Applying AUML and UML 2 in the Multi-agent Systems Project

Conference Paper *in* Lecture Notes in Computer Science · November 2009

DOI: 10.1007/978-3-642-04947-7_14 · Source: DBLP

2 authors:

Gilleanes Guedes
Federal University of Pampa
12 PUBLICATIONS **36** CITATIONS
SEE PROFILE

Rosa Maria Vicari
Federal University of Rio Grande do Sul
344 PUBLICATIONS **2,499** CITATIONS
SEE PROFILE

Applying AUML and UML 2 in the Multi-agent Systems Project

Gilleanes Thorwald Araujo Guedes and Rosa Maria Vicari

Instituto de Informática
Programa de Pós-Graduação em Computação (PPGC)
Universidade Federal do Rio Grande do Sul (UFRGS) - Porto Alegre - RS - Brasil
gtaguedes@inf.ufrgs.br, rosa@inf.ufrgs.br
http://www.inf.ufrgs.br/pos

Abstract. This article discusses the viability of the AUML and UML languages employment, from the latter's version 2.0 on, in the multi-agent systems project. In this article some works that have used UML for the project of systems that involved agents, as well as some AOSE (Agent Oriented Software Engineering) methodologies that use in some way UML or AUML (or both), are presented. Immediately afterwards the article approaches the AUML language, highlighting the innovations proposed by same and how it can be applied to the multi-agent systems project, identifying its advantages and disadvantages. After that, the paper passes on to describe how UML, from its version 2.0 on, has bypassed AUML and how the former can be applied to the multiagent systems project, pinpointing its positive aspects and its deficiencies.

Keywords: AUML, UML 2, Agents, Use Case Diagram, Actors, Internal Use Cases, Sequence Diagram, Combined Fragments, State Machine Diagram, Composite States, Activity Diagram. Activity Partition.

1 Introduction

Along the years, researchers in the area of software engineering have endeavored to set up methods, techniques, and modeling languages/notations with the goal of creating formal software project patterns while establishing well-defined steps for software building in such a way as to make their development more robust, faster, organized, coherent, trustworthy, easier to maintain and re-use, and presenting better quality. At the same time, in the area of Artificial Intelligence, the employment of intelligent agents as auxiliary aids to software applied to the most diverse dominions is being spread out. This practice has shown to be a good alternative for the development of complex systems, fostering a great increase of agent-supported software development in the several areas.

However, the development of this kind of system has presented new challenges to the software engineering area and this led to the surfacing of a new sub-area, blending together concepts brought over from both the software engineering and artificial intelligence areas, which is known as the AOSE - Agent Oriented Software Engineering, whose goal is that of proposing methods and languages for projecting and modeling agent-supported software. Among the several AOSE methods nowadays extant MaSE, MessageUML, Tropos, and Prometheus can be named. For an example of modeling language we can mention AUML - Agent Unified Modeling Language - that was derived from the UML, though it is possible to find jobs that employ UML proper, that is becoming particularly attractive for multi-agent system projects from its 2.0 version.

In this paper a discussion about the employment of the modeling languages, AUML and UML from its version 2.0, will be presented as an aid for the multi-agent systems project. Firstly, some works that have used UML for the project of systems that involved agents, as well as some AOSE methodologies that use in some way UML or AUML (or both), are presented. The article approaches the AUML language, highlighting the innovations proposed by same and how it can be applied to the multiagent systems project, identifying its advantages and disadvantages. After that, we describe how UML, from its version 2.0 on, has bypassed AUML and how the former can be applied to the multiagent systems project, pinpointing its positive aspects and its deficiencies.

2 UML, AUML and UML 2

UML has become a standard for software modeling and is broadly accepted and used throughout software engineering industry. Therefore, needless to say, many agent-supported software projects use directly the UML language for the designing of this kind of software, for instance [7], [8], [9] and [10]. In addition, some AOSE methods, like MaSE [3] or Tropos [5] employ UML in a partial fashion. For more details about UML see [11] or [12].

2.1 AUML - Agent Unified Modeling Language

Since software agents present specific features in comparison with more traditional software methods, some attempts to adapt UML to these characteristics were made, which brought forth the surfacing of AUML - Agent UML - whose main documentation can be seen in [13].

All the same, the AUML language currently supports only weak notions of agency, representing agents as objects, while employing state-machine diagrams to model their behavior and extended Interaction diagrams to model their communicative acts [1], not support cognitive or social abstractions. Caire in [4] comments that, although this notation is useful, it bears no agent concept in its core, stating also that specifying the behavior of an object in terms of interaction protocols will not change such an object into an agent.

One of the main AUML contributions is the document on interaction diagrams to be found in [13], where it is attempted to extend UML diagrams toward the supporting of communicative acts and the modeling of inter-agent communication. In its original format, this document also proposed some notation for multiple choice representation and parallelism.

However, as can be seen in [14] and in [13] itself, whose document was updated, this symbology was set aside in favor of an alternative proposed by UML 2 itself, as can be seen in [11] and [12], where resources such as the use of combined fragments of both types 'par' (as in <parallel>) and 'alt' (as in <alternative>) was employed in the sequence diagram. Realizing the superiority of the new UML version, [14] has then proposed the adaptation of this new notation to AUML, adding some small innovations to the interaction diagrams, as the lifeline notation for agent representation (as objects). Nevertheless, it is important to highlight that AUML interaction diagrams seem to be the most employed by AUML-based AOSE methods.

Peres in [15] accentuates the existence of scarce documentation on AUML and an even smaller number of updating steps, while also stating that those papers to be found on the language employment always repeat the very same examples presented in the original documentation. The paper also highlights that the diagram definitions are not very accurate into their establishing the degree into which the UML paradigm is to be followed or broken in relation to the multi-agent system particulars. Still, [15] appends that a UML extension will not be sufficient to create an agent-oriented modeling language, because agents need more abstractions and a semantic treatment to focus on their own particular features.

The AUML is currently inactivated, according to [16], where a note informs us that this occurs for three reasons, to wit, first, the launching of UML 2.1, containing many of such agent-related features that AUML required. Secondly, the release of the language, SysML - System Modeling Language - prepared to customize UML, another item to supply many of the characteristics searched by AUML. Thirdly, the UML Profile and Metamodel for Services (UPMS) RFP, that is under elaboration requires a services metamodel and profile for extend UML by means of capacities applicable to service modeling with the employment of SOA (Service Oriented Architecture). All these emerging standards have the stated goal of including agent-related concepts.

Even so, as mentioned above, several methods are trying to apply AUML/UML somehow into their life cycles, especially as related to interaction diagrams, as in MESSAGE/UML [4], Tropos [5] and Prometheus [6]. However, according to [1], many of the AOSE methods present weak notions for agency, never focusing in a satisfactory fashion such cognitive abstractions as beliefs, desires, and intentions, as well as social abstractions like cooperation, competition, and negotiation. In addition, these methods possess strong associations with object-oriented software engineering methods, oftentimes dealing with agents as if they were objects.

The representation of agents as objects conflicts with the definition of agents for, according to [1], 'an agent is a computational process located within an environment and designed to achieve a purpose within said environment by means of autonomous, flexible behavior'.

To adhere to this concept, for cognitive applications an agent should not be represented by classes alone, as happens in AUML, but as an actor able to interact with the system. Probably, along the processes in which it was supposed to partake, an agent would be expected to interact with classes, whose objects store their own knowledge, among other things, but the agent in itself should not be an object instantiated from a class.

The AUML proposal might be valid, however, whenever active objects were represented as agents. Active objects could be compared to reactive agents that, as can be seen in [2] can only react to events and don't own any model of the world in which they are inserted, all the while cognitive agents are endowed with some knowledge about their environment, own mental states, like beliefs, desires, and intentions, and can set up communications with other agents to negotiate help toward achieving their goals. This kind of agents cannot be deal with simply as if they were objects.

2.2 UML 2

As quoted in the prior section, agents can be represented as actors, like happens in [7] and [9]. The latter states that agents can also be represented as actors in the UML use cases diagram and that associations among actors and use cases can be employed to represent perceptions and actions of an agent represented by an actor. However, [9] does not maintain the agent's representation as an actor in the sequence diagram, in which agents are represented as objects, according to AUML proposal in [13]. Agents are already represented by [10] as a bland of actor and object, employing objects with the stereotype control, that modifies the object standard design into a circle shaped by an arrow and inserting within this circle the actor's symbol. Therefore, [10] represents agents as a kind of control-type object, only with features particular to agents. There still is a notation proposed by [17] who suggests modifying the standard design the component, actor, to represent agents, presenting same as square-headed actors. [17] even goes as far as suggesting the creation of cloud-like component to represent the agents' goals. All the same, as [17] identifies active objects as agents, these are modeled like objects in the sequence diagram, as happens in AUML.

In UML, actors are usually modeled, within the use case diagram, as external to the system, for they interact with the system, but cannot be a part of it. Therefore, actors are placed outside the system border, as can be seen in [12].

Anyway, most of the times, the software agents are not external to the software, rather they customarily are inserted in the system environment and, as they are independent, proactive, and able to interact with the software according to their goals, therefore these should be represented as actors. As can be seen in [11], an actor models a type of role played by an entity that interacts with the subject but which is external to the subject In that way, for a multiagent systems project, as states [9], it is necessary to adapt this concept, considering that agents can be internal to the system, that is, an agent can be a part of the subject and, therefore, if we are to represent agents (their roles) as actors, said actors should be internally represented within the system's borders, for they belong in the software, as shows Figure 1. The system boundary is represented by a square the involves the system's functionalities. In the example above, the borderline represents a system of research through Internet. We can notice that there is an external actor who represents a human Internet user, to wit, a real

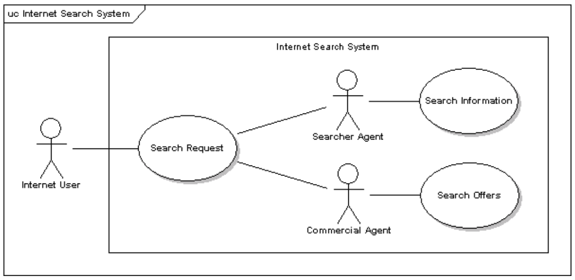

Fig. 1. Example of actors representing agents - based on [12]

person who interacts with the system by means of the use case "Search Request". We can also notice that there are two more actors named "Searcher Agent" and "Commercial Agent" and these are within the system's border because they represent software agents, as occurs in [9] and [17]. "Searcher Agent" is responsible for performing the search required by "Internet User", all the while "Commercial Agent" is responsible for seeking offers associated to the performed research. We can also observe that there are two use cases internal to the system, which represent the processes of search for information and search for offers and these use cases can only be employed by agents internal to the system and are not accessible by the external agents.

The process "Search Request" can be detailed by means of a sequence diagram, as demonstrates Figure 2 (based on [12]). Other than in the AUML approach [13], instead of representing agents as objects, we chose to represent them as actors, in the same way they were represented in the use case diagram. We believe this is more correct in relation to the definition for agent herein adopted and, besides, this keeps coherence with the use case diagram and allows to better differentiate the agents from the real objects instantiated from the class diagram, for in the AUML sequence diagram those objects that represent agents only differ from normal objects by textually identifying the agent's name and the role represented by it in the format "agent/role", instead of being simply shown by the object's name.

The justification for the employment of actors instead of objects, besides that of the concept of agent itself described in [1], is based upon the concept of an actor as described in [11], where is stated that an actor represents a role played by some entity and that an entity may play the role of several different actors and an actor may be played by multiple different entities. Although the concept of role used by UML might be different than the concept of role used in multi-agent systems, they seem to be, at first sight, similar to each other. Thus, the actors seem able to represent the roles of an agent when this interprets more than one. The employment of objects to represent agents could be valid in those projects that contain only reactive agents, that could be taken as active objects; anyway, in such situations that contain cognitive agents, we believe that the representation of same as actors would be more correct.

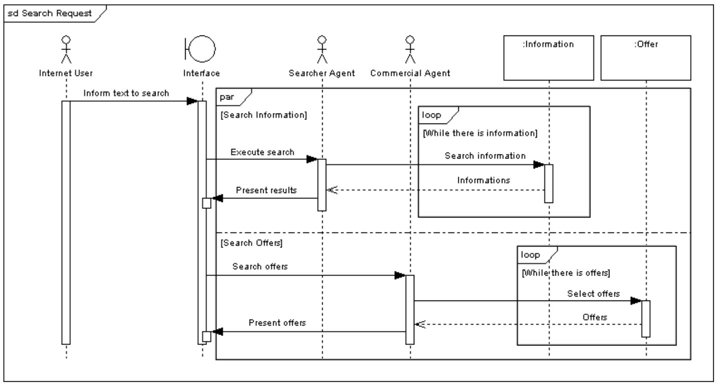

Fig. 2. Detailing of the “Search Request” Process by means of a Sequence Diagram

The diagram presented in Figure 2 also spans those internal use cases that were presented in the Figure 2. They could be detailed in separate diagrams, but the example seems to be more complete and understandable the way it is. In this process, the actor, ”Internet User”, supplies the text to be researched to the interface and this re-passes same to the actors/agents ”Searcher Agent” and ”Commercial Agent”, that, as modeled in the use cases diagram, are internal to the system and the event, ”Inform text for search” is forwarded to them through the system’s interface. ”Searcher Agent” is responsible for the search of information related to the required research, while ”Commercial Agent” is responsible for seeking offers related to said research, like products, books, or courses, for example. Observe that both tasks are performed in parallel and the results are presented simultaneously. This is possible to represent by means of the employment of a type ”par” (as in ”parallel”) combined fragment that demonstrates that concurrence situation within which two agents execute their tasks at the same time. Notice that an interrupted line separates those operations performed by each agent. Each parallel process’s acting area is called, interaction operand. Further observe that each interaction operand has a guardian condition to establish its function. Obviously, in a real system there would be included many more classes to represent those information pools that are needed for this kind of research.

The same process can be represented in the state machine diagram below, obviously under a different focus, by means of composite states with orthogonal regions, as shown in the Figure 3 (based on [12]). In the example shown in Figure 3, the process begins by a static state that waits up the user until he types a text for researching. The text for research insertion event generates a transition into a composite state which contains two orthogonal regions. In the first region, like informs its title in the shape of a guard condition, is done the search for information relevant to the research, while in the second region the search for offers related to the looked-for text is performed. The states in each region occur in parallel. The same process can be even more detailed by

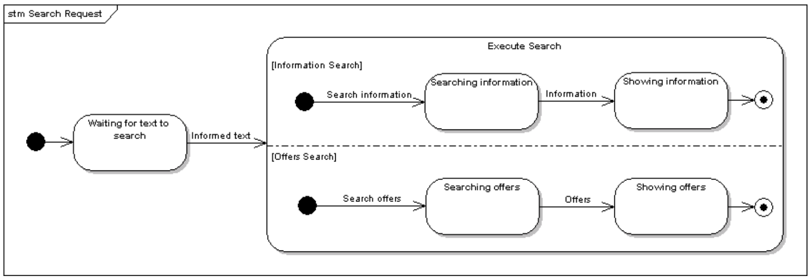

Fig. 3. State Machine Diagram employing States composed by Orthogonal Regions

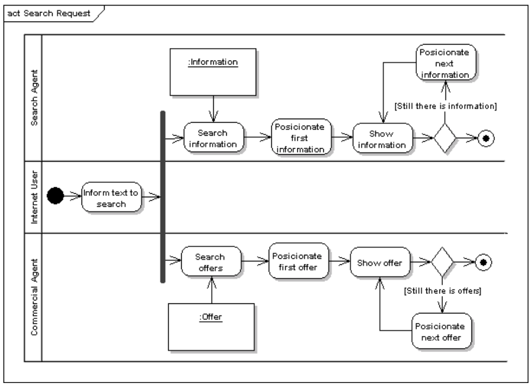

Fig. 4. Activity Diagram employing Activity Partitions

means of the activity diagram, which is an extremely detailed diagram which can be used for the modeling of plans, as state [9] and which is already employed in the Tropos methodology [5]. This parallelism can also be represented through the use of activity partitions and fork/join nodes, as demonstrates Figure 4.

In the example shown by that Figure each actor/agent was represented as an activity partition, something that can also be seen in [8]. The process is initiated when "Internet User" supplies the text to be researched. From that point, the flow is cut by a fork node and divides into two parallel flows, one to each activity partition, where information and offers related to the research performed by the actor will be looked for. For an obvious matter of physical space, this diagram was simplified.

3 Conclusions

AUML, in its present state of development, is still to be found within a rather tentative phase, therefore showing no great innovations in relation to the language from which it was derived. The biggest AUML contribution is an initial proposal for interaction diagrams, whose main original contribution was that of making some changes into the sequence diagram for the representation of parallelism by means of threads, also used in situations of choosing an option over others. This representation used a symbol to multiplex processing into several threads or to represent flow alternatives. However, this notation was abandoned face the notation newly adopted by UML 2, that came to employ combined fragments, a system that looks much more practical and supple.

Considering it is hard to pinpoint palpable differences between AUML and UML and noting how the latter presented great innovations from its version 2.0 on, besides the fact the AUML Project itself acknowledged that many of its goals were achieved by the last few innovations within UML and SYSML, it would seem that UML 2 is the most adequate for multi-agent system projects. However, some authors believe that both UML and AUML are too concerned with object orientation, a feature that would not allow it to be wholly used for multi-agent systems modeling. Maybe UML is adequated to the project of systems in which agents present more reactive characteristics than cognitive ones, but it is possibly insufficient for multi-agent system projects working with BDI architectures, which will require strong agency notions and, if this be proved, there will come up the need to adapt the language for said purpose.

The representation of agents/roles as actors, both in the cases of use diagrams and the sequence diagrams seem to be more correct than their representation as objects, as suggested by AUML. The concept of actor presented by UML is much closer to the concept of agent as the interpreter of a role than the concept of object. The representation of agents as objects would become valid when these will be active objects, but will show itself insufficient when dealing with cognitive agents. Besides, for a matter of coherence, if agents/roles were to be represented as actors in a diagram, like that of use cases, the former should remain so represented in other diagrams, whenever this becomes possible.

There is also the issue of goal representation, much necessary in multi-agent system projects. Goals can be defined as an agent's wishes, that is, whatever he wants to achieve. UML owns no specific constructions for this type of representation, to wit, there is no component to identify a goal to be reached by one or more agents. The closest to this would be the use case component, but any use case would identify a system functionality, that is, a function, service, or task the system is expected to supply, something that is not exactly a goal the way this concept is understood within the area of multi-agent systems. A functionality can represent a goal, but can contain more than one, depending on the situation. Anyway, it might be possible to create a stereotype dubbed, for example, <<goal>>, to identify such use cases that might be considered as goals, other than normal use case and, in that situation in which the goal had sub-goals, another stereotype could be created for this, like <<sub-goal>> and identify these by means of specialization/generalization associations with the main goal. The employment of stereotypes is perfectly valid in UML and its function is precisely that of allowing flexibility to the language, making it possible to attribute new characteristics and functions to already extant components.

The use of stereotypes could also be useful to establish differences between normal actors and agents, by the employment of a stereotype dubbed, for example, <<agent>>. As there is the happenstance that an agent being able to interpret many roles, another stereotype, named <<role>> or <<agent role>> could be created to determine when an actor represents a role interpreted by an agent. In the eventuality of being needed to identify which roles are identified by an agent it would be feasible, within a separate diagram, to create a role hierarchy, where the agent would be identified at the top of the hierarchy and be represented as an actor containing the stereotype <<agent>>, while its roles would be sub-actors, associated to the agent by means of generalization/specialization associations and spanning the stereotype <<agent role>>. This would not stop more than an agent to interpret the same role. Anyway, both the question of stereotype usage for the identification of goals and that for the identification of roles are still to be verified on depth so as to determine whether this alone would be sufficient for this purpose.

Another issue would be that of how to represent the agents' beliefs. Beliefs represent the information agents have about the environment and about themselves. Perhaps it would be feasible to store such information into classes, but it is not still clear how this could be achieved, for beliefs cannot be defined either as attributes or as methods, which are the two kinds of information a class usually contains. To represent this kind of information, we could try to create a profile, like the data-modeling profile, used to map classes on tables, where the <<table>> stereotype is used to represent a class the same way as it were a table and the <<column>> stereotype to represent attributes as colums.

Anyway, it might be possible to create one or more classes that spanned a single attribute (a string-like one, for example) that would simply textually store the agent's beliefs with a single instance to store each of his beliefs. Besides, some languages which implement beliefs usually define same directly in the software codes, without their being stored within a repository. If we were to employ a class diagram to identify the beliefs presented by an agent, it would be necessary to create some sort of mapping to connect their representation in the class diagram and the way they were expected to be implemented.

It seems to be possible to represent communication between agents by means of a sequence diagram, where messages would stand for communicative acts, as is suggested by AUML, but representing the agents/roles as actors and not as objects. Also negotiations, that are the way that agents reach an agreement to help each other so as to further a goal, can be represented by means of messages and combined fragments. Agents collaboration and competition features can be equally represented by state machine diagrams and activity diagrams, employing respectively composite states with orthogonal regions and activity partitions. However, deeper studies would be necessary into the application of said diagram, so as to demonstrate whether it is actually sufficient for this kind of representation.

Finally, we conclude that UML, for all that this is methodology-independent, can be useful in a multi-agent system project when used and possibly adapted to an AOSE methodology specifically oriented toward the development of agent-supported software, in which there are to be defined those UML diagrams that can be shown useful to the methodology and in which moment they are supposed to be applied, as well as which possible adaptations should be performed.

References

1. Vicari, R.M., Gluz, J.C.: An Intelligent Tutoring System (ITS) View on AOSE. International Journal of Agent-Oriented Software Engineering (2007)
2. Alvares, L.O., Sichman, J.S.: Introdução aos sistemas multiagentes. In: Jornada de Atualização em Informática (JAI 1997). cap. 1. Medeiros, C. M. B., Brasília (1997)
3. Deloach, S.A.: Analysis and Design using MaSE and agentTool. In: 2nd Midwest Artificial Intelligence and Cognitive Science Conference, Oxford (2001)
4. Caire, G.: Agent Oriented Analysis Using Message/UML. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, p. 119. Springer, Heidelberg (2002)
5. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems 8, 203–236 (2004)
6. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In: Giunchiglia, F., Odell, J.J., Weiss, G. (eds.) AOSE 2002. LNCS, vol. 2585, pp. 174–185. Springer, Heidelberg (2003)
7. Coelho, N.P., Guedes, G.T.A.: Tutorial Hipermídia Aequator utilizando um Agente Estacionário Reativo Simples: Apresentação da Modelagem. In: 11° Congresso de Informática e Telecomunicações SUCESU-MT. Cuiabá (2006)
8. Kang, M., Wang, L., Taguchi, K.: Modelling Mobile Agent Applications in UML 2.0 Activity Diagrams. In: 3rd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, pp. 104–111. IEEE Press, Edinburgh (2004)
9. Bauer, B., Odell, J.: UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard. Journal of Engineering Applications of AI (2005)
10. Dinsoreanu, M., Salomie, I., Pusztai, K.: On the Design of Agent-Based Systems using UML and Extensions. In: 24th International Conference on Information Technology Interfaces (ITI 2002), Cavtat, Croatia, pp. 205–210 (2002)
11. OMG - Object Management Group. Unified Modeling Language: Superstructure Specification - Version 2.1.1. OMG (2007), http://www.omg.com
12. Guedes, G.: UML 2 - Uma Abordagem Prática. Novatec Editora, São Paulo (2009)
13. Huget, M.P., et al.: Interaction Diagrams. Working Documents. AUML Official Website (2003), http://www.auml.org/
14. Huget, M.P., Odell, J.: Representing Agent Interaction Protocols with Agent UML. In: Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), New York City (2004)
15. Peres, J., Bergmann, U.: Experiencing AUML for MAS Modeling: A Critical View. In: Software Engineering for Agent-Oriented Systems, SEAS (2005)
16. AUML Official Website, http://www.auml.org
17. Depke, R., Heckel, R., Kuster, J.M.: Formal Agent-Oriented Modeling with UML and Graph Transformation. Sci. Comput. Program. 44, 229–252 (2002)