UML and Agents: Current Trends and Future Directions

Marc-Philippe Huget
Agent ART Group
Department of Computer Science
University of Liverpool
Liverpool L69 7ZF
United Kingdom
M.P.Huget@csc.liv.ac.uk

James Odell
James Odell Associates
3646 W. Huron River Dr.
Ann Arbor, MI 48103
USA
email@jamesodell.com

Bernhard Bauer
Siemens, CT IC 6
81730 München
Germany
bernhard.bauer@siemens.com

Abstract

Recently, the development of multiagent systems is increasing significantly, particularly in the domain of electronic commerce and business [31]. At the same time, designing multiagent systems becomes more and more complex. As a consequence, designers need methodologies and tools to build them. Current methodologies used in object-oriented software engineering do not address this richness and the distribution of agents increases the complexity of the problem. Two main directions are considered in multiagent system design: extending software engineering methodologies (or knowledge engineering ones) or providing specific agent engineering methodologies. UML with its extension mechanisms (stereotypes and tagged values) is a worthwhile candidate. Moreover, UML is widespread in software community. UML has been extended for agents along several approaches: Agent UML [36], MESSAGE/UML [8], AOR [43], Tropos [32] or PASSI [7] among others. The paper presents both the state of the art for using UML to represent agent-based systems, as well as possible future directions of work in this domain.

1. Introduction

Multiagent systems are a domain of research in distributed artificial intelligence as well as distributed problem solving. Multiagent systems may be considered as a prolongation of the work on expert systems where distribution of the expertise has been added. Agents are spread all over the network and expertise is divided among agents. Two domains of research are particularly considered in multiagent systems: problem solving and simulation. In a context of problem solving, agents encompass a portion of the expertise. The problem is solved through their interactions. Early examples are DVMT [1] and flight scheduling [10]. More recent work is done around mobile computing, business processes or logistics, e/m-commerce, information sourcing and filtering. In a context of simulation, agents and multiagent systems act according to a model. Such models are commonly based on an ethology such as ants [13] or from human societies. In the latter case, the range of applications is broad. It might be simulation of humans in ecosystems [33], economic-based simulation [29] or industrial simulation [17].

A commonly-held idea is to consider agents and objects as equivalent notions. This view is too restrictive to show all the richness of agents. As Odell [35] and Wooldridge [45] note it, even if agents and objects have some points in common, many differences exist between these two approaches. The main differences are autonomy and interaction. Actually, agents are autonomous, reactive, proactive and intelligent. Agents are capable of initiating action independent of any other entity. Autonomy for agents means that agents are able to react to events occurring in the environment. Proactive agents will actually poll the environment for events and other messages to determine what action they should take. They are not triggered by other agents or by humans. Finally, interaction between agents is richer than the one in object-oriented systems and also more abstract through high-level messages. Interaction in object-oriented systems corresponds to method calls. Interaction between agents is completely different. Messages between objects are simpler since it is only possible to invoke methods and give parameters. These parameters are fixed and if designers want different kinds of parameters, they have to provide as much methods as they have different kinds. Agents use agent communication languages such as KQML [15] or FIPA's ACL [16] to exchange messages. The aim of agent communication languages is to provide a precise

[1] http://dis.cs.umass.edu/research/dvmt.html

syntax and semantics for interaction between agents.

Several other areas where agents and objects differ include scheduling, learning, adaptivity, multiple and dynamic classification or the emergence. As soon as agents have goals, they derive plans to fulfill them. By learning and adaptivity, we mean that agents have the ability to modify their behaviors given events occurring in the environment or to acquire new behaviors based on their observation of other agents or of the environment. Multiple and dynamic classification refer to the ability for agents to have different roles during an execution and to move from one role to another one during it. The emergence within multiagent systems means that an overall behavior emerges from the interactions between agents [34]. We let readers consult [35] [46] for further details and another differences.

Current methodologies used in object-oriented software engineering do not address this richness and the distribution of agents increases the complexity of the problem. Two main directions have been considered by multiagent system designers: extending software engineering (or knowledge engineering) methodologies or defining specific agent methodologies. In the latter, we can find methodologies such as Gaia [46], MaSE [12] or DESIRE [6]. In this paper, we focus on the first direction: extending software engineering methodologies. UML [5] with its extension mechanisms and its metamodel seems to be the right candidate. Moreover, UML is widespread in software community and several industrial-strength tools are available. The use of UML in the design of multiagent systems gives birth to several approaches: Agent UML [36], MESSAGE/UML [8], AOR [43], Tropos [32] or PASSI [7]. This paper attempts to recap what is currently done in the extension of UML for agents and what could be future directions of work. This paper does not claim to do an in-depth comparison of these approaches. This work will be done in the future when approaches try to unify into one unique approach.

Readers have to be aware that the work on UML for agents is an ongoing research and, as a consequence, UML for agents does not claim to be as strong as UML is for software systems.

The remainder of this paper is as follows. In a first part, we briefly present the different approaches: Agent UML in Section 2.1, MESSAGE/UML in Section 2.2, AOR in Section 2.3, Tropos in Section 2.4 or PASSI in Section 2.5.

In the second part (see Section 3), we describe what elements within multiagent systems are already considered through these different approaches. By elements, we mean agents, beliefs, desires, intentions, goals, plans, social structures, etc.

In the third part (see Section 4), we discuss what could be some future directions of work for these approaches.

Section 5 concludes the paper.

2. Current Approaches

Several approaches address the problem of extending UML for agents. This section briefly presents some of them.

2.1. Agent UML

Of the approaches presented here, Agent UML [37] [3] is now one of the most commonly used notations for agent-based modeling. Like other approaches, the aim of Agent UML is to provide a specific version of UML tailored to multiagent system designer needs. Agent UML addresses the analysis and design stages of life cycle. It particularly addresses the question of how representing agents and interaction protocols. Three diagrams are supplied by Agent UML: agent class diagrams [2] [21], sequence diagrams (also called protocol diagrams) [36] [4] [22] and group diagrams [40]. The international association FIPA for the standardization in multiagent systems selects Agent UML sequence diagrams to describe interaction protocols [16].

Some new works now consider the forward engineering and test stages. [23] describes the generation of code for sequence diagrams. [24] [41] [30] [28] are several different approaches to check properties on sequence diagrams either with Promela and SPIN [19] or with Petri nets.

2.2. MESSAGE/UML

MESSAGE (Methodology for Engineering Systems of Software AGEnts) [8] is an agent-oriented software engineering methodology based on UML. The MESSAGE methodology covers multiagent system analysis and design.

MESSAGE defines several views to cope the analysis stage:

Organization View This view shows the agents, the organizations, the roles and the resources used in the system as well as the relationships between these elements. The relationships are aggregation, power and acquaintance. Power relationships refer to a hierarchy between two agents. Acquaintance relationships indicate the existence of at least one interaction involving the entities concerned.

Goal/Task View This shows Goals, Tasks, and dependencies among them.

Agent/Role View This focuses on the individual Agents and Roles. For each agent/role it uses schemata supported by diagrams to its characteristics such as what Goals it is responsible for, what events it needs to sense, what resources it controls, what tasks it knows how to perform, etc.

Interaction View For each interaction among agents/roles shows the initiator, the collaborators, the motivator (generally a goal the initiator is responsible for), the relevant information supplied/achieved by each participant, the events that trigger the interaction, other relevant effects of the interaction.

Domain View This shows the domain specific concepts and relations that are relevant for the system under development.

Several diagrams are provided by MESSAGE: *organization diagram* defines the organization of the system, *goal/task diagram* shows the decomposition of goals into sub-goals, *workflow diagram* describes the task ordering for a particular goal, *delegation structure diagram* presents the delegation between organizations, roles and goals. *Agent/role schema* describes the goals, the capabilities, the beliefs and the requirements for each role. *Interaction diagram* shows the interaction between agents. It presents agents or roles, goals and interactions. Finally, *Domain diagram* provides the different concepts in use in the system. This diagram is usually a UML class diagram.

Several new stereotypes are supplied by MESSAGE to cover notions not present in UML such as roles, organizations, goals, tasks or interactions. The complete list is in [9].

2.3. AOR

AOR (Agent-Object Relationship) focuses on the design of organizations and organization information systems [43]. A specific language is provided to this purpose: AORML (AOR modeling language). In the AORML, an entity is either an agent, an event, an action, a claim, a commitment or an ordinary object.

Two models are considered in AOR: external model and internal model. An external model adopts the perspective of an external observer who is observing the agents and their interactions in the problem domain under consideration. In an internal AOR model, designers adopt the internal (first-person) view of a particular agent to be modeled. This distinction suggests the following system development path: in the analysis stage, draw up an external AOR model of the domain under consideration including one or more focus agents; in the design stage, for each focus agent, transform the external AOR model into an internal one according to the agent's perspective; then, refine the internal AOR model of each focus agent into an implementation model for the target language.

AOR considers three types of agents: artificial agents, human agents and institutional agents. Institutional agents refer to organizations such as banks, hospital or organizational units.

An external AOR model may comprise one or more of the following diagrams:

Agent diagram This depicts the agent types of the domain, certain relevant object types and the relationships among them.

Interaction frame diagram This depicts the action event classes and commitment/claim classes that determine the possible interactions between two agent types.

Interaction sequence diagram This depicts prototypical instances of interaction processes.

Interaction pattern diagram This focuses on general interaction patterns expressed by means of a set of reaction rules defining an interaction process type.

An internal AOR models may comprise one or more of the following diagrams:

Reaction frame diagram This depicts other agents (or agent types) and the action and event types, as well as the commitment and claim types that determine the possible interactions with them.

Reaction sequence diagram This depicts prototypical instances of interaction processes in the internal perspective.

Reaction pattern diagram This focuses on the reaction patterns of the agent under consideration expressed by means of reaction rules.

Several new stereotypes are supplied by AOR to cover notions like agents, events and actions (see the list in [42]).

2.4. Tropos

Tropos [32] is another agent-oriented software engineering methodology based on UML notation. The two key features of Tropos are: (1) the use of knowledge level concepts such as agent, goal, plan and other through all phases of software development and (2) a pivotal role assigned to requirement analysis when the environment and the-system-to-be is analyzed.

The phases covered by the methodology are as follows:

Early requirements: during this phase the relevant stakeholders are identified, along with their respective objectives; stakeholders are represented as actors, while their objectives are represented as goals.

Late requirements: the system-to-be is introduced as another actor and is related to stakeholder actors in terms of actor dependencies; these indicate the obligations of the system towards its environment, also, what the system can expect from actors in its environment.

Architectural design: more system actors are introduced and they are assigned subgoals or subtasks of the goals and tasks assigned to the system.

Detailed design: system actors are defined in further detail, including specifications of communication and coordination protocols.

Implementation: during this phase, the Tropos specification, produced during detailed design, is transformed into a skeleton for the implementation.

Different diagrams are considered within Tropos: UML *class diagrams* with some new stereotypes supplied by Tropos to represent the different actors of the system, Agent UML *sequence diagrams* to represent interactions between actors, once again some new stereotypes supplied by Tropos are used, *plan diagram* to represent plans. The list of new stereotypes is in [32].

2.5. PASSI

PASSI [7] is a step-by-step requirement-to-code methodology for designing and developing multiagent societies. The PASSI methodology is made up of five models concerning different design levels, and twelve steps in the process of building multiagent systems.

The models of PASSI are the following:

System Requirements Model. An anthropomorphic model of the system requirements in terms of agency and target. It comprises four steps: domain description, agents identification, roles identification and tasks specification.

Agent Society Model. A model of the social interactions and dependencies between the agents playing a part in the solution. It comprises four steps: roles identification, ontology description, roles description, protocols description.

Agent Implementation Model. A model of the solution architecture in terms of classes and methods. It comprises two steps: agents structure definition and agents behaviour description.

Code Model. A model of the solution at the code level. It comprises two steps: code reuse and code completion.

Deployment Model. A model of the distribution of the system's parts across hardware processing units, and of their migration. It comprise one step: deployment configuration.

Several diagrams are in use in PASSI: *class diagrams* for ontology description, role description, agent structure definition and code reuse, *use case diagrams* for domain description and agent identification, *sequence diagrams* for role identification and protocol description, *activity diagrams* for task specification, agent behavior specification and code reuse and *deployment diagrams* for deployment configuration.

3. Current Trends

The previous section presented different approaches where UML is extended to represent agents and agent-based systems. In this section, we describe what elements within multiagent systems are already considered in UML or one of its extension described above. By elements, we mean agents, beliefs, goals, plans, interactions, interaction protocols, groups, and so on.

3.1. Requirement Analysis

Requirement analysis is already considered either in UML or in its extensions. Requirement analysis is rendered through use case diagrams. Use case diagrams are central to modeling the behavior of a system, a subsystem or a class. Each one shows a set of use cases and actors and their relationships. A use case is a description of a set of sequences of actions, including variants, that a system performs that yields an observable result of value to an actor [5]. Use case diagrams can be found in the PASSI methodology to model either the domain description or the roles. One can also quote the ROADMAP methodology [26] or in MAS-CommonKADS [25]. Tropos with i^* is the most achieved approach. Tropos considers two steps of requirements: early requirements and late requirements. The late requirement stage corresponds to a more accurate view of the early requirement stage.

3.2. Agents

This notion encompasses two meanings: (1) representing the internal structure of each agent and (2) representing the relationships between agents. Since agents are more complex than objects, one can think that UML class diagrams are no longer sufficient to represent agent structure. A first attempt is in the PASSI methodology. Class diagrams contain a new compartment dealing with plans. AOR is another approach and presents some interests with its specific relationships related to relationships in human organizations [42]. The most achieved solution seems to be the one supplied by Agent UML: Bauer's proposal [2] and its elaboration in [21]. In these two proposals, some new elements are included within class diagrams:

- Agent's name: as classes, agents are identified by a name which could be an instance of parent agent. The agent's name is prepended by the stereotype `<<agent>>` to make the distinction between agents and objects. Actually, agents are frequently implemented as a set of objects or use objects. Such an example is the case of ant society [13] where ants are agents and pheromones are objects.

- Agent's roles: agents play roles in multiagent systems. A role is a class that defines a normative behavioral repertoire of an agent [40][38]. It groups common features such as knowledge, behaviors or data. For instance, for auctions, we have two roles: *Auctioneer* and *Participant*. An agent playing an Auctioneer role tries to sell items to participants at the highest price; an agent playing the *Participant* role tries to buy items at the lowest price.

  Agents in a multiagent systems may have several roles. For instance, an agent may be an auctioneer for one auction and participant for another one.

  This piece of information is also provided in UML but, since agents are richer than objects, a role contains more information. Moreover, agents have the ability of multiple and dynamic classification. It means that agents can have different roles during their executions and they can go from role to role during their executions.

- State: states correspond to attributes used in classes.

- Operation: operations correspond to methods used in classes. However, it is possible to adorn these operations with pre-conditions and post-conditions. It means that the operation is runnable if and only if the pre-conditions are satisfied and if and only if the post-conditions are satisfied. These conditions are related to the agent autonomy. Agents can do tasks if and only if they think that these tasks are of interest to them.

- Capability: this piece of information denotes what actions the agents are able to do. Agents need to coordinate each other to complete their tasks. These capabilities help agents to know what agents can help them for a particular task. Capabilities are written as a free-format text or formally.

- Perception: as stated in introduction, agents are reactive. It means that they react to events occuring in the environment. The list of perceptions can be represented elsewhere in various ways, such as in a statechart. Even if objects can react to events, they are less autonomous since they have to react or if they refuse, such a response has been *traditionally* considered to be an abnormal execution of the program.

- Interaction protocol: interaction protocols (IPs) are another difference between agents and objects. Agent-based systems can employ IPs as a mechanism to guide and constrain agent interactions. Objects can also use an IP mechanism. However, object-oriented systems traditionally practice a synchronous form of interaction that is centrally controlled, whereas agent-based IP typically employ an asynchronous form that is more organic in nature. The list of protocols in this class corresponds to defined elsewhere, which are typically expressed as sequence diagrams or activity diagrams.

- Group: A group is a set of agents that are related via their roles [40]. Agents have the ability to join groups in order to cooperate and coordinate. This capacity is another difference between agents and objects. Objects are traditionally grouped according to a master/slave relationship. Agents can be grouped similar to human ones such as manufacturing cells, organizations, or markets. We consider organizations to be a subtype of group. Here, an organization is defined as a group whose roles and interactions are typically expected to be relatively stable and change slowly over time [40]. For those groups that are not stable or that change often, the notion of organization cannot apply. In this way, then, the concept of group is based on agents, roles, and their interactions - and not necessarily with respect to the volatility of its set.

  This compartment only specifies the name of the group. The two pieces of information *constraints* and *role* gives respectively the constraints that have to be satisfied before entering the group and the role played in this group.

- Service: services are equivalent to the ones defined for classes except that we had conditions. These conditions have to be satisfied in order to apply for these services.

- Knowledge: with groups, protocols and perceptions, knowledge is certainly the most important difference between agents and objects. We propose to represent knowledge as objects. As a consequence, it is easier for agents to update a piece of knowledge if knowledge adopt some specific patterns. Moreover, there exist mutual beliefs within multiagent systems [44]. If beliefs are outside agents, it is easier to share them for several agents.

3.3. Knowledge and Ontology

Cognitive agents encompass knowledge and domain knowledge in order to complete their tasks. An ontology defines the meaning of terms and concepts and describes

the relationships between the elements [14]. Ontologies are practical to deal with heterogeneous agents. Then, agents which do not share the same vocabulary can interact each other if they are able to know how to translate words from one language to another one. This feature is interesting in a context of interoperability.

MESSAGE proposes to represent domain concepts and knowledge through class diagrams. Another approach is in PASSI where domain description is done through use case diagrams. In our opinion, the most achieved approach is the one from Cranefield [11] where a piece of knowledge is described as an object.

Such an approach is interesting in a context of mutual or shared beliefs. If knowledge is located outside agents, it is easier to share it.

3.4. Agent Behavior

Agents are reactive and act provided events in the environment or coming from other agents. Agents' behaviors can be represented through statecharts. A statechart diagram shows a state machine [18], consisting of states, transitions, events and activities. Statechart diagrams are used to illustrate the dynamic view of a system. They are especially important in modeling the behavior of an interface, class or collaboration. Statechart diagrams emphasize the even-ordered behavior of an object, which is especially useful in modeling reactive systems.

Each approach provides its specific statechart diagram or related statechart diagram to model agents' behaviors. An interesting approach is in [1] used to model robotic agents.

3.5. Goals/Tasks/Plans

Cognitive agents use desires and intentions. Intentions are described through goals. These goals give birth to plans as soon as they become executable. Activity diagrams seem to be the right candidate to represent goals, tasks or plans. An activity diagram shows the flow from activity to activity within a system while statechart diagrams show the flow from state to state. An activity diagram shows a set of activities, the sequential or branching flow from activity to activity, and objects that act and are acted upon. Activity diagrams are especially important in modeling the function of a system.

The PASSI approach to model goals and plans is similar to the one found in UML. The MESSAGE approach is related to a second diagram called workflow diagram describing the task ordering for a particular goal.

3.6. Groups

Agents in multiagent systems are grouped as sets of agents related via their roles. This notion of groups is not present within object-oriented systems. As a consequence, no diagram is supplied by UML for this purpose.

Several proposals are available: Parunak's and Odell's proposal [40] where a diagram gives the agents, their roles in the groups and the relationships between these agents (or roles). Another solution is the one from AOR [42]. This proposal is interesting since it proposes several new relationships between agents in comparison with the one provided by UML.

Even if groups are already considered through UML based diagrams, they are still immature. The number of information is less numerous than in electronic institution organizations such as Fishmarket [33].

3.7. Interaction Protocols

At this time, interaction protocols are certainly one of the most popular areas that require extending UML for agents. First works on this extension were on the description of interaction protocols [37] [3]. It is not surprising since interaction is one of the main differences between agents and objects. Interaction protocols are represented by sequence diagrams.

A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. A sequence diagram shows a set of objects and the messages sent and received by those objects. The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components or nodes.

Two solutions are currently considered: Tropos's proposal [32] where sequence diagrams are extended to describe the link to plans. The Agent UML proposal is the most achieved one [37] [3].

Since agents are represented by their roles, this piece of information replaces the information provided within the lifelines at the top of the diagram as shown on Figure 1. The general form of describing agent roles in Agent UML is:

instance-1. . .instance-n / role-1. . .role-m : class

denoting a distinguished set of agent instances *instance-1. . .instance-n* satisfying the agent roles *role-1. . .role-m* with n,m $\geq$ 0 and *class* it belongs to. Instances, roles or class can be omitted, in the case that the instances are omitted, the roles and class are not underlined.

Agents are more flexible for their interactions. It means that they are able to select a path in the interaction among others given their goals, intentions and beliefs. Some new connectors appear in Agent UML sequence diagrams to tackle this point (shown on Figure 2).

Three connectors are supplied for these features (shown on Figure 2). The connector AND is rendered as a thick vertical line as shown on Figure 2a. It means that messages

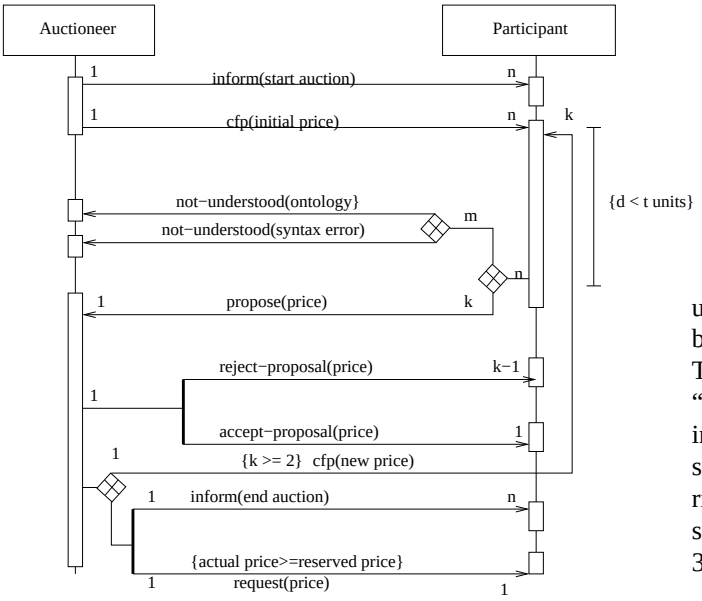

Figure 1. English Auction Protocol

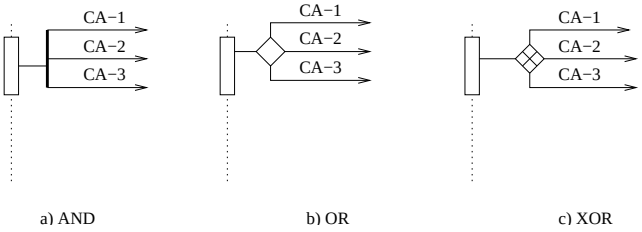

Figure 2. Agent UML Connectors

have to be sent concurrently. On Figure 2a, *CA-1, CA-2, CA-3* are sent in parallel. The connectors OR is rendered as a diamond as shown on Figure 2b and XOR is rendered as a diamond and a cross within it as shown on Figure 2c. They mean that a decision between several messages has to be done. When considering the connector OR, zero or several messages is chosen: a subset of the set $\{$*CA-1, CA-2, CA-3*$\}$. In the case of several messages are taken, the messages are sent in parallel. The connector XOR also represents a decision but in this case, one and only one message is chosen, it is either *CA-1* or *CA-2* or *CA-3*.

As stated in the introduction, agents send message asynchronously. Messages in interaction are sent usually asynchronously (the symbol with a stick arrowhead as shown on the Figure 3a). It shows the sending of the message without yielding control. It is also possible to sent messages synchronously (the symbol with a filled solid arrowhead as shown on the Figure 3b). It shows the yielding of the thread of control (wait semantics), i.e. the agent role waits

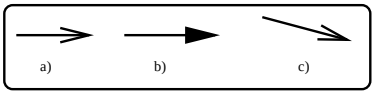

Figure 3. Types of Message Delivery in Agent UML

until an answer message is received and nothing else can be processed. Normally, messages are drawn horizontally. This indicates the duration required to send the message is "atomic", i.e. it is brief compared to the granularity of the interaction and that nothing else "happen" during the message transmission. If the messages require some time to arrive, for instance for mobile communication, during which something else can occur. The message is shown on Figure 3c.

The main differences between UML sequence diagrams and Agent UML sequence diagrams are the fact that messages are sent asynchronously and that agents can choose the path in the interaction. This choice is done according to their beliefs, intentions, goals or conditions.

This list is extended in [22] with some new features such as time, broadcast, synchronization, triggering actions and exceptions.

4. Future Directions

Previous sections described the current use of UML for agents and multiagent systems. The elements presented do not cover all the needs that multiagent system designers have. This section attempts to give an agenda of future work on UML for agents.

Some elements within agents or multiagent systems are not yet considered. For instance, the notion of planning is still immature. Designers need to represent the plan but also the decomposition of this plan by agents, the conditions to be satisfied to execute each part of the plan, etc. Activity diagrams are the right candidate, but they need to be enhanced for this purpose.

Even if some preliminary work is available around mobility [27], this notion has to be extended to several kinds of diagrams. Deployment diagrams are not the only ones that have to deal with mobility issues. Activity diagrams can be enhanced to express mobility by employing UML swimlanes that represent agent platforms. Moreover, the notion of mobility has to appear on class diagrams when designing agents and the relationships between agents or between agents and objects.

Few works consider the whole life cycle. It is important to develop the implementation and test stages. Forward engineering is only considered for Agent UML sequence dia-

grams in [23] but remains incomplete and this work does not consider all the features present in Agent UML sequence diagrams. A second approach is in PASSI [7] through code reuse library.

Testing and quality of service (QoS) are also issues. Several work are presented in 2002 on this subject: two papers [24] [28] propose to derive directly a program in PROMELA [19] [20] and using the model checker SPIN. Two other papers [41] [30] propose to derive Agent UML sequence diagrams into Petri nets, then to check properties on these Petri nets.

The main pitfall in using Agent UML to design multiagent systems is there are no tools dedicated to Agent UML. Moreover, it is not easy to extend UML tools to exploit UML extended diagrams. Wagner proposes a template for Visio for his AOR models. Burrafato and Cossentino propose an add-in for Rational Rose.

Another direction of work is to extend XMI [39] in order to store UML-based representations and to exchange them between users and tools.

A current problem for UML-based extensions is the lack of formal semantics. It is crucial that a common understanding appears for each stereotypes or elements in UML based extensions.

Finally, the most promising work is certainly trying to merge all these approaches into a unified modeling language that is coherent and consistent. Some propose methodological aspects, some propose new diagrams or stereotypes. For the moment, the overlapping is not too important. As a consequence, it seems possible to accomplish this work in the near future.

5. Conclusion

Agents and multiagent systems are now viewed as a serious approach for designing open, complex systems such as electronic commerce applications. Several methodologies exist in multiagent system literature, but their main drawback is that users need to learn how to use it. For some of them, a formal background is required. All these points slow down the adoption by software engineering users. A new approach appeared recently that proposes to make profit of software engineering. This new approach proposes to make profit of the extensibility of UML. Several UML based extensions are available in the literature: Agent UML [36], MESSAGE/UML [8], AOR [43], Tropos [32] or PASSI [7] among others.

This paper presented a state of the art of the use of UML for agent and multiagent system design. Then, it describes what could be the future directions for these approaches. The most important seems to be the definition of a unique UML extension using the features of these different approaches.

Acknowledgements. Marc-Philippe Huget thanks the UK government for its support through the EPSRC project GR/R27518 (Verifiable Languages and Protocols for Multiagent Systems).

References

[1] T. Arai and F. Stolzenburg. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, volume 1, pages 11–18, Bologna, Italy, July 2002. ACM Press.

[2] 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.

[3] B. Bauer, J. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering (AOSE-00)*, 2000.

[4] B. Bauer, J. P. Müller, 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.

[5] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1999.

[6] F. Brazier, B. Dunin-Keplicz, N. R. Jennings, and J. Treur. Formal specification of multi-agent systems: a real-world case. In V. Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 25–32, San Francisco, CA, 1995. MIT Press.

[7] P. Burrafato and M. Cossentino. Designing a multiagent solution for a bookstore with the PASSI methodology. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Toronto, Canada, May 2002.

[8] C. Caire, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using MESSAGE/UML. In *Proceedings of Agent-Oriented Software Engineering (AOSE 01)*, Montreal, Canada, May 2001.

[9] C. Caire, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet. Message: Methodology for agent-oriented software engineering. Technical report, Eurescom, 2001. Deliverable 3.

[10] S. Cammarata, D. M. Arthur, and R. Steeb. Strategies of cooperation in distributed problem solving. In A. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 102–105. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[11] S. Cranefield. Networked knowledge representation and exchange using UML and RDF. *Journal of Digital Information*, 1(8), 2001.

[12] S. A. DeLoach. Multiagent systems engineering: a methodology and language for designing agent systems. In

Proceedings of Agent Oriented Information Systems '99 (AOIS'99), pages 45–57, Seattle, USA, May 1999.

[13] A. Drogoul. *De la simulation multi-agents à la résolution collective de problème. Une étude de l'émergence de structures d'organisation dans les systèmes multi-agents*. PhD thesis, Universit Paris 6, 1993.

[14] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2001.

[15] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Third International Conference on Information and Knowledge Management (CIKM-94)*. ACM Press, 1994.

[16] FIPA. *Specifi cation*. Foundation for Intelligent Physical Agents, http://www.fi pa.org/repository/fi pa2000.html, 2000.

[17] M. S. Fox, J. G. Chionglo, and M. Barbuceanu. The integrated supply chain management system. Technical report, Enterprise Integration Laboratory, University of Toronto, 1993.

[18] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[19] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[20] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5), May 1997.

[21] M.-P. Huget. Agent UML class diagrams revisited. In B. Bauer, K. Fischer, J. Muller, and B. Rumpe, editors, *Proceedings of Agent Technology and Software Engineering (AgeS)*, Erfurt, Germany, October 2002.

[22] 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.

[23] M.-P. Huget. Generating code for agent UML sequence diagrams. In B. Bauer, K. Fischer, J. Muller, and B. Rumpe, editors, *Proceedings of Agent Technology and Software Engineering (AgeS)*, Erfurt, Germany, October 2002.

[24] M.-P. Huget. Model checking Agent UML protocol diagrams. In *ECAI Workshop on Model Checking Artifi cial Intelligence (MoChArt)*, Lyon, France, July 2002.

[25] C. Iglesias, M. Garrijo, J. Gonzales, and J. Velasco. Design of multi-agent system using MAS-CommonKADS. In Springer-Verlag, editor, *Proceedings of ATAL 98, Workshop on Agent Theories, Architectures, and Languages*, volume LNAI 1555, pages 163–176, Paris, France, July 1998.

[26] T. Juan, A. Pearce, and L. Sterling. Extending the Gaia methodology for complex open systems. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 02)*, Bologna, Italy, July 2002. ACM Press.

[27] C. Klein, A. Rausch, M. Sihling, and Z. Wen. Extension of the Unifi ed Modeling Language for mobile agents. In K. Siau and T. Halpin, editors, *Unifi ed Modeling Language: Systems Analysis, Design and Development Issues*, chapter 8, pages 116–128. Idea Publishing Group, 2001.

[28] J.-L. Koning and I. Romero-Hernandez. Generating machine processable representations of textual representations of AUML. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *AAMAS Workshop on Agent-Oriented Software Engineering (AOSE)*, Bologna, Italy, July 2002.

[29] B. LeBaron. Agent based computational fi nance: Suggested readings and early research. Technical report, Brandeis University, 1999. o appear in the Journal of Economic Dynamics and Control.

[30] H. Mazouzi, A. El Fallah Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy, July 2002.

[31] J. Müller, B. Bauer, and M. Berger. Software agents for electronic business: Opportunities and challenges. In V. M. et al., editor, *Proceedings of MASA 2001*, number 2322 in LNAI, pages 61–106. Springer, 2001.

[32] J. Mylopoulos, M. Kolp, and J. Castro. UML for agent-oriented software development: the tropos proposal. In *Proceedings of the Fourth International Conference on the Unifi ed Modeling Language (UML 2001)*, Toronto, Canada, October 2001.

[33] P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autnoma de Barcelona, 1998.

[34] J. Odell. Agents and complex systems. *Journal of Object Technology*, 1(2), July-August 2002.

[35] J. Odell. Objects and agents compared. *Journal of Object Computing*, 1(1), May 2002.

[36] 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 Artifi cial Intelligence*, Austin, Texas, july, 30 2000. ICue Publishing.

[37] J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Proceedings of First International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, june, 10 2000. Springer-Verlag.

[38] J. Odell, H. V. D. Parunak, and M. Fleischer. Designing effective agent organizations: Using roles. In A. Lucena, F. Zambonelli, A. Omicini, and J. Castro, editors, *Software Engineering for Large-Scale Multi-Agent Systems*. Springer-Verlag, 2002.

[39] OMG. XMI 1.1. Technical Report 01-03-10, OMG, 2001.

[40] H. V. D. Parunak and J. Odell. Representing social structures in UML. In M. Wooldridge, G. Weiss, and P. Ciancarini, editors, *Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, LNCS, Montreal, Canada, May 2001. Springer-Verlag.

[41] D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy, July 2002.

[42] G. Wagner. The agent-object-relationship metamodel: Towards a unifi ed conceptual view of state and behavior. *Information Systems*, 2002. to appear.

[43] G. Wagner. A UML profi le for external AOR models. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Proceedings of Third International Workshop on Agent-Oriented Software Engineering (AOSE-2002)*, Bologna, Italy, July 2002.

[44] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[45] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons, April 2002.

[46] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.