. This restriction of constructs makes designers concentrate on narrow scope and models maintain consistency. All agents know at least one connector to communicate other agents. This means that all agents participate in one or more collaborations.

Stereotype ArModel is a instance of meta-class Model

[1]ArModel contains architectural components

self.oclType.modelElement \rightarrow forall (e / e.stereotype = ArAgent or e.stereotype = ArConnector or e.stereotype = ArAttachment or e.stereotype = ArInterface or e.stereotype = ArOperaton or e.stereotype = ArMessage)

[2]Each ArAgent must participate at least one ArAttachment

Let cls = self.oclType.modelElement \rightarrow select (e.stereotype = ArAgent)

*cls \rightarrow forall (c / c.assocEnd.association
 \rightarrow select (a / a.stereotype = ArAttachment) \rightarrow size ≥ 1)*

[3]Each ArConnector must participate at least one ArAttachment

Let cls = self.oclType.modelElement \rightarrow select (e.stereotype = ArConnector)

*cls \rightarrow forall (c / c.assocEnd.association
 \rightarrow select (a / a.stereotype = ArAttachment) \rightarrow size ≥ 1)*

Figure 8. Model Elements Restrictions

4.4 Selective Graphical Diagrams

Architecture-centric design focuses on artifacts to describe architecture of systems in certain level of abstraction. In the UML, use case model can be used for establishing the desired architecture of the systems like figure 9. Part (a) shows the model for the RETSINA architecture. It mainly consists of collaborations, which their shapes are dotted ellipse, rather than use cases, since the RETSINA architecture is a kind of generic architecture and a basis for the design of the VHS. Part (b) shows use case diagram for the VHS. The use case diagram reflects that the VHS should be developed under the RETSINA architecture. The collaborations that are defined for the RETSINA architecture are used for realizing use cases of the VHS.

In the RETSINA architecture, collaborations for representing interactions among agents are *perform task*, *integrate information*, *conflict resolution*, and *collaborative query*. In fact, these collaborations require more detailed design, but, for the brevity,

it is assumed that they are realized by mediator pattern [7]. Figure 10 depicts a model of agent collaboration using mediator pattern. Structural aspect of mediator collaboration is shown in part (b) of figure 10. The mediator pattern among design patterns is the best match to the case that facilitator is used for communication between agents. It is used in following cases: a set of objects communicates in well-defined but complex way. The resulting interdependencies are unstructured and difficult to understand. Reusing an object is difficult because it refers to and communicates with many other objects. A behavior that's distributed between several classes should be customizable without a lot of subclassing.

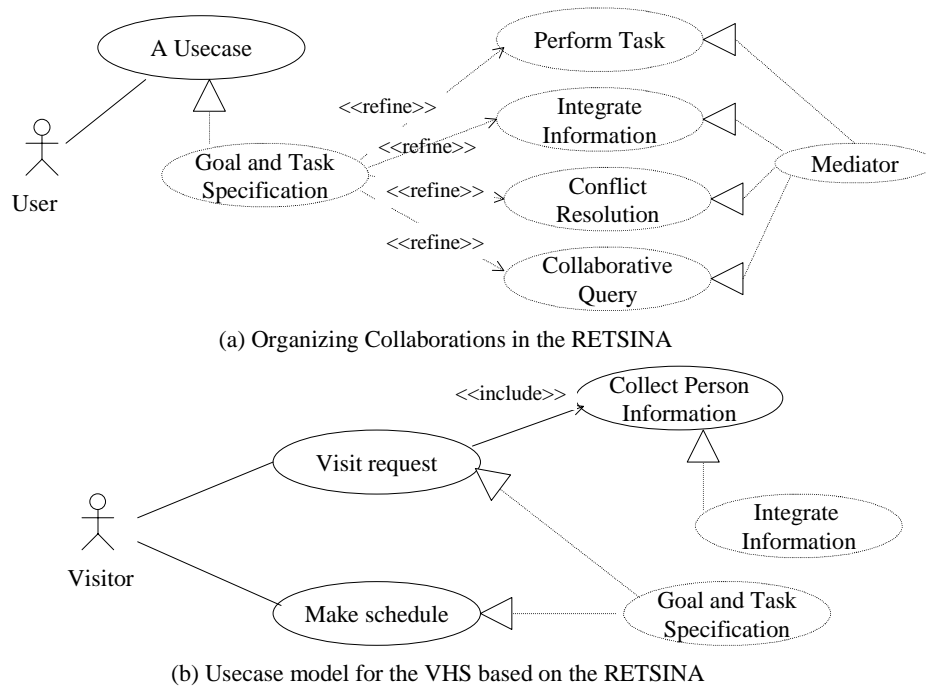
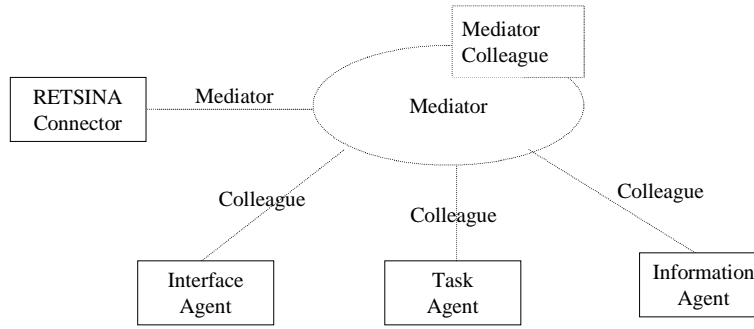
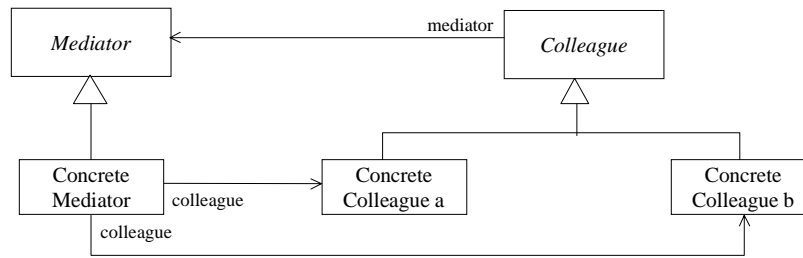


Figure 9. Use case model for the RETISNA and the VHS

Architecture-centric design favors pattern-based mechanism over inheritance for reuse. Figure 11 shows the realization of pattern modeling in part (a) of figure 10. This approach uses substitution of parameter rather than inheritance for reuse. Components for the VHS are used for parameters of mediator pattern. In the figure, Components including RETISNA connector, interface agent, task agent, and information agents substitute for colleague and mediator parameters. As results of this approach, figure 11 shows the structural aspects and behavioral aspects of the *integrate information* collaboration in the RETSINA architecture.



(a) Agent Collaboration in the RETSINA Architecture

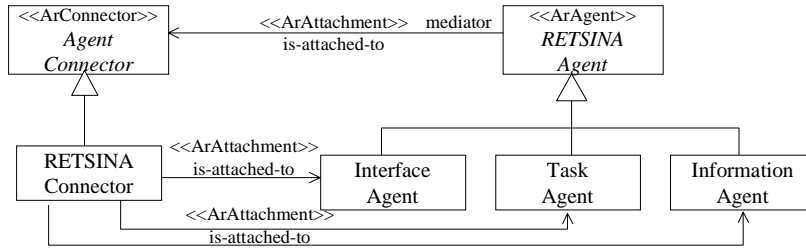


(b) Class Diagram for Mediator Collaboration (Design Pattern)

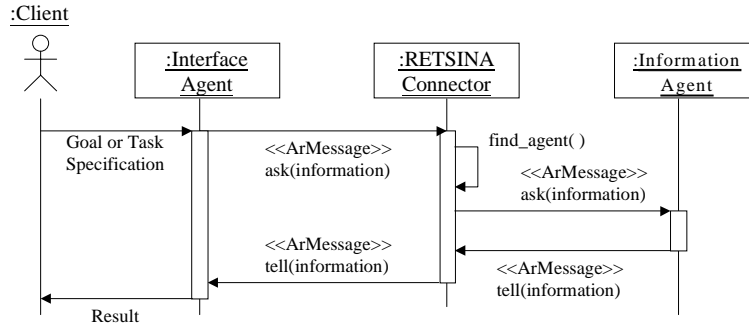
Figure 10. Pattern Modeling

In models of figure 11, all of classes are *ArAgents* that are stereotype classes for agents. The *RETSINA connector* is introduced to simplify relationships among agents. It is an agent having role of facilitator, but has more semantics than traditional facilitator by the connector restrictions. It does not resolve the connection problems among scattered agents in distributed network, but also abstracts collaborations among agents. It encapsulates how a set of agents interacts. It promotes loose coupling by keeping agents from referring to each other explicitly, and it makes designers model their interaction independently. All of relationships are *ArAttachments* that are stereotype relationships for relationship between agents. They derive topological rule that agents are not directly connected to each other without a connector.

In models of figure 11, the *RETSINA connector* agent is a mediator. The *RETSINA agents* send and receive requests from the *RETSINA connector* agents. The *RETSINA connector* agent implements the cooperative behavior by routing requests between appropriate the *RETSINA agents*. Messages between agents are represented by stereotype *ArMessages*. The messages have performatives to inform protocols between agents.



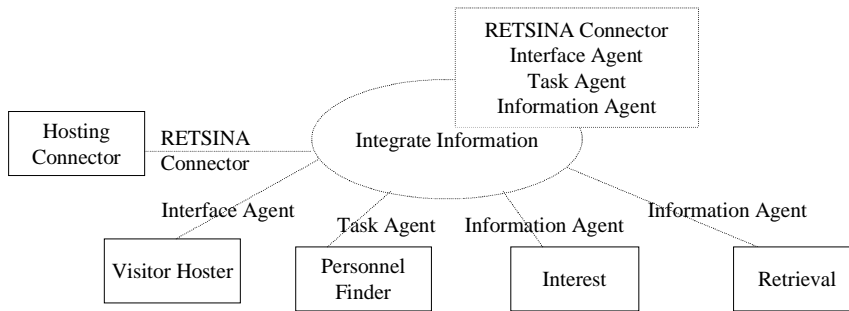
(a) Class Diagram for Integrate Information Collaboration in the RETSINA Architecture



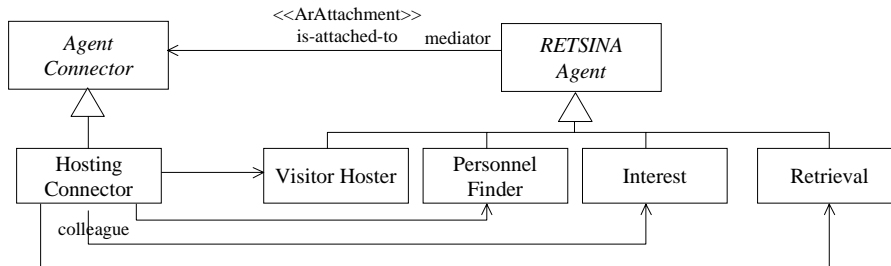
(a) Sequence Diagram for Integrate Information Collaboration in the RETSINA Architecture

Figure 11. Two Aspects for the Integrate Information Collaboration

Figure 12 and 13 show the model for the VHS based on the RETSINA models. The VHS directly reuses the structure and behavior of the RETISNA without restructuring. This means well-defined architecture is re-usable for the same domain.



(a) Integrate Information Collaboration for the VHS



(b) Class Diagram for Integrate Information Collaboration

Figure 12. The VHS modeling based on the RETSINA model

Figure 13 shows collaboration diagrams for behavioral aspects of use cases of the VHS. Figure 14 shows an activity diagram for the collaboration between a connector and agents. This diagram is used for modeling a detail algorithm for collaboration protocols. In figure 14, it is assumed that the connector has pre-defined references of agents for the collaboration instead of searching appropriate agents.

In this case study, all of selective diagrams use extended modeling elements that are defined based on the suggested design principles. They satisfy syntactic or semantic constraints to properly design the case. Although detailed design is required for implementations of the case, selective diagrams show sufficient logical models to represent the case. This implies that the suggested design principles help to build more reusable and well-structured systems for large and complex MAS.

In the UML, there are other diagrams: statechart, component, and deployment diagram. These diagrams may contribute to model in certain aspects of MAS, but they are omitted because this paper focuses on logical model to design of MAS.

7. Conclusion and Further Researches

Agent technology is used in wide variety of applications with range from comparatively small systems to large, complex, and mission critical systems. While the popularity of agent-based system is increasing in trends, practical development support of methodology has fallen behind the trend of increasing popularity.

Based on classical object-oriented methodologies, agent-oriented methodologies have been suggested to reflect characteristics of agents or multi-agent systems, but they are just immature or research-oriented. In particular, multi-agent architecture plays a role in representing relationship and collaboration among agents in multi-agent systems. Actually, most of multi-agent systems have been suggested multi-agent architecture to describe relationship and collaboration among agents, but current agent-oriented methodologies do not have model and modeling elements to analysis, design and evaluate the multi-agent architecture.

In this paper, an architecture-centric object-oriented design method for multi-agent systems is suggested. The suggested method adopts object-oriented method as a core design method and then extends it to properly represent collaboration among agents. The suggested method provides design principles that are derived from agent characteristics and concepts of software architecture. Concept of software architecture, which software systems consist of components and connectors, is introduced to properly represent multi-agent architecture. It provides clean separation between individual

agents and their interactions in overall agent systems. The separation makes large applications more tractable, global analysis feasible, and software reusable.

The suggested method adopts the UML as a core design method and extends it by extension mechanism of the UML without syntactic and semantic changes of original models. It implies that the availability of developers and tools, the utilization of past experiences and knowledge, and the seamless integration of other systems. The extension is based on design principles and formalized by OCL (Object Constraint Language). These extensions have roles that are kinds of syntactic or semantic constraints to restrict on model or modeling elements. These formalized restrictions contribute to accuracy and verification of models in principled way.

This approach will improve the productivity and quality of MAS development by exploiting existing software-engineering progresses. In future, we will employ the methods in the real applications. The feedback from these applications will refine the methods.

References

- [1] Booch, G., *Object-Oriented Analysis and Design with Application*, Benjamin/Cummings, 1994.
- [2] Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
- [3] Brenner, W., Zarnekow, R., and Wittig, H., *Intelligent Software Agents: Foundations and Applications*, Springer-Verlag, 1998.
- [4] Burmeister, B., "Models and Methodology for Agent-Oriented Analysis and Design", K. Fischer, editor, *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems*, Germany, 1996.
- [5] Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, H., and Jeremaes, P., *Object-Oriented Development: The FUSION Method*, Prentice Hall, 1994.
- [6] Decker, K., Sycara, K. and Williamson, M., "Matchmaking and Brokering", *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, December 1996.
- [7] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [8] Garlan, D. and Shaw, M., "An Introduction to Software Architecture", *Software Engineering and Knowledge Engineering*, Vol. I, World Science Publishing, 1993.
- [9] Foundation for Intelligent Physical Agents, "FIPA Architectural Overview", *FIPA '99 Specification*, July 1999. Available from <http://www.fipa.org>.
- [10] Finin, T. and Wiederhold, G., "An overview of KQML: A Knowledge Query and Manipulation Language", Dept. of Computer Science, Stanford Univ., 1991.
- [11] Iglesias, C.A., Garijo, M., and Gonzalez, J.C., "A Survey of Agent-Oriented Methodologies", *ATAL'98*, Paris, France, 1998, pp. 185 – 198.
- [12] Iglesias, C.A., Garijo, M., Gonzalez, J.C., and Velasco, J.R., "Analysis and Design of Multi-Agent Systems Using MAS-CommonKADS", M.P Singh, A.Rao, M.J. Wooldridge, editors, *Intelligent Agent IV(ATAL'97)*, LNAI 1365, Springer-Verlag, Berlin, Germany, 1998, pp. 314 – 327.
- [13] Jacobson, I., M. Christerson, P. Jonsson, G. Overgaard, *Object-Oriented Software Engineering. A Use Case Driven Approach*, Addison-Wesley, 1992.
- [14] Jennings, N. R., Sycara, K., and Wooldridge, M., "A Roadmap of Agent Research and Development", *Journal of Autonomous Agents and Multi-Agent Systems*,

1998, vol.1, pp. 275-306

- [15] Kendall, E. A., M. Malkoun, and C. H. Jiang, "A Methodology for Developing Agent Based Systems for Enterprise Integration", *Modelling and Methodologies for Enterprise Integration*, Chapman and Hall, P. Bernus and L. Nemes, Editors, 1996.
- [16] Kinny, D., Georgeff, M., and Rao, A., "A Methodology and Modeling Technique for Systems of BDI Agents", *LNAI 1038*, Springer-Verlag, Berlin, Germany, 1996, pp. 56-71
- [17] Robbins, J.E., Medvidovic, N., Redmiles, D.F., and Rosenblum, D.S., "Integrating Architecture Description Languages with a Standard Design Method", *Proceedings of the 1998 International Conference on Software Engineering*, Kyoto, Japan, 1998, pp. 209 – 218
- [18] Taylor, R. N., Medvidovic, N., Anderson, K., Whitehead, Jr., E.J., Robbins. J.E., Nies, K. A., Oreizy, P., and Dubrow, D.L., "A Component and Message-based Architectural Style for GUI Software", *IEEE Transaction on Software Engineering*, June 1996, Vol. 22, No. 6, pp. 390-406.
- [19] Rational Partners, Unified Modeling Language Documents, Version 1.1, Rational Software Corporation, September 1997, Available from <http://www.rational.com>.
- [20] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling & Design*, Prentice-Hall, 1991
- [21] Sycara, K., Decker, K., Pannu, A., Williamson, M. and Zeng, D., "Distributed Intelligent Agents", *IEEE Expert*, December 1996.
- [22] Winograd, T. (1987), "A language/action perspective on the design of cooperative work," *Human-Computer Interaction* 3:1 (1987-88), 3-30.
- [23] Wooldridge, M., "Agent-based Software Engineering", *IEE Proceedings on Software Engineering*, 144(1), pp. 26 - 37, February 1997.
- [24] Wooldridge, M., Jennings, N.R., Kinny, D., "A Methodology for Agent-Oriented Analysis and Design", In Etzioni, O., Muller, J.P., and Bradshaw, J., editors: *Agents '99: Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, May 1998.