Fourth International Conference on Semantics, Knowledge and Grid

Use of UML 2.1 to Model Multi-Agent Systems based on a Goal-driven Software Engineering Ontology

Pornpit Wongthongtham, Darshan Dillon, Tharam Dillon, Elizabeth Chang

Digital Ecosystems and Business Intelligence Institute, Curtin University
GPO Box U1987 Perth Western Australia 6845 Australia
{p.wongthongtham, d.dillon, t.dillon, e.chang}@cbs.curtin.edu.au

***Abstract*— In this paper, we present the use of UML 2.1 to model multi-agent systems based on a goal-driven software engineering ontology. The lack of an efficient standardized modeling language is evident. The uses of UML and stereotypes UML to model multi-agent systems have been proposed. However, there are still a number of issues with the existing approaches due to inconsistent semantics of the existing UML diagrams and unintuitive and complex notations. UML 2.1 allows representing more complex scenarios and introducing greater details into the modeling process enabling effective capture and representation of multi-agent actions and interactions. UML 2.1 has not only enabled the introduction of a notation for the Ontology based multi-agent systems, but also effective capture and representation of the dynamic processes associated with these the Ontology based multi-agent systems.**

I. INTRODUCTION

The ontology provides an important mechanism to facilitate producing semantically information used by software agents. Since the ontology has been used to express a formally shared understanding of information [1], it enables the sharing of an agreement among software agents by making assumptions explicit. The key idea is to have agreement explicitly interpreted by software agents rather than just being implicitly interpreted by humans. The representation of knowledge including ideas, tasks, models, processes as well as documentation using an ontology and sub-ontology, will provide intuitive, clear, precise concepts and ideas, knowledge and classified issues. Knowledge is organized into the ontology and used by agents as the basis for classifying knowledge enabling questions and problem solving. Additionally knowledge can be shared among agents in community.

Software agents in multi-agent systems are intelligent, autonomous problem solvers capable of getting answers from user queries, making decisions based on appropriateness, communicating with other agents and conveying results to the system or the users. They have their own goals, capabilities and beliefs which allow them to act intelligently within their field of expertise. Ontologies coupled with a multi-agent system allows greater ease of communication by aggregating the agreed knowledge and the domain knowledge into a shared information resource platform and allow them to be shared among users and enable the intelligent agents to use the ontology to carry out initial communication with users when the problem is raised in the first instance. The system utilizes software agent based computing in the sense that the agent has knowledge through consultation with ontologies in the ontology repository. Due to agent capacities in reading and reasoning published knowledge with guidance of the ontology, the shared ontology enables agents to have meaningful communications.

Issues within Software Engineering Ontology (SE Ontology) based multi-agent systems modeling include the lack of an efficient standardized modeling language. The uses of UML and adopted UML have been used to model the Ontology based multi-agent systems as a language for modeling. The latest version of UML, namely UML 2.1, has greater expressive power than previous UML versions. This allows representing more complex scenarios and introducing greater details into the modeling process enabling effective capture and representation of multi-agent actions and interactions. In this paper we illustrate how UML 2.1 can be used to model an Ontology based MAS specifically designed to software engineering domain. UML 2.1 has not only enabled the introduction of a notation for the Ontology based MAS, but also effective capture and representation of the dynamic processes associated with these the Ontology based MAS.

In section 2, we review literatures on multi-agent systems modeling. SE Ontology based multi-agent systems are described in section 3 and in section 4 we illustrate how UML 2.1 can be used to effectively model SE Ontology based multi-agent systems. The paper is concluded in section 5.

II. LITERATURE REVIEWS

UML has been used to model multi-agent systems [2]. Adopted UML also has been used as a language for modeling of agent-based systems [3-6]. Kavi et al. [3] propose to extend UML with a number of modeling constructs. Next to the Agent, the additional modeling constructs include (1) Belief, Goal and Plan to enable modeling of the reactive and proactive behaviors of agents; (2) FIPA Performative, KQML Performative and Blackboard to model agent's communication. The authors use the Sequence Diagram. However, they have changed the semantics of the Sequence Diagram by using smiley faces, thought clouds, and the like.

978-0-7695-3401-5/08 $25.00 © 2008 IEEE
DOI 10.1109/SKG.2008.45

Da Silva et al. [4] propose MAS-ML as a modeling language to support modeling of multi-agents systems. MAS-ML is an extension of UML and uses Organization, Role and Class Diagrams to model static aspects of an application while Sequence Diagrams are used to model the dynamic aspects of an application. We notice changes in the semantics of rectangles without the use of a stereotype. Additionally, the use of <<role_change>> is syntactically correct but the resulting diagram appears complex and is difficult to follow.

VisualAgent [5] is a Java-based development environment which uses the MAS-ML and is composed of three tools: a graphical tool, a transformation tool and a code generation tool. The VisualAgent can be used to present some preliminary ideas, but doesn't allow for detailed presentation as it virtually lacks existing UML diagrams or stereotypes.

Da Silva et al. [2] use UML2.0 Activity Diagrams to model agent plans and actions. They consider a plan to be an activity, decompose them into a number of actions and define the action execution sequence. The strength of this approach is that it allows definitions of stereotypes for Activity Diagrams. However, the chosen notation appears to be difficult to understand and to follow.

Use of 'AgentUML' [6] to model multi-agent systems involves the use of many standard UML diagrams. For example, a Sequence Diagram involves the inter-Agent communication where each rectangle represents an Agent/Role combination instead of a single object. Each rectangle can then be expanded to have internal processing represented with an Activity Diagram or a Statechart. Odell et al. [6] illustrate the use of Activity Diagrams as well as the use of Statecharts, one per Agent, using the Sequence Diagram rectangles as a starting point one per Agent. In this case, however, there is a departure from the normal usage of a Statechart where is represents the states that an object (not an Agent) goes through during its lifecycle. Finally, Odell et al. [6] use the Collaboration Diagram as a mirror of the Sequence Diagram with each node representing an Agent/Role combination.

We have examined AUML diagrams given in [2] and noticed that in the case of Sequence & Collaboration Diagrams each rectangle at the head of lifelines represents a single Agent/Role combination. This means that no stereotype has been defined, yet the semantics of the diagrams have been changed. This is a concern to us. Similarly, the semantics of the Statechart has been changed so the collection of states represents the lifecycle of an Agent/Role, not a single object as they are meant to. The proper way to change the semantics of existing diagrams is to define UML stereotypes. This has not been done. Also, there are bound to be unforeseen problems by simply substituting an Agent/Role where an Object is meant to be in these UML diagrams. The final problem we see with AgentUML is that is has been applied repeatedly in the very narrow domain of Agent Protocols. To make it useful it needs to be applied in a broad range of problem domains. This is what we are endeavoring to do.

III. Agents for Software Engineering Ontology

Ontologies coupled with a multi-agents systems allow greater ease of communication by aggregating the agreed knowledge (project data / information / agreement) and the domain knowledge of software engineering into a shared information resource platform and allow them to be shared enable the intelligent agents to use the ontology to carry out initial communication with users.

The agent has knowledge through consultation with ontologies in the ontology repository. Due to agent capacities in reading and reasoning published knowledge with guidance of the ontology, the shared ontology enables agents to have meaningful communications. We design a set of agents cooperating with each other and interacting with users or team members, and these are

- user agents which represent each team member being provided with services,
- safeguard agent which represents system authentication for user authorisation and access level,
- ontology agent which represents manipulation and maintenance of the SE Ontology, and
- decision maker agent which represents decision making on the matter of updating the SE Ontology.

A software engineer having and working with their own repository of software components, documents and codes, etc. interacts with his/her user agent in the system when he/she wants to enquire, to discuss a problem, to raise an issue, to make a decision, or to find answers in a multi-site distributed environment. If he/she requests to change or update project data and it is beyond his/her user agent to decide then the user agent will communicate with the decision making agent. The decision making agent then gathers information from the other team members as well as consults the ontologies from the ontology repository. Making the decision is based on the information obtained from consulting ontologies. The final solution(s) will then be raised up and sent back to the involved software engineers. In a case of the decision making agent has difficulties coming up with any solutions; the agent will put it through to the authorised person(s) or team leader to make a decision. Once the agent gets the solution(s) from the person or the team leader, it will automatically reconfigure or update the ontology, the knowledge base in the resources as well as sending back the solution(s) to the involved software engineers. Ontology-based contents and agent capability descriptions are machine-processable and thus the ontology can be correctly reconfigured by the agents. In a case of changing domain knowledge requests, the user agent is to send the requests to the decision making agent which will then require domain expert involved.

IV. Using UML 2.1 to Model SE Ontology based Multi-Agent Systems

A. *Sociable SE Ontology based Multi-Agent Systems Modeling*

An UML Sequence Diagram is generally defined across the page by a series of rectangles, each of which represents a class. Each of these rectangles has a dotted line running vertically

down the page. These dotted lines are known as lifelines. As you go down the page, time passes as messages flow between objects. UMLS 2.1 allows for a particular class to have more than one lifeline. Namely, a particular class may have many ports, each one with its own lifeline. The agent may be represented by a rectangle, and have many ports, each with its own lifeline.

We use a Sequence Diagram where Composite Classes have more than one port and represent different roles of the same agent. Hanish & Dillon [8] have previously used a similar and related approach to represent an Agent/Class playing different roles. This will enable us to model SE Ontology based multi-agent systems and represent agents which play more than one role concurrently.

Each port has its own lifeline. If there are two ports, this signifies two roles that are played by the agent from which the ports come. We use Composite Class as a rectangle at the head of lifelines in a Sequence Diagram, and each port to represent a role played by the Composite Class, rather than repeating rectangles for each class. Figure 1 shows a sequence diagram representing a sequence of processes within the SE Ontology based multi-agent systems. A number of agents play multiple roles which are represented by multiple ports. The safeguard agent plays four roles: systems authentication, access level allocation, proposal management, and monitoring users' activities. The ontology agent also plays four roles: navigation, querying, instance knowledge manipulation, and domain knowledge manipulation. The decision making agent plays two roles: reputation based decision and domain expert based decision. Depending on which role the agent is acting in when it sends/receives messages, the sequence diagram shows arrows to/from a particular lifeline for the agent.

There are a couple of points worth noting in our sequence diagram:

- the lifelines of agents are solid throughout since agents tend to be persistent
- each rectangle represents a Composite Class which implements an agent
- each distinct role played by an agent is represented by a distinct port on the rectangle with its own lifeline.

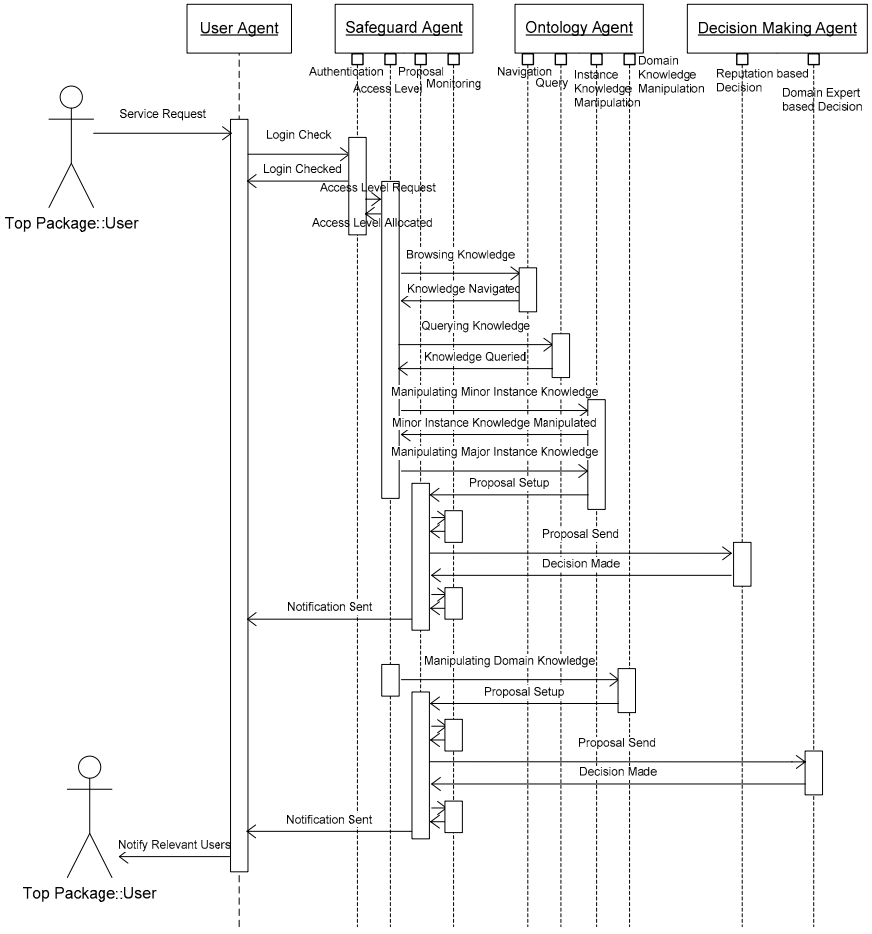

Fig.1 Sequence diagram representing sequence of processes within SE Ontology based multi-agent systems

As shown in Figure 1, we illustrate a system that has four agents. Each user is assigned to a user agent when a login is made. Each user agent is an initiator based on the user actions; the agent will carry out the specific operations accordingly. All the operations have different logic involved, but the structure of creating a user agent is the same. In the agent creation process, an agent object is created when a user login is carried out. The user agent will kill itself if the team member decides to log off the system or the team member is idle for too long. This is typical behaviour of goal specific agents, which exhibit one-shot behaviour. In other words, the agent is created for a purpose, and once the purpose is achieved, the agent will be terminated.

The safeguard agent will be doing user authentication and authorization, access levels allocation, proposal management, and monitoring users' activities. The user identification will be verified with the user database as well as access level allocation. There will be five possible cases in access level allocation. The first case is that the user navigates knowledge in the form of SE Ontology. In the second case, the user queries on the knowledge. These two cases require ontology agent to do navigation and querying on SE Ontology. Knowledge manipulation can form another three cases. Case of manipulating minor instance knowledge require ontology agent to do minor manipulating instance knowledge. Case of manipulating major instance knowledge requires both ontology agent and decision making agent to complete the task. Basically the ontology agent passes the request to the decision making agent to precede reputation based decision processes including gathering information, consulting the SE Ontology, etc. Case of manipulating domain knowledge also requires both ontology agent and decision making agent to complete the task. Similarly the ontology agent passes the request to the decision making agent to precede domain expert based decision processes. On passing through the decision making agent, the proposals are recorded through logging processes. The results of the processes are sent to the user agent that made the enquiry as well as relevant user agents that will have the affect of processes.

As the name itself states, the task of the decision making agent is to make decisions on the matters of major updates instance knowledge requests and the matters of updates domain knowledge requests. The role of reputation based decision making provides a mean for making the changes to the reflected data in the SE Ontology based on the reputation of users involved in the software engineering project. Reputation based decision making detailed processes can be found in literatures [9, 10]. Domain expert based decision making detailed processes which involve human domain experts can be found in literature [11].

To represent method lifting to define a composite class of the ontology agent, we illustrate in Figure 2. The ontology agent is defined by roles of navigation, query, instance knowledge manipulation, and domain knowledge manipulation, each of which is associated with their distinct interface. We specify these interfaces by method lifting method as shown in Figure 2. For example, interface of component class navigation relates to the interface of a composite class ontology agent. Component classes of instance knowledge manipulation and domain knowledge manipulation are inherited from component class manipulation. The instance knowledge manipulation and domain knowledge manipulation interfaces of composite class ontology agent relate to the interfaces from component class manipulation.

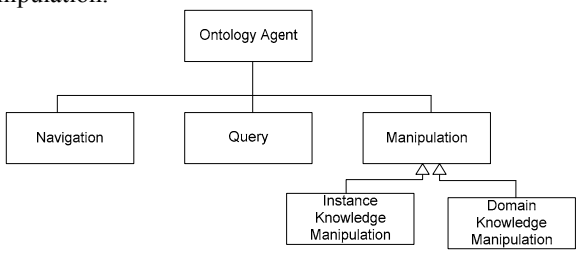

Fig. 2 Method lifting for composite class Ontology Agent

B. *Goal-driven SE Ontology based Multi-Agent Systems Modeling*

We can model the goal-driven aspect of the agent by a Composite Structure Diagram with Parts, and Ports. Each part represents a distinct area of processing within the agent. Each port represents a different role played by the agent. The <<Agent>> stereotype based on the Composite Structure Diagram [12] can be used to model the safeguard agent, the ontology agent, and the decision making agent. The <<Agent>> stereotype must have a name, at least one part which controls the efforts of the agent to achieve a goal and at least one port which relates to its playing a role.

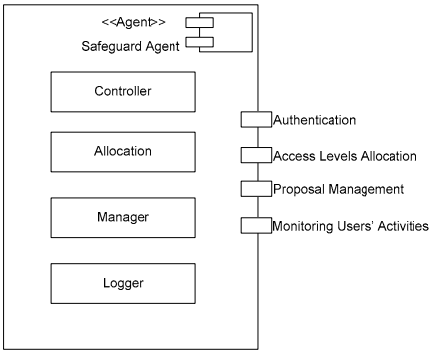

Fig. 3 Composite Structure Diagram for the safeguard agent

We use a Composite Structure Diagram to represent the goal-driven nature of an agent. In the case of the safeguard agent shown in Figure 3, we have four ports which correspond to four different roles of this agent, and four parts which show distinct areas of process within the agent. Note that the same two ports (authentication, access levels allocation, proposal management, and monitoring users' activities) that were present in the sequence diagram are also present here. Each of the ports is a construct which enables the Agent to interact

with other Agents, namely Ontology agent and decision making agent.

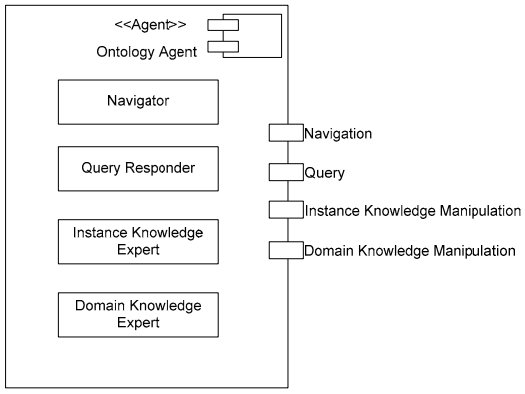

Fig 4 Composite Structure Diagram for the ontology agent

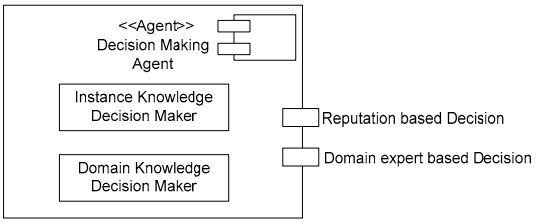

Fig 5 Composite Structure Diagram for the decision making agent

Composite structure diagrams representing goal-driven characteristic of the ontology agent and decision making agent are shown in Figure 4 and Figure 5 respectively. We have four ports corresponding with four roles of the ontology agent and four parts illustrating distinct processes within the ontology agent while we have two ports (two roles) for the decision making agent and two parts within the decision making agent.

V. CONCLUSIONS

In this paper, we use UML 2.1 sequence diagram and composite structure diagram to model social and goal-driven SE Ontology based multi-agent systems. The crucial point is that we have not changed the semantics of the sequence diagram which makes our use of UML 2.1 valid. In our future works, we will examine the use of other UML 2.1 diagrams.

REFERENCES

[1] Gruber, T.R. *Toward principles for the design of ontologies used for knowledge sharing*. in *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*. 1993. Padova, Italy: Kluwer Academic Publishers, Deventer, The Netherlands.

[2] J. Odell, H.P.V.D.a.B.B., *Representing Agent Interaction Protocols in UML*, in *Agent-Oriented Software Engineering*, M.W. P. Ciancarini, Editor. 2001, Springer-Verlag. p. 121-140.

[3] K. Kavi, D.C.K., H. Bhambhani, G. Pancholi, M. Kanikarla. *Extending UML for Modeling and Design of MultiAgent Systems*. in *the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS2003)*. 2003. USA.

[4] V. T. da Silva, R.C.N., C. J. P. de Lucena, *Using the MAS-ML to model a multi-agent system.* Software engineering for multi-agent systems II : research issues and practical applications, 2004. **2940**: p. 129-148.

[5] B. A. de Maria, V.T.d.S., R. C. Noya, C. J. P. de Lucena. *VisualAgent: A Software Development Environment for Multi-Agent Systems*. in *The 20th Brazilian Symposium on Databases and 19th Brazilian Symposium on Software Engeneering*. 2005. Brazil.

[6] V. T. da Silva, R.C.N., C. J. P. de Lucena. *Using the UML 2.0 Activity Diagram to Model Agent Plans and Actions*. in *the 4th International Conference on Autonomous Agents and Multiagent Systems*. 2005. Netherlands.

[7] Wand, Y., V.C. Storey, and R. Weber, *An Ontological Analysis of the Relationship Construct in Conceptual Modeling.* ACM Transactions on Database Systems, 1999. **24**(4): p. 495-528.

[8] Dillon, A.A.H.a.T.S. *Object-oriented behavior modeling for real-time design*. in *the3rd International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '97)*. 1997. USA.

[9] Wongthongtham, P., Chang, E., Dillon, T.S., Sommerville, I. , *Software Engineering Ontology - Instance Knowledge Part I.* International Journal of Computer Science and Network Security, 2007.

[10] Wongthongtham, P., *A methodology for multi-site distributed software development*, in *School of Information Systems*. 2006, Curtin University of Technology: Perth.

[11] Wongthongtham, P.C., E. , *Towards Social Network based Approach for Software Engineering Ontology Sharing and Evolution*, in *LNCS On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*. 2007, Springer LNCS

[12] D.S. Dillon, T.S.D., and E. Chang. *Using UML 2.1 to model Multi-Agent Systems*. in *the 6th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*. 2008. Italy.