An Application of Agent UML to Supply Chain Management

Marc-Philippe Huget

Agent ART Group
University of Liverpool
Liverpool L69 7ZF
United Kingdom
M.P.Huget@csc.liv.ac.uk

Abstract. Agent UML is a graphical modeling language for describing multiagent systems but until now, it has not been applied to real-world applications. The aim of our project is to apply Agent UML to Supply Chain Management. This project has several objectives: (1) it allows us to discover which diagrams have to be enhanced, modified or created and, (2) it allows us to define a methodology based on Agent UML and several tools. This paper summarises our preliminary results on applying Agent UML to Supply Chain Management and gives an overview of the UML diagrams that are of particular interest to design and development of multiagent systems. We conclude with a discussion of future work in this area.

1 Introduction

For several years, methodologies and graphical modeling languages have been supplied to designers in order to design systems, software and components. UML [3] is certainly the best known graphical modeling language. For several years, multiagent system designers have the same possibility with some modeling languages like Agent UML [9] [2]. Agent UML is based on UML. As Odell and Bauer quoted it, it is not possible to directly use UML since several differences exist between agents and objects like the autonomy or the ability to cooperate [8]. However, it seems to be important to capitalize on the skills of designers. Multiagent system designers are often software engineers who use UML or are aware of it. Even if a strong interest is manifest on Agent UML (see the Internet site http://www.auml.org for papers on the subject), at present, it does not exist an application of Agent UML to real-world applications. The only example that we can give is MOTIV-PTA [1] but this paper does not present the impact of Agent UML during the design.

We currently lead a project trying to apply Agent UML to the example of Supply Chain Management [10] in order to (1) find which diagrams in Agent UML have to be enhanced, modified or created and (2) design methodologies and tools for Agent UML. This paper presents our first results on applying Agent UML to the application of Supply Chain Management.

Due to lack of space, we only present here our conclusions on applying Agent UML diagrams. A longer version of this paper is in [5].

2 Agent UML Diagrams

As stated in introduction, Agent UML is an extension of UML in order to tackle the differences between agents and objects. As a consequence, Agent UML has several representations:

1. Sequence diagrams
2. Collaboration diagrams
3. Activity diagrams
4. Statechart diagrams
5. Use case diagrams
6. Class diagrams
7. Object diagrams
8. Packages
9. Component diagrams
10. Deployment diagrams

Sequence diagrams are used in Agent UML to represent interaction protocols. These diagrams are already modified [9] [2] [6]. The application gives us some feedback on what could be done: (1) it is impossible during all the interaction to stop and to prevent abnormal execution. The notion of exceptions and triggering actions are not present. Such proposals are given in [6]. Moreover, it is not possible to link several protocols together. We do not speak about nested or interleaved protocols but about the side effect of protocols each other. For instance, if the agents currently consider the negotiation and simultaneously, the client cancels her order, the first protocol is no longer valid. It is not possible to notify the side effects between interaction protocols. (2) implementing this kind of protocol diagram by hand is painful and it does not prevent errors. Designers need tools for designing their protocol diagrams.

Collaboration diagrams and activity diagrams are not considered for the moment.

Statechart diagrams allow us to represent the dynamic of the system and particularly, the flow between elements in the system. Statechart diagrams consider the different states of the system and how to go from state to state through actions. Statechart diagrams seem to be an interesting approach in order to represent agents' behaviors. When reading statechart diagrams, readers can point out that it is impossible to know (except with notes) who is responsible for the events and to whom the actions are addressed. The main drawback in statechart diagrams when considering multiagent systems seems to be the inability to represent concurrent actions. Actually, it is possible that the order acquisition agent receives an order while negotiating another one.

Use case diagrams represent the use cases, the actors and the relationships between the actors and the use cases. A use case can be seen as a scenario in the system, for instance when the user tries to log on a system and a password is required. In multiagent systems, use cases are interesting when realizing the requirement analysis. They are helpful during meetings between end users and designers since they are graphical. It is easier for users to seize the different elements of the system. The main advantage of the use cases is to focus on the what and not on the how, that is to say on the system behavior and not how the system is implemented.

Class diagrams in UML represent the different classes and their connections. Class diagrams correspond to the architecture of the system. Class diagrams seem to be interesting in order to represent the different agent roles and the relations between the roles. Since agents and objects are not exactly the same, we think that class diagrams have to be updated in order to consider agent features like beliefs, intentions, plans or knowledge. UML class diagrams only consider attributes and operations. A first proposal of extension of these class diagrams is in [1]. We enhance this proposal in [4].

UML and as a consequence Agent UML allow us to represent several level of abstractions when designing class diagrams. Sometimes it is not interesting to have an accurate view of the system with the dependencies and the attributes. High-level class diagrams allow to seize the system in its entirety. We consider here two levels: the conceptual level and the implementation level. The conceptual level is a high-level view of the system getting rid of the details such as how agents are implemented or the connected classes. The implementation level is a detailed view of the system with the detailed information. Designers can also define a range of view between these two ones according to their requirements.

Class diagrams are central and a lot of details have to be given on class diagrams. It is not easy to do that without a proper method. It seems to be important to have a methodology for defining these class diagrams.

Object diagrams are not considered for the moment. They allow to represent the snapshot of instances of classes during execution. They could be interesting for a validation stage if designers want to check if at some moment, agents have a particular state.

Component diagrams describe the physical elements used during run-time. Components represent elements accessible by interfaces. Components are Java classes, data files, databases, legacy software, etc. Previous diagrams presented above concern the design of multiagent systems. This diagram and the next one deal with the development of multiagent systems.

It seems that component diagrams are required in multiagent systems in order to perform compilation of agents and in order to know the dependencies between elements. The first use does not arise some missing elements in these diagrams.

Deployment diagrams are used to represent the multiagent system configuration at run-time; that is, these diagrams describe how agents and resources are deployed on machines.

Deployment diagrams intervene at the very end of the development and allow to describe the needed resources and how agents are deployed. These diagrams seem to be worthwhile when multiagent system designers and deployment designers are not the same. Deployment diagrams represent the static view of the deployment. It could be a problem in the context of mobile agents since they are not able to represent the translation.

3 Conclusion

The graphical modeling language Agent UML is now in the toolbox of multiagent system designers. However, Agent UML is in its infancy and further effort is required in order to instantiate completely Agent UML to the domain of multiagent systems. Until now, there is few work on Agent UML and less work on applying Agent UML to real-world applications. The aim of this paper is to present our very first results on the application of Agent UML to the Supply Chain Management example. This example seems to be enough important to allow designers to seize what is missing in Agent UML and what is wrong.

The application of Agent UML to Supply Chain Management arises several questions and highlights some lacks in the current version of Agent UML. Numerous directions of research are possible. We specifically follow these directions: (1) going deeper in the design of this application in order to find what is missing in Agent UML, (2) designing tools for handling Agent UML diagrams[1] and (3) defining a methodology for the design of diagrams.

References

1. B. Bauer. UML class diagrams revisited in the context of agent-based systems. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Proceedings of Agent-Oriented Software Engineering (AOSE 01)*, number 2222 in LNCS, pages 1–8, Montreal, Canada, May 2001. Springer-Verlag.
2. B. Bauer, J. P. Muller, and J. Odell. An extension of UML by protocols for multiagent interaction. In *International Conference on MultiAgent Systems (ICMAS'00)*, pages 207–214, Boston, Massachussetts, july, 10-12 2000.
3. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1999.
4. M.-P. Huget. Agent UML class diagrams revisited. Technical Report ULCS-02-013, Department of Computer Science, University of Liverpool, 2002.
5. M.-P. Huget. An application of agent UML to supply chain management. Technical Report ULCS-02-015, Department of Computer Science, University of Liverpool, 2002.
6. M.-P. Huget. Extending agent UML protocol diagrams. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *AAMAS Workshop on Agent-Oriented Software Engineering (AOSE)*, Bologna, Italy, July 2002.
7. M.-P. Huget. A language for exchanging Agent UML protocol diagrams. Technical Report ULCS-02-009, Department of Computer Science, University of Liverpool, 2002.

[1] A first step towards this goal is to define a format for exchanging diagrams. For further details see [7].

8. N. R. Jennings and M. Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, *Handbook in Agent Technology.* MIT Press, 2000.
9. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, Austin, Texas, july, 30 2000. ICue Publishing.
10. J. Swaminathan, S. Smith, and N. Sadeh-Koniecpol. Modeling supply chain dynamics: A multiagent approach. *Decision Sciences*, April 1997.