Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp313-318)

ADAM: Architecture-Driven Multi-Agent Systems Development Methods

HONG SOON YIM[1], HABIN LEE[2], SUNG JOO PARK[1]
[1] Graduate School of Management
Korea Advanced Institute of Science and Technology
207-43 Cheongryangri2-dong, Dongdaemoon-gu, Seoul, 130-722
KOREA
[2]Intelligent Systems Research & Innovation Centre
British Telecommunications plc.
PP12/MLB1 B62 Adastral Park, Ipswich, IP5 3RE,
UNITED KINGDOM

Abstract: This paper introduces ADAM: an Architecture-Driven multi-Agent systems development Method. ADAM addresses structural issues such as the structuring of domains, the agent's organization with roles, the agent's interaction with control mechanisms, and the reusability of the model. These issues help in the design of reusable and well-structured MAS based on multi-agent architecture. ADAM extends UML (Unified Modeling Language) to support a set of concepts specific to MAS, such as loosely coupled agent organization and protocol-based agent interaction, and also the formal semantics of extensions. The extension allows one to use the original object-oriented method for ADAM without syntactic or semantic changes.

Key-Words: - multi-agent systems, architecture, object-oriented development methods, interaction pattern, UML, agent oriented software engineering

1 Introduction

As the number of agents increases in a Multi-agetn system (MAS), the system becomes more complex with increased interactions between agents within the system. The interaction or dependency does not only make the analysis difficult, but also becomes an obstacle to the reusability, extensibility, and maintainability of the system. Given this, the main design problem is specifying an overall agent structure rather than the properties and capabilities of individual agents [4]. The structural issues of the MAS include structuring a group of agents, role-based agent organization, control mechanisms for agent collaborations, and model transferability across other systems. Addressing these issues help one to construct a well-defined agent structure that improves reusability, extensibility, and maintainability, and also increases analyzability of the system.

This paper details ADAM (Architecture Driven Multi-agent Methodology) as a MAS development methodology wherein interaction pattern is a key concept through all sub phases of it. For this, software architecture concept is used as a means to model interaction patterns. The methodology shows how interaction patterns can be modeled using architecture concept in analysis and design phases. This paper is composed as follows: in section 2, related works are reviewed. Section 3 gives an overview of the main concepts used in ADAM. Section 4 gives illustrative examples, and the contribution of the paper is summarized in section 5.

2 Literature Review

Software architecture description is a high-level model of software systems with a collection of computational components and their interactions which are the connectors such as procedure calls, event broadcasts, database queries and pipes. The advantages of software architecture lay within providing mutual communications, early design decisions, and transferable abstractions of a system [4]. In addition, it provides a clean separation between components and their interactions in the system. The separation makes complex systems more tractable, analyzable, and reusable [11].

For the design of an architectural model of MAS, ADL (Architecture Description Languages) and techniques such as a design pattern can be used. The ADL, however, focuses only on architectural description without mentioning development artifacts such as the modeling of the system development [4, 11]. An exception is the work of

Robbins et al. [11] which integrates the ADL with the development methodology, UML.

The research on agent architecture can be classified into two categories: internal architecture of a single agent and the architecture of a multi-agent system. Kinny et al. [9] considers a logical model, BDI (Belief, Desire, and Intention), as an architecture of a single agent. In MAS, multi-agent architecture plays an important role in defining relationships and collaboration among agents [1]. The multi-agent architecture, as used in most MAS, is outlined in informal diagrams [1, 13] which focus on implementation rather than the analysis, design, and evaluation of the architecture.

Due to the similarities between an object and an agent, most of the current AOMs use the notations and techniques of classical OOMs (Object-Oriented Methodologies) with a slight extension of modeling elements [2,6,7,9]. The classical OOM include the OMT (Object Modeling Technique) [12] and the OOSE (Object-Oriented Software Engineering) [7]. The agent not only has attributes and methods, but also a mental state and concepts such as a plan, a goal, and an intention. It communicates with other agents by structured or meaningful messages and uses protocols to collaborate, while messages between objects are passed simply for the purposes of method invocation in the normal object-oriented approach [5].

Burmeister [2] and Kinny et al. [9] extended the OMT. Kendall et al. combined the OOSE and IDEF (Integration DEfinition for Function modeling) for agent system modeling. Iglesias et al. [6] suggested the MAS-CommonKADS that is an integrated AOM of the OMT and the CommonKADS, a methodology for the development of knowledge-based systems.

Since the extensions fail to adequately capture the characteristics of MAS such as the agent's autonomous behavior and the complexity of the organizational structure, the Gaia [15] and the MaSE (Multi-agent Systems Engineering) [3] methodologies proposed their own special notations and semantics for agent-oriented systems. Recently, an agent-specific extension of UML, the AUML (Agent Unified Modeling Language), has been suggested [10]. It focuses on the representation of the agent's behavior based on agent interaction protocol and introduces techniques for representing the characteristics of MAS using UML diagrams, such as the behavior of the agent role and physical distribution.

3 ADAM (Architecture-Driven multi-Agent systems development Method)

ADAM models can be classified into two categories of layers, i.e. the architecture and application layers. The generic models of ADAM reside in the architecture layer. The models in the application layer can be considered as instantiations of the models in the architecture layer. The generic models of ADAM include the models for problem structure, the agent organization, the agent interactions and the control states which extend UML. In particular, the semantics of ADAM modeling constraints and elements are represented by the OCL (Object Constraints Language) of UML. The semantics are restricted by constraints, tagged values, and stereotypes. Constraints place restrictions on design elements. Tagged values allow new attributes to be added to particular elements of the model. Stereotypes allow the addition of new elements representing a subclass of an existing element. Table 1 shows the overall model and modeling elements of ADAM.

3.1 The Problem Structure Model

The Problem Structure model represents how the overall problem is divided into sub- problems and how they are related to each other. A sub-problem is represented by a MAA (Multi-Agent Architecture). The Problem Structure model consists of a set of use cases and MAAs represented by collaboration that is a modeling element to describe a general arrangement of classes that interact within a context to implement a behavior such as use case or operation [1]. There are semantic restrictions in the problem structure model as follows:

The Stereotype *MAA* is an instance of meta-class *Collaboration*. (1) Parameters of MAA are ArRole and ArCollaboration. (2) Parameters of MAA should have one or more ArRole. (3) Parameters of MAA should have one or more ArCollaboration.

The Stereotype *Problem Structure model* is an instance of a meta-class *Model*. (1) Problem Structure model is tagged for identifying corresponding abstraction level. (2) The Problem Structure model contains the stereotyped collaboration, MAA.

The MAA consists of three sub models: the agent organization, the agent interaction, and the control state models.

Table 1. Abstracts of ADAM model and modeling elements

Models	Diagrams	Stereotyped Modeling Elements		Semantic Restrictions
		Architecture View	Application View	
Problem Structure	Use Case	MAA	Use Case	MAA, Model
Agent Organization	Class	ArRole, ArCollaboration, ArParticipate	ArAgent, ArRole ArCollaboration, ArParticipate, ArPlay	Interface, Component, Connector, Relationship, Model
Agent Interaction	Sequence or Collaboration	ArRole, ArCollaboration, ArMessage	ArAgent, ArCollaboration, ArMessage	Message, Model
Control State	State-Transition	ArMessage	ArMessage	Model

3.2 The Agent Organization Model

The Agent Organization model captures architectural aspect of the MAA or use case using a class diagram. For representing agent's organization, the Agent Organization model introduces a collaboration class as a connector class that connects between components such as agent or role classes. The collaboration class controls interactions among participating role classes according to the interaction protocol employed in the MAA. An agent role defines the responsibility of an agent in the organization. All the agent classes take charge of roles for the protocol and should interact with other agent classes through the collaboration class. This leads to the separation of agents and their interactions, which eliminates the dependencies between agents and makes ADAM more analyzable, reusable, and tractable.

The first restriction for the agent organization model is about agent's responsible functions. An agent is responsible for a message with a communicative act received from other agents. In ADAM, the behaviors of agents and their internal objects are abstracted as an interface with which agents interact.

Stereotype *ArOperation* for an instance of meta-class Operation. (1) ArOperatons are tagged to identify corresponding performatives of a message. (2) ArOperations have no return values.

Stereotypes, *ArRole* and *ArAgent*, are an instance of meta-class *Class*. (1)All ArRole operations correspond to stereotype ArOperations. (2)The Mental state of an agent has a tagged value either agreement or disagreement. (3)ArAgent has ArOperations corresponding to the ArOperation of ArRole.

In ADAM, agents communicate with each other maintaining independence through a collaboration that is a connector class. The collaboration class has interaction protocols and states for representing collaboration states between agents.

Stereotype *ArCollaboration* is an instance of the meta-class *Class.* (1)ArCollaboration has tagged value identifying protocol types. (2)ArCollaboration has tagged value identifying collaboration states in a protocol.

Stereotype *ArParticipate* is an instance of meta-class Association. (1)ArParticipate is binary association. (2)The first end of the association must be to an ArRole. (3)The second end of the association must be to an ArCollaboration. (4)Multiplicity of ArPtotocol that participate ArParticipate is at minimum one and at maximum one. (5)Multiplicity of ArRole that participate ArParticipate is at minimum one and at maximum many.

Stereotype *ArPlay* is an instance of meta-class Association. The same as for the restrictions of the ArParticipate, but the ArAgent should substitute for ArCollaboration except that the multiplicity of ArAgent participating in ArPlay is "1..M".

Finally, semantic restriction restricts a spectrum of modeling elements in the agent organization model.

The Stereotype *agent organization model* is an instance of the meta-class *Model*. (1)The Agent organization model contains architectural components. (2)Each ArRole must participate in at least one ArPlay. (3)There are the same restrictions between ArAgent and ArPlay, ArRole and ArParticipate, ArAgent and ArParticipate, and ArCollaboration and ArParticipate

3.3 The Agent Interaction Model

The agent interaction model captures a behavioral aspect of a group of agents using a collaboration or sequence diagram. It represents chronic sequences of messages among agents (or roles) according to an interaction protocol. Semantic restrictions for the constructs of the agent interaction model are shown as follows.

Stereotype *ArMessage* for an instance of meta-class Message. (1)ArMessages are tagged identifying performatives of a protocol.
Stereotype *agent interaction model* is an instance of meta-class Model. (1)Agent interaction model contains architectural components.

3.4 The Collaboration State Model

The collaboration state model captures a dynamic aspect of a MAA using state-transition diagrams. The state transition diagrams capture state transition processes of the component and connector classes. In particular, the state transition diagram of the agent collaboration class, which is a connector class, provides an analysis mechanism of collaboration states between agents because it controls all the agent interactions in a MAA.
The stereotype, *Collaboration State model*, is an instance of meta-class Model. (1)Collaboration State model contains architectural components.

3.4 Development Processes

Compared to the development processes of the traditional object-oriented system such as incremental and iterative processes, the processes of ADAM give a clean separation between architectural model and agent model. An architectural model focuses on a logical model consisting of roles, while agent model specifies a physical model based on the architectural model. Thus, this separation helps designer in constructing well-structured models.
The development process of ADAM consists of three major phases: requirement analysis, architectural analysis and design, and system analysis and design as shown in Fig. 1.
Activities of an architectural analysis and design are centered to construct role specifications based on requirement models in units of collaboration. In this phase, properties and capabilities of roles are specified by modeling, refining, and complementing between role organization and interaction models. These phases are looped in activities until role specification is completed. Then, the development goes to the agent system analysis and design phase.
In agent system analysis and design, the first activity is to map roles to agents. A cardinality of mapping between roles and agents depends on the environments of system. For example, various roles are mapped to an agent in reasons such as physical distribution of agents or performance of the overall system. In results, agents have properties and capabilities of corresponding roles. Additional properties and capabilities can be specified by modeling agent organization and interaction models. The final result of development process is a set of agent specifications. Note that role specifications are independent in implementing the system, while the implementation is to realize agent specifications.

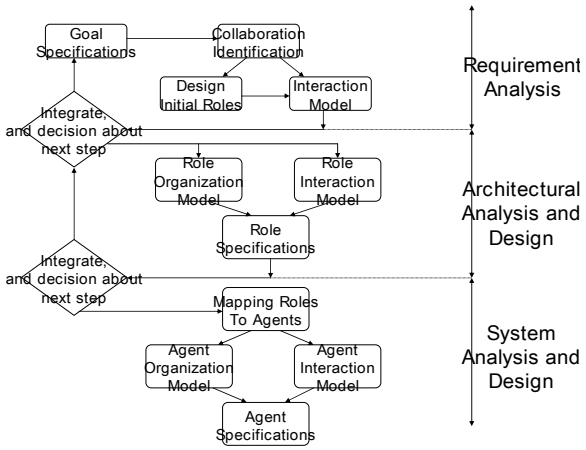
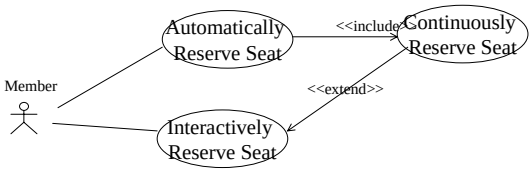


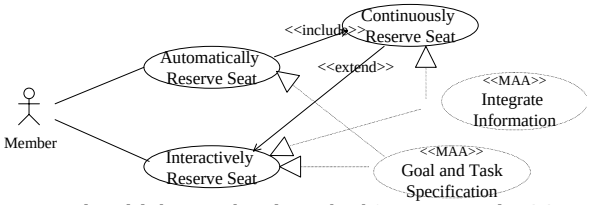
Fig. 1 Development process of ADAM

4 Case Study

This section briefly illustrates ADAM with a case study of the analysis and design of MAS for the traveler's ticketing assistance.



(a) Use case diagram of the GilBot System

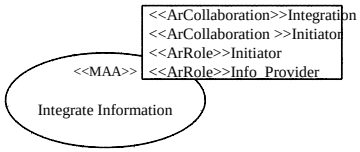


(b) Integrated model showing the relationship b/w use case and MAA

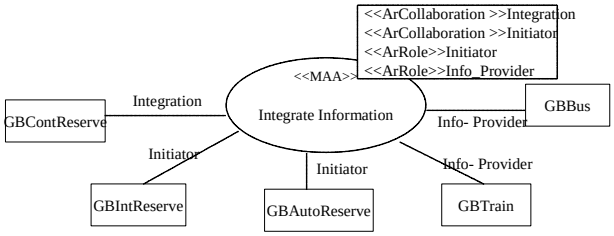
Fig. 2 The Problem Structure Model of the GilBot System

The GilBot system is an integrated ticketing system with intelligent agents. The goal of the system is to aggregate the fragmented ticket information of two transportation systems: express bus and train; and to provide integrated ticket information for reservations and confirmations. In addition, the system can continuously monitor and reserve tickets where they are available. The system includes agents such as GBHoster agent for interacting with users, GBBus

agent for providing and reserving an express bus ticket, and GBTrain agent for a train ticket.



(a) *Integrate Information* MAA

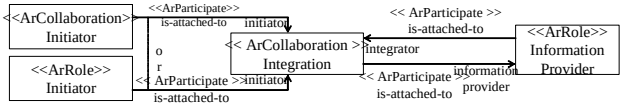


(b) Realization of the *Continuous Reservation* by the *Integrate Information*

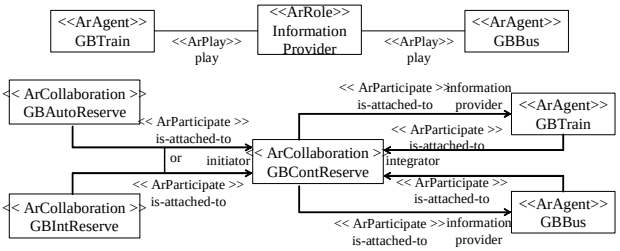
Fig. 3 Realization of a use case from the MAA

Part (a) of Fig. 2 depicts a problem structure model of the GilBot system. A user can reserve a seat automatically or interactively. The use case, "Automatically Reserve Seat", represents a subsystem that automatically reserves a seat with information of departure/arrival time and city while the use case, "Interactively Reserve Seat", is used to reserve a seat by a user. The "Continuously Reserve Seat" is a special use case for constantly monitoring the availability of seats when they are sold out and then reserving them, once they become available. Part (b) of Fig. 2 shows that the use cases of part (a) are the instantiations of MAAs, "Integrate Information" and "Goal and Task Specifications". This means that the structure and behavior of use cases are defined by the MAA. For example, the "Continuously Reserve Seat" use case has agents that are in charge of roles that are defined in the "Integrate Information" MAA. The participating agents of the "Continuously Reserve Seat" use case comply with the patterns that defined in the "Integrate Information" MAA.

ADAM integrates the use case and MAA for the implementation of requirements. The MAA is parameterized in terms of agent roles and collaborations. Part (a) of Fig. 3 depicts "Integrate Information" MAA having initiator and information provider roles and integration collaboration. This means that the "Integrate Information" MAA consists of the roles such as an initiator and an information provider, a collaboration to integrate a set of partial information, and their relationships and interactions. Part (b) of Fig. 3 depicts a realization of the "Continuously Reserve Seat" use case from the "Integrate Information" MAA by mapping agents to corresponding roles. Note that the "Integrate Information" MAA can be initiated by roles or other agent collaborations.



(a) Agent Organization Model of *Integrate Information* MAA in Architecture Layer

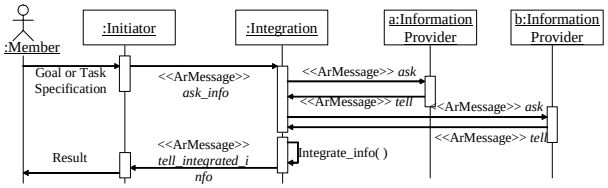


(b) Agent Organization Model of *Continuous Reservation* Use Case in Application Layer

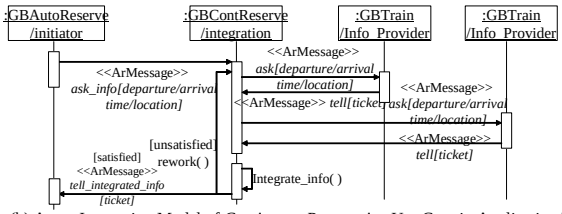
Fig. 4 Agent Organization Models of ADAM

An agent organization model of the GilBot system is shown in Fig. 4. Part (a) describes an agent organization model of the "Integrate Information" MAA in the architecture layer. In this model, there are role and collaboration classes as stereotyped classes such as integration collaboration, initiator role and information provider role classes. They are associated with <<ArParticipate>> as a stereotyped relationship. Part (b) shows how the "Integrate Information" MAA is instantiated for the "Continuously Reserve Seat" use case. GBBus and GBTrain are stereotyped classes representing agents with the information provider role while GBContReserve is a stereotyped class representing the integration collaboration. The model describes an overall structure of the MAA and use case with satisfying the topology rule. This means that a relationship between agents is defined by roles that the agents are in charge of and agents are temporally related in a given collaboration with an interaction protocol.

Fig. 5 shows interaction models of the GilBot system. Part (a) describes an agent interaction model of the "Integrate Information" MAA. In the model, there are <<ArMessages>> as a stereotyped message and the message passing between roles is represented. Part (b) is an interaction model of the "Continuously Reserve Seat" use case as a result of instantiation from part (a) like an agent organization model in Fig. 4.



(a) Agent Interaction Model of *Integrate Information* MAA in Architecture Layer



(b) Agent Interaction Model of *Continuous Reservation* Use Case in Application Layer

Fig. 5 Agent Interaction Models of ADAM

Finally, modeling a collaboration as a class in ADAM enables one to derive and analyze a state-transition model for each collaboration rather than an overall system collaboration when there are many collaborations in the system.

5 Summary and conclusion

In this paper, an Architecture-Driven multi-Agent systems development Method (ADAM) is introduced. ADAM adopts an architecture- driven approach to handle the architectural issues of MAS design such as problem structuring, agent organization with roles, and agent interaction with control. In particular, it recognizes collaborations with interaction protocols as a modeling element. This approach provides a clean separation between individual agents and their interaction. The separation makes the system more analyzable, tractable, and reusable which helps to construct a well-defined MAS.

All of models and modeling elements of ADAM extend ones of UML (Unified Modeling Language) using extension mechanisms of UML without syntactic and semantic changes to the original models, which increases the availability of developers and tools. Semantics of models and modeling elements of ADAM can be formalized by OCL.

The usefulness of ADAM has been illustrated by the specific case of the GilBot which is a MAS for the traveler's integrated ticketing system.

References:

[1] Brenner, W., Zarnekow, R., and Wittig, H., *Intelligent Software Agents: Foundations and Applications*, Springer-Verlag, 1998.

[2] 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.

[3] DeLoach, S., Wood, M., and Sparkman, C., "Multiagent Systems Engineering", *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No.3, World Science Publishing, 2001, pp. 231-258

[4] Garlan, D. and Shaw, M., "An Introduction to Software Architecture", *International Journal of Software Engineering and Knowledge Engineering*, Vol. 1, World Science Publishing, 1993.

[5] Iglesias, C.A., Garijo, M., and Gonzalez, J.C., "A Survey of Agent-Oriented Methodologies", *ATAL'98*, Paris, France, 1998, pp. 185 – 198.

[6] Iglesias, C.A., Garijo, M., Gonzalez, J.C., and Velasco, J.R., "Analysis and Design of Multi-Agent Systems Using MAS-CommonKADS", *Intelligent Agent IV(ATAL'97), LNAI 1365*, Springer-Verlag, Berlin, Germany, 1998, pp. 314 – 327.

[7] Jacobson, I., M. Christerson, P. Jonsson, G. Overgaard, *Object-Oriented Software Engineering. A Use Case Driven Approach*, Addison-Wesley, 1992.

[8] Kendall, E. A., M. Malkoun, and C. H. Jiang, "A Methodology for Developing Agent Based Systems for Enterprise Integration", *Modeling and Methodologies for Enterprise Integration*, Chapman and Hall, P. Bernus and L. Nemes, Editors, 1996.

[9] Kinny, D., Georgeff, M., and Rao, A., "A Methodology and Modeling Technique for Systems of BDI Agents", *LNAI 1038*, Springer-Verlag, Germany, 1996, pp. 56-71.

[10] Odell J., Van Dyke Parunak H. and Bauer B. "Extending UML for Agents". *AOIS Worshop at AAAI 2000*, Austin, TX, 2000.

[11] Robbins, J.E., Medvidovic, N., Redmiles, D.F., and Rosenblum, D.S., "Integrating Architecture Description Languages with a Standard Design Method", *ICSE*, Japan, 1998, pp. 209 – 218.

[12] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling & Design*, Prentice-Hall, 1991.

[13] Sycara, K., Decker, K., Pannu, A., Williamson, M. and Zeng, D., "Distributed Intelligent Agents", *IEEE Expert*, December 1996.

[15] Wooldridge, M., Jennings, N.R., Kinny, D., "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems*, Vol.3, No. 3, pp. 285-312, 2000.