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 certainly the most well-known graphical modeling language for describing multiagent systems but until now, it is not applied to real-world applications. The aim of our project is to apply Agent UML to the Supply Chain Management. This project has several objectives: (1) it allows to prove that Agent UML can be applied to real-world applications, (2) it allows to discover what is missing in Agent UML and what is wrong and finally, (3) it allows to define a methodology based on Agent UML and several tools. The aim of this paper is to sum up our first results on appying Agent UML to the Supply Chain Management and especially, the paper sketches what diagrams are interesting and what could be done after in our project.

This paper is published as the technical report ULCS-02-015 from the department of computer science, University of Liverpool. Version 2002/04/22

1 Introduction

Since several years, methodologies and graphical modeling languages are provided to designers in order to design systems, software and components. UML [3] is certainly the most well-known graphical modeling language. Since some years, multiagent system designers have the same possibility thanks to 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 use directly UML since several differences exist between agents and objects like autonomy or ability to cooperate [8]. However, it seems to be important to capitalize on the skills of designers. Indeed, 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 a real-world application. 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.

The aim of this paper is to present the first results of a project trying to apply Agent UML to the example of Supply Chain Management. This project has three objectives: (1) proving that Agent UML can be applied to real-world applications. (2) applying Agent UML to a real-world application so as to determine what diagrams are missing, what diagrams are incomplete and what is wrong. Readers must bear in mind that Agent UML diagrams are in fact UML diagrams and since there are some differences between agents and objects. It is possible that diagrams are not perfectly fitted for multiagent systems. (3) defining methodologies and tools in order to help designers using Agent UML.

The Supply Chain Management [12] corresponds to the management of the production in companies from the acquisition of orders to the production and the delivery. Supply Chain Management is focused on companies. Supply Chain Management concerns also clients and providers of materials.

The remainder of the paper is as follows. Section 2 presents the different representations available in Agent UML. Section 3 explains what the Supply Chain Management is. Section 4 gives the agent decomposition of the Supply Chain Management. This section is the chore of this paper and presents the different diagrams that we apply to this application. Section 5 sums up the first results of the application of Agent UML to the Supply Chain Management example and gives some directions of our project.

2 Agent UML representations

As quoted 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

The five first representations are dynamic representations whereas the four last one are static. Sequence diagrams are already modified since they are called now Protocol diagrams [9] [2] and correspond to the description of interaction protocols. Class diagrams are already modified. Readers do not forget that agents and objects are not exactly the same explaining why these diagrams have to be changed. Two proposals exist for Agent UML Class diagrams: Bauer's approach [1] and ours [5]. We do not explain in this section the meaning of all these diagrams. They are explained in details further except for the following diagrams which are not used: collaboration diagrams, activity diagrams and object diagrams.

3 What is the Supply Chain Management?

The Supply Chain Management models the production in companies from placing an order to the delivery. This application implies receiving an order as well as generating a plan for producing this order, negotiating the price and the delay, modifying the plan if some constraints are violated, producing the product and finally, delivering it.

The Supply Chain Management considers three type of actors (see Figure 1): (1) the clients who place, modify and delete orders, (2) the company with the customer relationships and the production and (3) the providers of raw materials who are another companies.

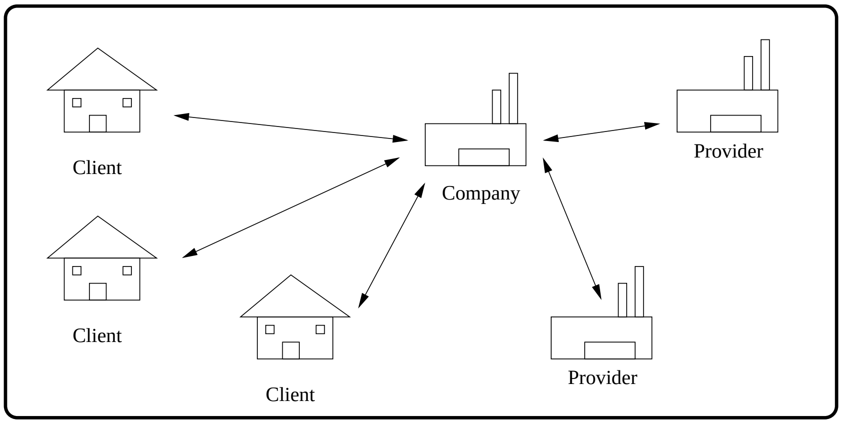

Fig. 1. The Supply Chain Management

4 Application of Agent UML Diagrams

4.1 Agent Decomposition of the Supply Chain Management

Several applications of multiagent systems to the Supply Chain Management exist in the literature. Shen and Norrie describe several platforms [11]. Multiagent system for Supply Chain Management is one domain of research of the Entreprise Integration Laboratory from the university of Toronto[1]. Our multiagent system decomposition is close to the one proposed in the Integrated Supply Chain Management system [4].

Several agents are involved in this application as shown on Figure 2:

➢ a client agent who places, modifies and deletes orders.

[1] http://www.eil.utoronto.ca

- ➢ an order acquisition agent who receives the orders and negotiates the price and the delay with the logistics agent. It is responsible for the connection between clients and the company. It informs clients as soon as deliveries are postponed or delivered.
- ➢ a logistics agent who negotiates the delay and the price with the order acquisition agent. As soon as orders are accepted, it asks the scheduler agent to generate a plan for this order. This plan is sent to the agents transporter, resource and dispatcher. It asks for a new plan as soon as some modifications appear either if the client wants to modify the order or if some delays arise on the production.
- ➢ a scheduler agent who generates a plan according to the order, the delay and the allocation of workers and resources.
- ➢ a transporter agent who is responsible for the delivery of raw materials for the production and for the delivery of the final product to the client.
- ➢ a resource agent who manages the allocation of materials in order to realize the order. If the materials are not available, it places an order to provider agents.
- ➢ a dispatcher agent who manages worker agents. It allocates work to worker agents. The dispatcher agent informs the logistics agent if some problems arise occurring a delay for the delivery.
- ➢ several worker agents who realize the work. If they are unable to complete their tasks at time, they inform the dispatcher agent.
- ➢ several provider agents who provide raw materials if the company has not enough materials for the order. In fact, a provider agent is an order acquisition agent from another company.

The remaining of this section gives the application of Agent UML diagrams to the Supply Chain Management example.

4.2 Use Case Diagrams

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 meeting 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.

The Figure 3 presents the use case diagram for the client agent.

We have one actor on this figure: the client agent. Use cases are rendered as ovals: place order, modify order and delete order. Solid lines between the actor and the use cases denote that the actor uses these scenarios.

Use case diagrams seem to have a minor interest when designing multiagent systems. They could be interesting if end users are not the designers or if end users do not understand another diagrams.

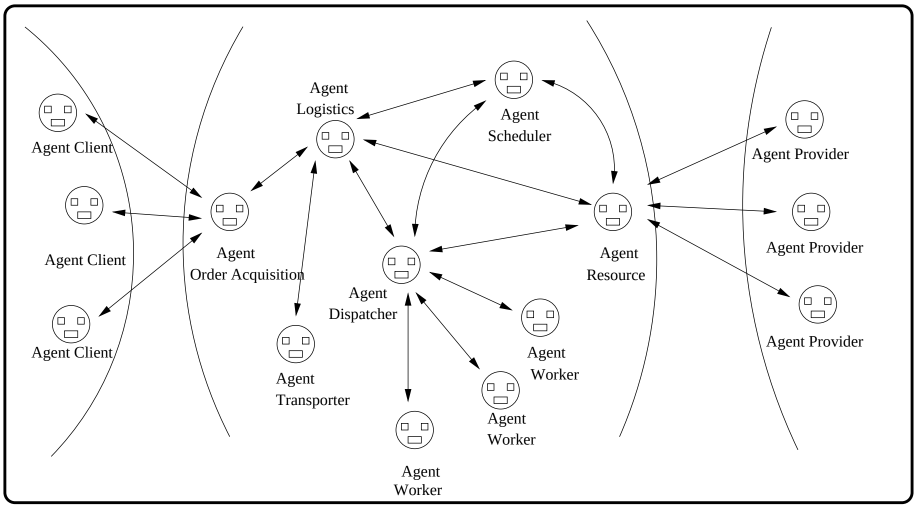

Fig. 2. Multiagent System Decomposition of the Supply Chain Management Application

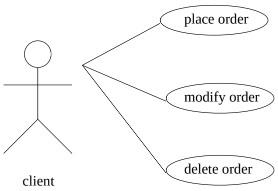

Fig. 3. Use Case Diagram for the Client Agent

4.3 Class Diagrams

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, one can 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 [5].

UML and as a consequence Agent UML allow to represent several level of abstractions when designing class diagrams. Actually, sometimes it is not interesting to have an accurate view of the system with all the dependencies and all 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 off the details such as how agents are implemented or the connected classes. The implementation level is a detailed view of the system with all the information. One more time, designers can define a range of view between these two one according to their needs.

A class diagram for the conceptual level for the Supply Chain Management example is shown on Figure 4.

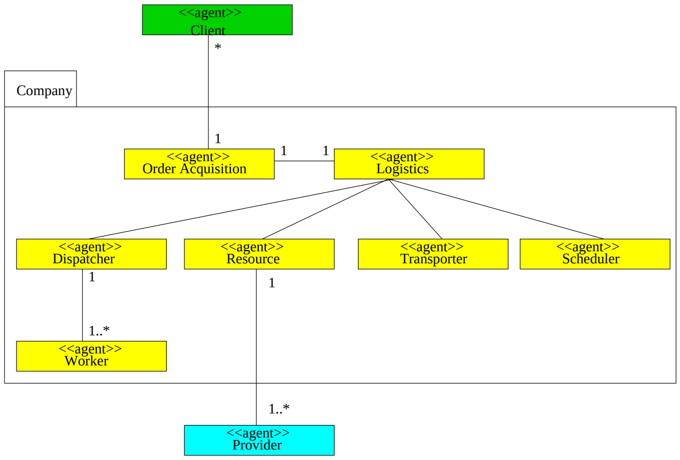

Fig. 4. Conceptual Level of the Supply Chain Management

On this figure, we suppress all the connected classes since they are not vital for the comprehension of the system. We add one package in order to group all the agents of the company. A package is a standard mechanism in UML for grouping elements.

The second level is the implementation level which gives a detailed view of the agents, the classes and the connections. The implementation level class diagram of the client agent is shown on Figure 5.

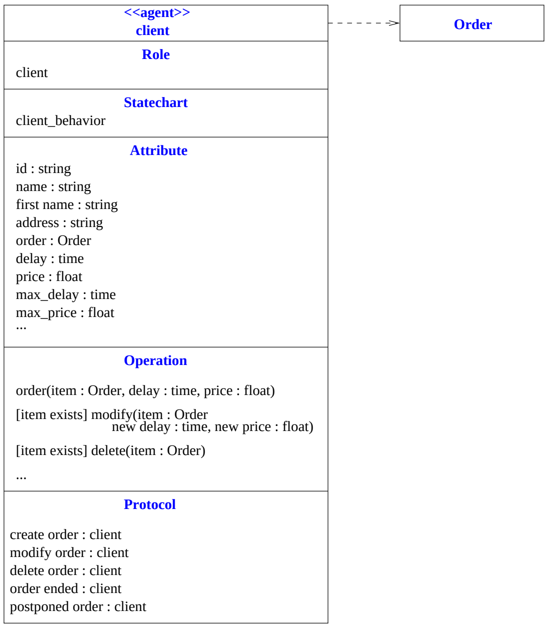

Fig. 5. Implementation Level of the Client Agent

Class diagram of the client agent is composed of several compartments. The first compartment gives the name of the agent. In order to distinguish agents from objects, each agent is prefixed by *«agent»*. The second compartment corresponds to the different roles played by this agent. In the case of the client agent, there is only one role. The next compartment gives the behavior of the agent. Since statechart diagrams can be huge, we only insert the name of the diagram *client_behavior* and define the diagram elsewhere. The compartment *attribute* gives all the attributes for this agent. We add beliefs and knowledge as well. These attributes deal with the order and the identity of the agent. Ellipsis at the end of the compartment means that the list of attributes is not given here in its entirety. It is the case since we do not describe attributes such as the physical address of the agent (IP address).

The compartment *operations* gives all the operations in the agent. We only give three operations here: order, modify and delete. Ellipsis at the end of the compartment indicates that the list is not exhaustive. The operation *order* has parameters and the operations *modify* and *delete* have pre-conditions. These conditions must hold in order to execute the operations. The format for the conditions is free, either designers use the formal language OCL [10] or a text. Pre-conditions mean that the order must exist before modifying or deleting it. The last compartment deals with the interaction protocols. It describes what protocols the agent can use and what are its roles in these protocols. Finally, one dependency is given between the client agent and a class order. Actually, some operations in the agent use this class.

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.

4.4 Statechart Diagrams

Statechart diagrams as well as activity diagrams and collaboration diagrams belong to interaction diagrams. These diagrams allow 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 one state to another one through actions.

Statechart diagrams seem to be an interesting approach in order to represent agents' behaviors. Indeed, agents behave according to actions in the environment or inner actions. Each time actions occur, agents move from one state to another one.

Due to lack of space, it is impossible to represent all the possible statechart diagrams for this application. We only consider the statechart diagram for the order acquisition agent (see Figure 6).

Agent states are rendered as a rectangle with rounded corners. The connections between two states correspond to either an action or an event. Actions and events are distinguished by the parentheses after the name. Actions are equivalent to methods in programming languages explaining why they are written with parentheses. The initial state is the state *idle*.

When reading statechart diagrams, readers can point out that it is impossible to know (except with notes) who is responsible of the events and to whom the actions are addressed. The main drawback in statechart diagrams when considering multiagent systems seems to be the unability to represent concurrent actions. Actually, it is possible that the order acquisition agent receives an order while negotiating another one. In this case, the client has to wait that the order acquisition agent has to move to the initial state.

4.5 Protocol Diagrams

Protocol diagrams called Sequence diagrams in UML describe the message flow between agents. In fact, messages do not go from one agent to another one but from one role to another one. The advantage of this approach is to reduce the size of the diagram. Designers do not represent the message from the client Smith to the company ACME but from the client role to the seller role. Several work exist on protocol diagrams [9] [2] among others.

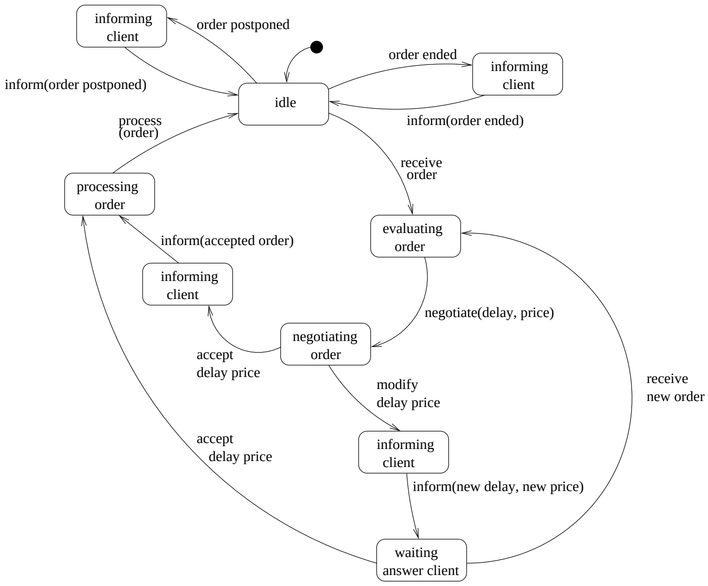

Fig. 6. Statechart Diagram of the Order Acquisition Agent

Several interactions exist in our example of Supply Chain Management given numerous protocol diagrams. Here are the list of scenarios given interaction protocols:

➢ order a product where all the agents in the multiagent system intervene.
➢ modify an order where all the agents in the multiagent system intervene.
➢ cancel an order where the agents `client`, `order acquisition`, `logistics`, `dispatcher`, `transporter`, `resource` and `worker` intervene.
➢ deliver a product where the agents `client`, `order acquisition`, `logistics` and `transporter` intervene.
➢ postpone the delivery of a product where the agents `client`, `order acquisition`, `logistics`, `scheduler` and `transporter` intervene.

Due to lack of space, we only focus here on the order of one product. This protocol interaction is really central in the multiagent system and all agents are used. Actually, the client agent passes an order, the order acquisition agent receives an order and negotiates the price and the delay with the client agent and the logistics agent. The logistics agent computes the price and the delay according to the information coming from the scheduler agent. The scheduler agent needs to ask the agents transporter, resource and dispatcher for generating a plan and defining a delay. The provider agent is called if the resource agent does not have the materials for the order.

The corresponding protocol diagram is shown on Figure 7.

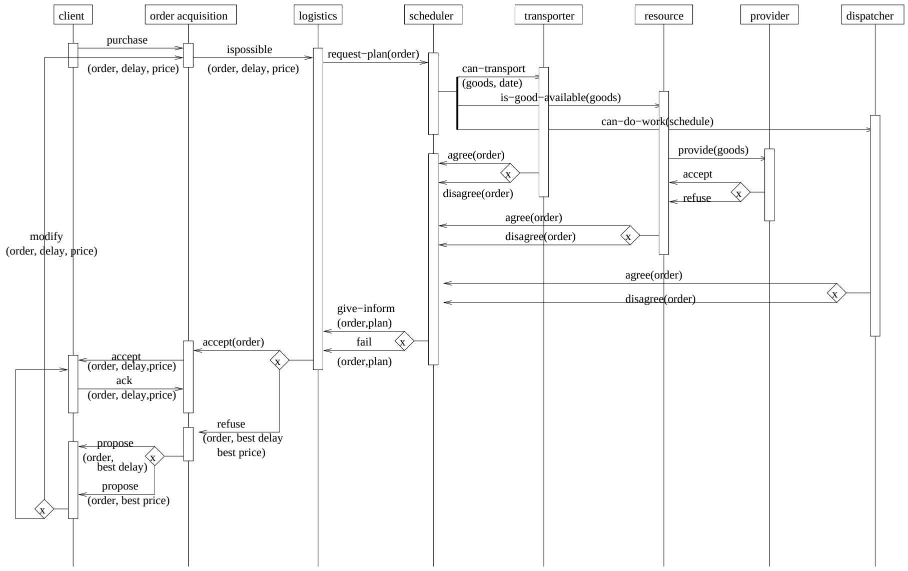

Fig. 7. Protocol Diagram for the Order

We do not explain in details this protocol diagram since many messages are used. Several notions from protocol diagrams are present in this diagram:

- ➢ the roles played by agents, for instance client, order acquisition, etc.
- ➢ the messages between roles, for instance purchase(order,delay,price) between the client agent and the order acquisition agent.
- ➢ the connector AND used by the scheduler agent. The connector AND means that all messages are sent concurrently. The connector AND is rendered as a thick solid line.
- ➢ the connector XOR used by several agents, the order acquisition agent and the logistics agent for instance. The connector XOR means that one message has to be chosen between the set of messages. The logistics agent informs the order acquisition agent either it accepts the order with the conditions or it refuses the order with these conditions. The connector XOR is rendered as a line ending with a diamond and an 'x' within.

Protocol diagrams are certainly the most achieved diagrams in Agent UML since several work already consider them. However, this example 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 notions of exceptions and triggering actions are not present. Such a proposal is 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 one protocol on another one. For instance, if the agents currently consider the order of a production and 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) realizing this kind of protocol diagrams by hand is painful and it does not prevent errors. Designers need tools for designing their protocol diagrams.

4.6 Component Diagrams

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

In this section, we only give the example of the Order Acquisition agent (see Figure 8). Several elements are represented on this diagram: (1) the executable component called OrderAcquisition.class which is a Java class. This component is connected to several source files: OrderAcquisition.java, Order.java and Client.java. The first one corresponds to the agent source code. The two others are used for handling order database and client database.

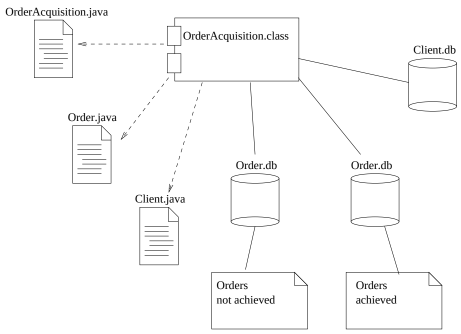

Fig. 8. A Component Diagram for the Order Acquisition Agent

Two types of connections are on this diagram: a solid line between OrderAcquisition.class and databases and a dashed line between OrderAcquisition.class and the source files. These relations correspond to the one that we can find in class diagrams. A solid line corresponds to an association. A dashed line corresponds to a dependency. Actually, as soon as the source files are modified, the executable must be compiled again.

Component diagrams seem to be needful in multiagent systems in order to do the compilation of agents and in order to know the dependencies between elements. This first use does not arise some missing elements in these diagrams.

4.7 Deployment Diagrams

Deployment diagrams are used to represent the multiagent system configuration at run-time. It means that these diagrams describe how agents and resources are deployed on machines. Since multiagent systems are distributed systems, deployment diagrams present on what machine agents are. Deployment diagrams are a static view of the deployment and can be seen as specialized class diagrams where one does not have classes but components, nodes and agents.

Such an example of deployment diagram is given on Figure 9 for our example of Supply Chain Management. Obviously, there does not exist an unique deployment diagram.

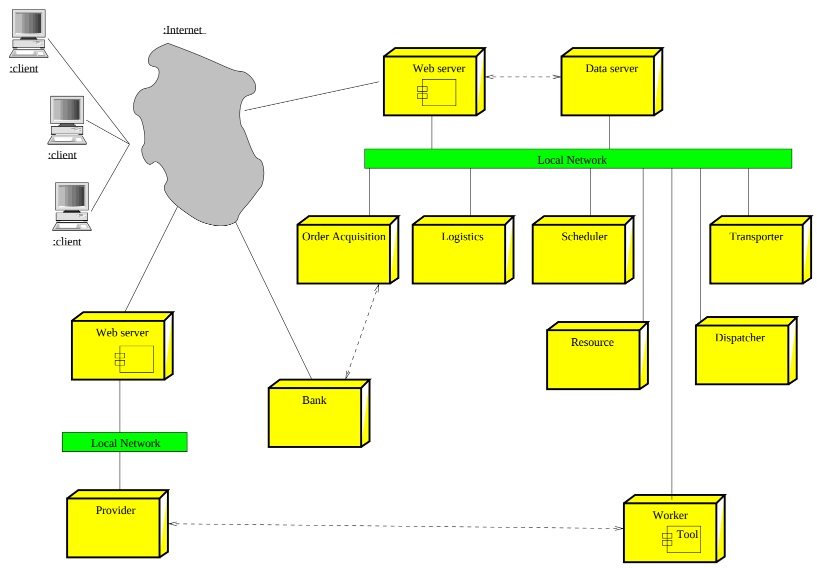

Fig. 9. A Deployment Diagram of the Supply Chain Management

Client agents are rendered as computers located on the left of the figure. We can see that clients do not have a specific name but only a class `:client`. Actually, it is not interesting to represent the instanciation of the class client like `Smith:client` since it is not possible to know in advance who are the clients. However, it would be possible to instantiate provider agents if this company only works with a specific set of other companies. Clients are linked to the company through the Internet and a Web server. A *web server* is represented as a node on the figure. A node is a physical element corresponding to computational resource and used during run-time. It is rendered as 3D box. This node has a component within which is the software used for providing a web server. This web server is connected to a data server rendered as a directed dashed line. All the agents of the company share a local network and they are all nodes. The worker agent has a component in its node: a tool. The provider agent has a specific local network since it belongs to another company. We finally insert the connection to the bank for the debit/credit operations.

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 move.

5 Conclusion

The graphical modeling language Agent UML is now in the toolbox of multiagent system designers. However, Agent UML is in its infancy and efforts have to be done 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 remaining of this conclusion presents our conclusion on this first step of the project.

1. Most of the UML diagrams seem to be used in Agent UML as this paper demonstrates it. We applied here the use case diagrams, the class diagrams, the statechart diagrams, the protocol diagrams, the component diagrams and the deployment diagrams.
2. Use case diagrams are interesting if designers want to describe what is the purpose of the system and no how it is realized. They can be used during requirement analysis in order to help end users understanding the architecture of the system and how it works.
3. Class diagrams are modified in depth [1] [5] since agents and objects are not the same. Class diagrams encompass now beliefs, intentions, plans, knowledge, behaviors and interaction protocols. These diagrams are vital and as a consequence really complex to achieve. A certain effort has to be done by designers in order to manage properly this stage. It is important that designers have methodologies for fulfiling these diagrams.
4. Statechart diagrams describe the state of the system and in our proposal, the agents' behaviors. Statechart diagrams are inefficient since they are unable to represent concurrent actions. It is then impossible to negotiate an order while receiving another one. Moreover, actions and events are unlabeled so it is impossible to know what agents are considered for these one.
5. Protocol diagrams are the most achieved diagrams. They describe interaction protocols. Few modifications have to be provided to these diagrams save the exception and the triggering actions. It is impossible to trap abnormal

behaviors. One can also point out that nothing is done for the side effect of protocols. Finally, no tools are available for designing Agent UML protocol diagrams.
6. Component and deployment diagrams answer to our needs. However, we can highlight that deployment diagrams are a static view of the deployment, thus it is not possible to represent mobile agents.
7. No methodologies or tools are available at present in order to help designers when they design Agent UML diagrams.

The application of Agent UML to Supply Chain Management arises several questions and highlight some lacks in the current version of Agent UML. Numerous directions of research are possible. We follow specifically 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. In this case, we need a format for exchanging diagrams. A first proposal is done in [7]. (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. S. Fox, J. G. Chionglo, and M. Barbuceanu. The integrated supply chain management system. Technical report, Enterprise Integration Laboratory, University of Toronto, 1993.
5. M.-P. Huget. Agent UML class diagrams revisited. Technical Report ULCS-02-013, Department of Computer Science, University of Liverpool, 2002.
6. M.-P. Huget. Extending agent UML protocol diagrams. Technical Report ULCS-02-014, Department of Computer Science, University of Liverpool, 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.
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. OMG. UML 1.4. Technical report, OMG, 2001.
11. W. Shen and D. H. Norrie. Agent-based systems for intelligent manufacturing: A state-of-the-art survey. *Knowledge and Information Systems*, 1(2):129–156, 1999.
12. J. Swaminathan, S. Smith, and N. Sadeh-Koniecpol. Modeling supply chain dynamics: A multiagent approach. *Decision Sciences*, April 1997.