A Systematic Approach for Mobile Agent Design Based on UML (Class and Sequence Diagrams)

M. S. Al_Kholy, A. R. Khalifa and M. G. Alsaied

Systems and Computer Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt

Abstract: Agent researchers are still trying to determine useful ways to represent agents and agent-based systems. So, this paper presents a proposal for a Systematic Approach for Agent Design by using a Unified Modelling Language (UML) diagram. Here we illustrate notions for the behavior of an agent using and extending UML class diagrams. Focus on representing the agent migration from take requests and between other hosts. In this case study, we explain one variant of notation that is the most suitable for given scenario, show that it is easier to design agent applications based on agent UML, by developing software for our case study generated by UML software package. [Journal of American Science. 2010;6(12):284-290]. (ISSN: 1545-1003).

Keywords: Mobile Agent Design, Class Diagram ,Sequence Diagram, UML, A Systematic Approach

1. Introduction:

Since a long time people have been using each other's and sometimes animals as their agents. Developments in information processing technology, computers and their networks, have made it possible to build and use artificial agents. These agents are the advanced tools that people can use to achieve different goals and to solve various problems. The main difference between ordinary tools and agents is that agents can function independently from those who delegated agency to the agents. Now, the most popular approach in artificial intelligence is based on agents. Intelligent agents form a basis for many kinds of advanced software systems that incorporate varying methodologies, diverse sources of domain knowledge, and a variety of data types. The intelligent agent approach has been applied extensively in business applications, and more recently in medical decision support systems [1, 2] as well as ecology [3]. In the general paradigm, the human decision maker is considered to be an agent and is incorporated into the decision process. The overall decision is facilitated by a task manager who assigns subtasks to the appropriate agent and combines conclusions reached by agents to form the final decision. This paper is structured as follows. Section 1 is this introduction. Section 2 gives the concept of agent (definitions). Section 3 represents related work that includes Historical overview and answer the question (why UML?). Section 4 shows the different UML diagrams and their applications for agent-based systems, basically concerning with class diagrams. Section 5 provides a case study with a searcher scenario. Section 6 represents a Class

Diagram for the Case Study. Section 7 concludes the paper.

2. The Concept of an Agent

There are several definitions of intelligent and software agents. Some of the major definitions and descriptions of agents are given as follows:

- Agents are semi-autonomous computer programs that intelligently assist the user with computer applications. This is achieved by employing artificial intelligence techniques to assist users with daily computer tasks, Such as reading electronic mail, maintaining a calendar, and filing information. Thus Agents can learn through example-based reasoning and can improve their performance over time.
- Agents are computational systems that inhabit some complex, dynamic environment, and sense, and thus act autonomously to realize a set of goals or tasks.Agents are software robots that think and act on behalf of a user to carry out tasks. An agent helps meet the growing need for more functional, flexible, personal computing and telecommunications systems. The usage of intelligent agents includes self-contained tasks, operating semi-autonomously, and communication between user and systems resources.
- Agents are software programs that implement user delegation. Agents manage complexity, support user mobility, and lower the entry level for new users. Agents are a design model similar to client-server computing, rather than being strictly a technology, program, or product [4].

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors, (Russel and Norvig, [5]).
- Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions, (Hayes-Roth, [6]).
- Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program, with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires [7].
- People, animals, and robots are examples of physical agents. Software agents and Ego in the sense of psychoanalysis are examples of mental agents. The head of a Turing machine (cf., for example, Burgin, [2]) is an example of a structural agent.

3. Related Work

3-1 Historical overview

A considerable number of agent-oriented methodologies and tools are available today, and the agent community is facing the problem of identifying a common vocabulary to support them (for details see the work in [9], on which this section is based). There is a considerable interest in the agent R&D community in methods and tools for analyzing and designing complex agent-based software systems, including various approaches to formal specification (see [10] for a survey). Since 1996, agent-based software engineering has been in the focus of the ATAL Workshop series; it also was the main topic of the 1999 MAAMAW Workshop [11]. Various researchers have developed methodologies for agent design, touching on representational mechanisms, like the GAIA methodology [12] or the extensive program underway at the Free University of Amsterdam on compositional methodologies for requirements [13], design [14], and verification [15]. In [16,17], Kinny et al. propose a modelling technique for BDI agents. The close affinity between design mechanisms employed for agent-based system and those used for object-oriented systems is shared by a number of authors, for example, [18]. In particular, since 2000, the Agent-Oriented Software Engineering Workshop (AOSE) has become the major forum for research carried out on these topics, including new methodologies such as Tropos [19], Prometheus [20], and MESSAGE [21]. Currently, most industrial methodologies are based on the Object Management Group's (OMG) Unified Modelling Language (UML) accompanied by process frameworks such as the Rational Unified Process (RUP), see [22] for details. The Model-Driven Architecture (MDA [23]) from the OMG allows a cascade if code generations from high-level models (platform independent model) via platform dependent models to directly executable code. Another approach for agile software engineering that has been receiving active coverage is Extreme Programming [24].

The UML is a standard modelling language for visualizing, specifying, constructing, and documenting the elements of systems in general, and software systems in particular [25]. UML has a well-defined syntax and semantics. It provides a rich set of graphical artefacts to help in the elicitation and top-down refinement of object-oriented software systems from requirements capture to the deployment of software components.

In UML, systems can be modelled by considering three aspects, the behavioural, the structural and the architectural aspects; each aspect is concerned with both the static and dynamic views of the system. The static view represents a projection onto the static structures of the complete system description. However, the dynamic view represents a projection onto the dynamical behaviour of the system. Finally, views are communicated using a number of diagrams including information emphasizing a particular aspect of the system.

3-2 Why UML

As an OMG standard, UML 2.0 has been considered a "final" standard, as of November 2004 [26]. In other words, many of the errors and inconsistencies of the original submission have been rectified. More than 3000 issues were files and resolved by the UML 2.0 Finalization Task Force. As such, software vendors can begin to build software tools that support the UML 2.0 Superstructure and Infrastructure. In addition, a firmer foundation is now available to adequately support the extensions for agent-based system modelling. The FIPA Modelling Technical Committee [27] and the OMG Agent Special Interest Group are actively working on extending UML for agent-based system modelling. These efforts are primarily supported by the work of more than a dozen software tool vendors.

4. Agent modelling with (UML)

UML is adequate for modelling object-oriented (OO) systems. But UML lacks the capability to readily model and specify agent systems. Unlike [Odell 2001a]'s Agent UML, we feel that every component of the UML must be extended. UML has a long history and is the result of a standardization effort on different modelling languages (like Entity-Relationship-Diagrams, the Booch-Notation, OMT, OOSE), namely Unified Modelling Language. The most popular versions of UML are UML 1.x, but now UML 2.0 is the upcoming new specification for development of systems. (UML) is a standard modelling language for visualizing (using the standardized graphic UML notations), specifying the static structure, dynamic behaviour and model organization as well as constructing system, by mapping UML to programming environment, generating some code automatically, and documenting every phase of the lifecycle from analysis and design through deployment and maintenance. UML consists of a notation, describing the syntax of the modelling language, a graphical notation, and a meta model describing the semantics of UML, namely the static semantics of UML, but no operational semantics. However, UML defines no software process, since a software process describes the development activities, the dependencies of these activities and how they are applied.

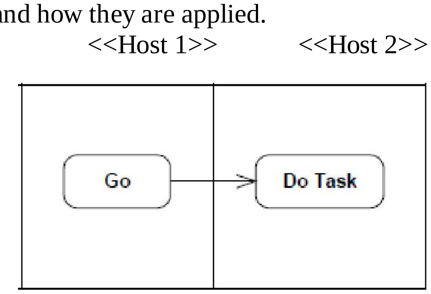

Figure1. Go action in UML

UML 2.0 supports the following diagrams: class, object, component, deployment and composite structure diagrams for modelling the static aspects of the systems and use case, state machine, sequence, activity, interaction overview, timing and communication diagrams for modelling dynamic aspects and packages, models and subsystems for modelling the model management [27]. Figure 1 shows an agent moving from location "host 1" to "host 2" and represented using "Go" activity.

Class diagram

In this section we focus on the first diagram (Class Diagram Figure 2) defined in the Superstructure Specification. We will use this distinction to present the diagram type and how it can be applied for modelling agent-based systems.

A Class Diagram describes on the one side a data model, i.e. collection of declarative (static) model elements, like classes and types, and on the other side their contents and relationships. Moreover the static structure of the system to be developed and all relevant structure dependencies and data types can be modelled with this class diagram [25]. They are applied in various phases of the project, e.g. analysis (conceptual modelling of the domain), design (platform independent description) of the implementation, detailed design (platform specific description) and to bridge the gap to the behavior diagrams. Class diagrams describe classes and interfaces with their attributes and operations, as well as associations between them (including aggregation and composition), but also generalization (a specific kind of inheritance) and dependencies among them. New to UML 2.0 is that attributes have ordering, graphical notations for associations are defined, graphical interface notation are introduced using lollipops, some unification on the notations for e.g. visibility, names and types has been done [26,28]. Moreover attributes have no implicit composition associations and dependencies are completely redefined. Class diagrams are illustrated in Figure 2. An agent model can be defined using class names, inheritance (generalization) of classes and adding name, type, position/role, capabilities and constrains, either directly or via associations. A role hierarchy can be defined using generalization. However, roles cannot be modeled in the necessary detail with any UML 2.0 diagram. Service models can also be done by this diagram type, e.g. defining services with input/output parameters and pre-/post-conditions as classes with attributes and functions (the service interface).

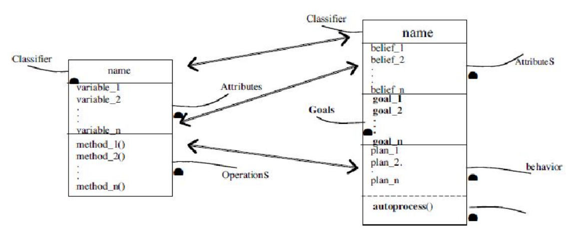

Figure2. Specifying agent behaviour using UML class diagram

5. Case Study: Book Searcher

The case study includes three network nodes: Home, Host 1 (British Library) and Host 2 (Congress Library) Figure 3. On Host 1 and Host 2 resides library agent, which is responsible for providing the books List. The searcher agent is created on the Home node. The input parameter is the item. The Searcher agent migrates from home node to Host1 node and requests library1 agent to give the books list. The library1 agent responds with the whole books list. The searcher extracts the book and migrates to the next node. After visiting all nodes the

Searcher agent migrates back to the Home node and informs the user where it has found the specified item.

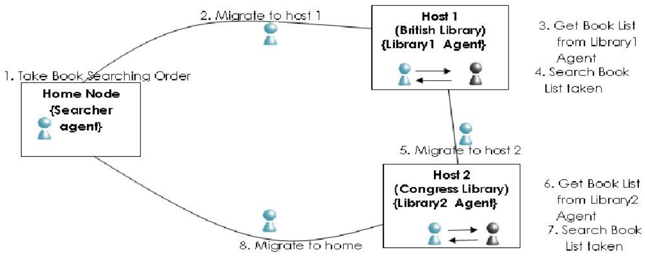

Figure3. Book searcher scenario

The mobile agent one-to-one relationship is the simplest; where the mobile agent (library agent) is placed between two negotiators (user searcher agent and the library) in this case. Similarly, one-to-many and many-to-one relationships; where the mobile agent (library agent) is placed between one negotiator at one side and more than one negotiator at the other side (a user searcher agent and more than one library) in this case.

The user inputs his demand through the Graphical User Interface (GUI) where it is going to be placed as a search_query. The user searcher agent then scans the network in order to build a list of available libraries.

The user searcher agent then takes the search_query and starts the journey by visiting the first library in the list.

Before the user searcher agent can reach the server of the library, it must pass the library's security check. While the user searcher agent enquires about the book needed, a local library agent, residing in the library server, is activated. There will be two scenarios with respect to the library: book found and book not found. The local library agent returns the results to the user searcher agent if the book is found then terminates the communication with the user searcher agent. If the book is not found, then the local library agent informs the user searcher agent that the book wasn't found and then terminates the communication with the user searcher agent. The user searcher agent then follows the itinerary and moves to the next library. Finally, the user searcher agent returns back to the user with the librarie's list where it found the book needed.

6. Sequence Diagram for the Case Study:

Before the user searcher agent can reach the server of the library, it must pass the library's security check. While the user searcher agent enquires about the book needed, a local library agent, residing in the library server, is activated. There will be two scenarios with respect to the library: book found and book not found. The local library agent returns the results to the user searcher agent if the book is found then terminates the communication with the user searcher agent. If the book is not found, then the local library agent informs the user searcher agent that the book wasn't found and then terminates the communication with the user searcher agent. The user searcher agent then follows the itinerary and moves to the next library. Finally, the user searcher agent returns back to the user with the libraries list where it found the book needed figure 4 illustrate Sequence diagram for case study.

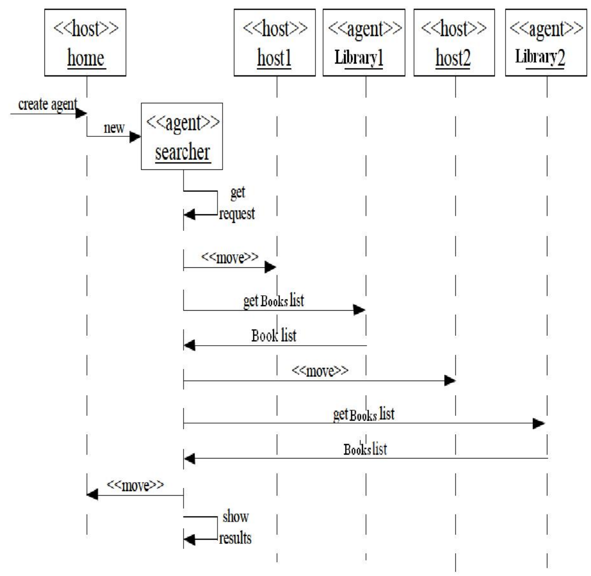

Figure. 4 Agent-Sequence diagrams applied to example

7. Class Diagram for the Case Study

In this section we show how usual UML class diagrams can be used and extended in the framework of agent oriented programming development. We will use the following notation to distinguish between different kinds of agent classes and instances. The first one denotes some agent class, the second some agent class satisfying distinguished roles and the last one defines some agent instance satisfying distinguished roles. The roles can be neglected for agent instances. According to the statement given above what has to be specified for agent classes we specify agents by the agent class diagram.

The usual UML notation can also be used to define such an agent class, but for more understandable reasons we have introduced the above notation. Using stereotypes, an agent class written as a class diagram can look as shown in Figure.2.

The Class and the Activity diagrams are generated as the static and dynamic aspects of objects by represented the attributes and operations of the object. Figure 5 shows the Class diagram and Activity diagram applied to our example. The Activity diagram shows how to search the information and find the best solution. In the Class diagram, there are four classes for our problem. Each class has attributes and operations, showing their roles as follows:

User_Interface class:

- read_search_query: This method is for reading the search criteria from the user through the GUI of the searcher agent.
- display_results: This method is for displaying the results found.
- trace: This method is for displaying any messages.

Agent Class:

- start_agent: This method is for starting the user searcher agent.
- stop_agent: This method is for stopping the user searcher agent after accomplishing the task.
- terminate: This method is for ending the code.

Agent_Control Class:

- scan_network: This method is for scanning the network to find the libraries servers.
- return_results: This method is for sending the results to the user.
- stop_agent_control: This method is for ending the Agent Control.

Library_Agent Class:

- start_agent: This method is for starting the library agent.
- stop_agent: This method is for stopping the library agent after accomplishing the task.
- find_item: This method is for searching the library server’s database for the book needed.
- return_results: This method is for sending the results to the user searcher agent.
- terminate_communication: This method is for ending the communication between the library agent and the user searcher agent.
- inform_termination: This method is for informing the user searcher agent that communication is terminated with the library agent.

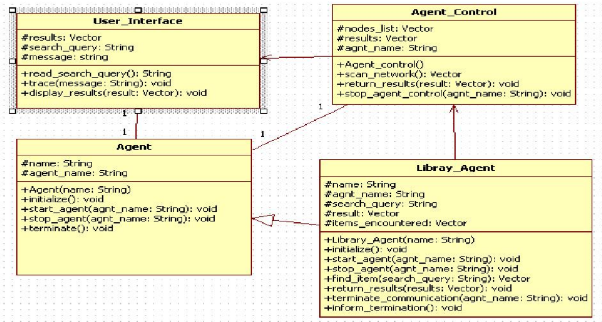

Figure. 5 Agent-class diagrams applied to example

8. Evaluation and Conclusion

This paper presents a Systematic Approach for Agent Design to support the modeling and the implementation of an agent using UML profile which defines a class diagram. From the end user's perspective, the goal is to provide a personal travel assistant, i.e., a software agent that uses information about the users' schedule and provides preferences in order to assist users in travel, including preparation as well as on-trip support. This requires providing ubiquitous access to assistant functions for the user, in the office, at home, and while on trips, using PCs, notebooks, information terminals, PDAs, and mobile phones.

The requirements for artifacts to support the analysis and design became clear, and the material described in this paper has been developed incrementally, driven by these requirements. So far, no empirical tests have been carried out to evaluate the benefits of the Agent UML framework. However, from this paper, we see two advantages as a result: First, they make it easier for users who are familiar with object-oriented software development but new to developing agent systems to understand what multi agent systems are about, and to understand the principles of looking at a system as a society of agents rather than a distributed collection of objects. Second, our estimate is that the time spent for design can be reduced by a minor amount, which grows with the number of agent-based projects. However, we expect that as soon as components are provided to support the implementation based on Agent UML specifications, this will widely enhance the benefit. In our work we use the star UML package to develop software for our case study by generating a code from star UML software package. This software can generate a code by more than one languages such as Java, C++, and others.

As for future work, we are looking forward to implement MA-UML diagrams. Also we plan to the design and implement of a mobile agent security based on A Systematic Approach for modelling Agent Mobility with other UML Diagrams.

Corresponding Authors
M. S. Al_Kholy
Systems and Computer Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt

Acknowledgement
The authors would like to thank Prof. Dr. Mohamed Zaki, for his leadership, continued help, support, and encouragement. We especially are grateful to him for sharing his knowledge, experience, and expertise.

9. References

1. Hsu C., Goldberg H.S.: Knowledge-mediated retrieval of laboratory observations, Proc. JAMIA, v.23:809--813(1999)
2. Lanzola G., Gatti L., Falasconi S., Stefanelli M.: A framework for building cooperative software agents in medical applications, Artificial Intelligence in Medicine, 16, 223--249. (1999)
3. Judson, O.P.: The Rise of the Individual-based model in Ecology, Trends in Ecology and Evolution, 9, 9-14 (1994).
4. Jansen, J.: Using Intelligent Agents to Enhance Search Engine Performance, Firstmonday, No.2/3, http://www.firstmonday.dk (1996)
5. Russel, S.J. and Norvig, P.: Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, N.J. (1995).
6. Hayes-Roth, B.: An Architecture for Adaptive Intelligent Systems. Artificial Intelligence: Special Issue on Agents and Interactivity, 72, 329-365 (1995).
7. IBM's Intelligent Agent Strategy white paper, http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm
8. Burgin, M.: Super-recursive Algorithms, Springer, New York/Berlin/Heidelberg (2005).
9. Bauer, B. and Müller, J.P.: Methodologies and Modeling Languages, in: Agent-Based Software Development, Luck M., Ashri R. D'Inverno M. (eds.) Artech House Publishers, Boston, London, 2004.
10. C. A. Iglesias, M. Garijo, and J. C. González. A Survey of Agent-Oriented Methodologies. Proceedings of Fifth International Workshop on Agent Theories, Architectures, and Languages, pages 185-198, University Pierre et Marie Curie, 1998.
11. F. J. Garijo, and M. Boman. Multi-Agent System Engineering. Proceedings of MAAMAW'99. Springer, ed., 1999.
12. M. Wooldridge, N. R. Jennings and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. International Journal of Autonomous Agents and Multi-Agent Systems, 3, 2000.
13. D. E. Herlea, C. M. Jonker, J. Treur, and N. J. E. Wijngaards. Specification of Behavioural Requirements within Compositional Multi-Agent System Design. Proceedings of Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, pages 8-27, Springer, 1999.
14. F. M. T. Brazier, C. M. Jonkers, and J. Treur. Principles of Compositional Multi-Agent System Development. Proceedings 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98, pages 347-360, Chapman and Hall, 1998.
15. C. M. Jonker, and J. Treur. Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. Proceedings of International Workshop on Compositionality (COMPOS'97), Springer, 1997.
16. D. Kinny, M. Georgeff, and A. Rao. A Methodology and Modelling Technique for Systems of BDI Agents. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)., pages 56-71. Springer, 1996.
17. D. Kinny, and M. Georgeff. Modelling and Design of Multi-Agent Systems. Intelligent Agents III, Springer, 1996.
18. Birgit Burmeister: Models and methodology for agent-oriented analysis and design. In Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, 1996. DFKI Document D-96-06.
19. A. Th. Schreiber, B. J. Wielinga, J. M. Akkermans, and W. Van de Velde. CommonKADS: A comprehensive methodology for KBS development. Deliverable DM1.2a KADSII/M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994.
20. Prometheus home page: http://www.cs.rmit.edu.au/agents/SAC/methodology.shtml.
21. MESSAGE web site: http://www.eurescom.de/public/projects/P900-series/p907/

22. Ivar Jacobson, Grady Booch, James Rumbaugh: The Unified Software Development Process, Addison Wesley, 1998.
23. Model-driven Architecture: http://www.omg.org/mda/.
24. Beck, K. Extreme Programming Explained. Addison Wesley, 1999.
25. G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, 1998.
26. OMG, Unified Modeling Language: version 2.0 (UML 2.0), Final Adopted Specification, 2003, http://www.uml.org/#UML2.0
27. FIPA Modeling TC, FIPA Modeling Area: Deployment and Mobility,2003/05/13, http://www.auml.org/auml/documents/DeploymentMobility.zip
28. Miao Kang, Lan Wang, and Kenji Taguchi, Modelling Mobile Agent Applications in UML 2.0 Activity Diagrams, 2004/04/21, http://www.auml.org/auml/supplements/UML2-AD.pdf

6/21/2010