A Systematic Approach for Agent Design Based on UML

Momtaz Al-Kholy, Ahmed Khalifa and Mohamed El-saied
Systems and Computer Engineering Department, Al-Azhar University, Egypt

Abstract *Agent researchers are still trying to determine useful ways of representing agents and agent-based systems. So this paper presents a proposal for a Systematic Approach for Agent Design by using Unified Modeling Language (UML) diagram. In this paper 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 a case study we explain one variant of notation that is the most suitable for given scenario. And show that it is easier to design agent applications based on agent UML, by develop software for our case study generated by UML software package.*

Keywords: *Agent Design, UML, A Systematic Approach*

Received May 18, 2010; Accepted September 15, 2010

1. Introduction

For a long time people used each others 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. Agents are the advanced tools people use to achieve different goals and to 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 [27], [6] and ecology [17]. 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 that assigns subtasks to the appropriate agent and combines conclusions reached by agents to form the final decision. This paper structured as follows. In section 2 we give the concept of agent (definitions). Section 3 represents related work that includes Historical overview and answer for question (why UML?). Section 4 showing the different UML diagrams and their application for agent-based systems, we concern with class diagram. Section 5 gives a case study with searcher scenario. Section 6 represents Class Diagram for the Case Study. Section 7 concludes the paper.

2. Background 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 computational systems that inhabit some complex, dynamic environment, and sense and it acts autonomously to realize a set of goals or tasks.

Agents are semi-autonomous computer programs that intelligently assist the user with computer applications by employing artificial intelligence techniques to assist users with daily computer tasks. Such as reading electronic mail, maintaining a calendar, and filing information. Agents learn through example-based reasoning and are able to improve their performance over time.

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 strictly a technology, program, or product [9].

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors, Russel and Norvig, [26].

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, [11].

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 [12].

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, [6]) is an example of a structural agent.

3. Related Work

3.1. 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 work in [1], this section is based on it). 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 [13] 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 [8]. Various researchers have developed methodologies for agent design, touching on representational mechanisms, like the GAIA methodology [8] or the extensive program underway at the Free University of Amsterdam on compositional methodologies for requirements [9], design [5], and verification [16]. In [9, 19], Kinny et al. propose a modeling 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, [3]. 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 [10], Prometheus [25], and MESSAGE [21]. Currently, most industrial methodologies are based on the Object Management Group's (OMG) Unified Modeling Language (UML) accompanied by process frameworks such as the Rational Unified Process (RUP), see [21] 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 [2].

The UML is a standard modeling language for visualizing, specifying, constructing, and documenting the elements of systems in general, and software systems in particular [4]. UML has a well-defined syntax and semantics. It provides a rich set of graphical artifacts 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 modeled by considering three aspects, the behavioral, 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 behavior of the system. Finally, views are communicated using a number of diagrams containing information emphasizing a particular aspect of the system.

3.2. Why UML?

As an OMG standard, UML 2.0 is now considered a "final" standard, as of November 2004 [24]. 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 modeling. The FIPA Modeling Technical Committee [8] and the OMG Agent Special Interest Group are actively working on extending UML for agent-based system modeling. These efforts are primarily supported by the work of more than a dozen software tool vendors.

4. Agent modeling with Unified Modeling Language (UML)

UML is adequate for modeling 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 modeling languages (like Entity-Relationship-Diagrams, the Booch-Notation, OMT, OOSE), namely Unified Modeling 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. The Unified Modeling Language (UML) is a standard modeling language for visualizing (using the standardized graphic UML notations), specifying the static structure, dynamic behavior and model organization as well as constructing system, by mapping UML to programming environment and generate 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 modeling language and 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, dependencies of these activities and how they are applied.

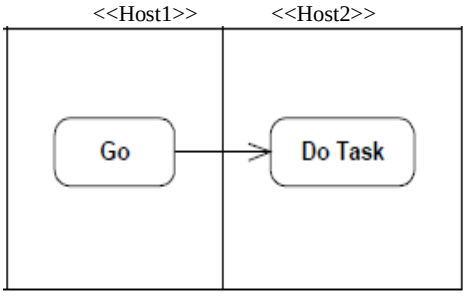

Figure 1. Go action in UML.

4.1. 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 can be applied for modeling 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 modeled with class diagram [14]. They are applied in various phases of the project, e.g. analysis (conceptual modeling 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[24, 22]. 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).

5. Case study: Book Searcher

The case study includes three network nodes: Home, Host1 (British Library) and Host2 (Congress Library) figure 3. On Host1 and Host2 resides library agent, which is responsible for providing the books List. The searcher agent is created on Home node. The input parameter is the item. The Searcher agent migrates from home node to Host1 node and requests library1agent 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

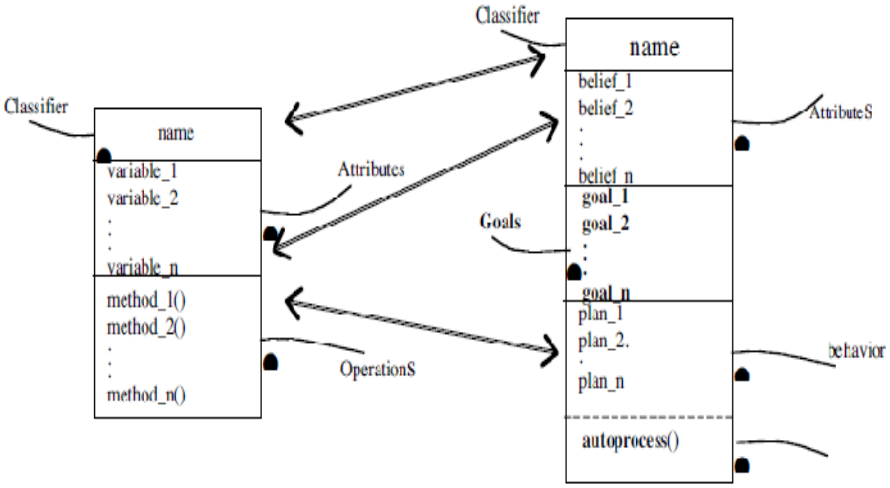

Figure 2. Specify agent behavior using UML class diagram

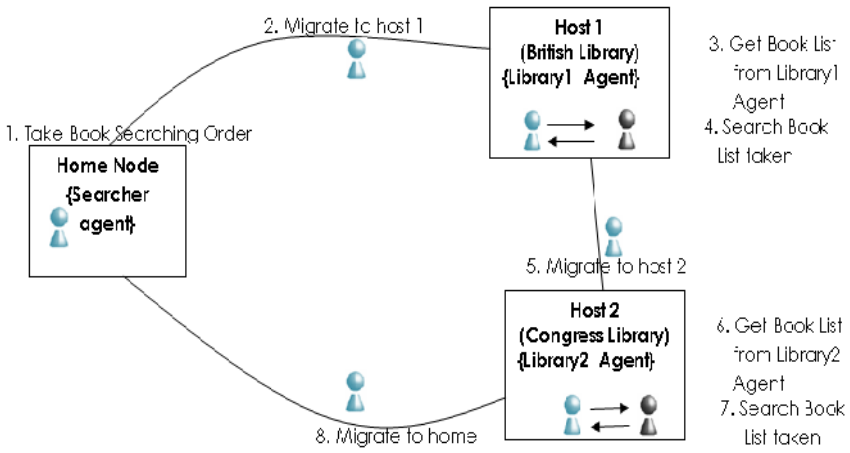

Figure 3. Book searcher scenario

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

The mobile agent one-to-one relationship is the simplest; where the mobile agent (library agent) isplaced 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 on 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 libraries list where it found the book needed.

6. Class Diagram for the Case Study

In this section we show how usual UML class diagrams can use 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 readable 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 Activity diagrams are generated as the static and dynamic aspects of objects by represented the attributes and operations of the object. Figure 4 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.

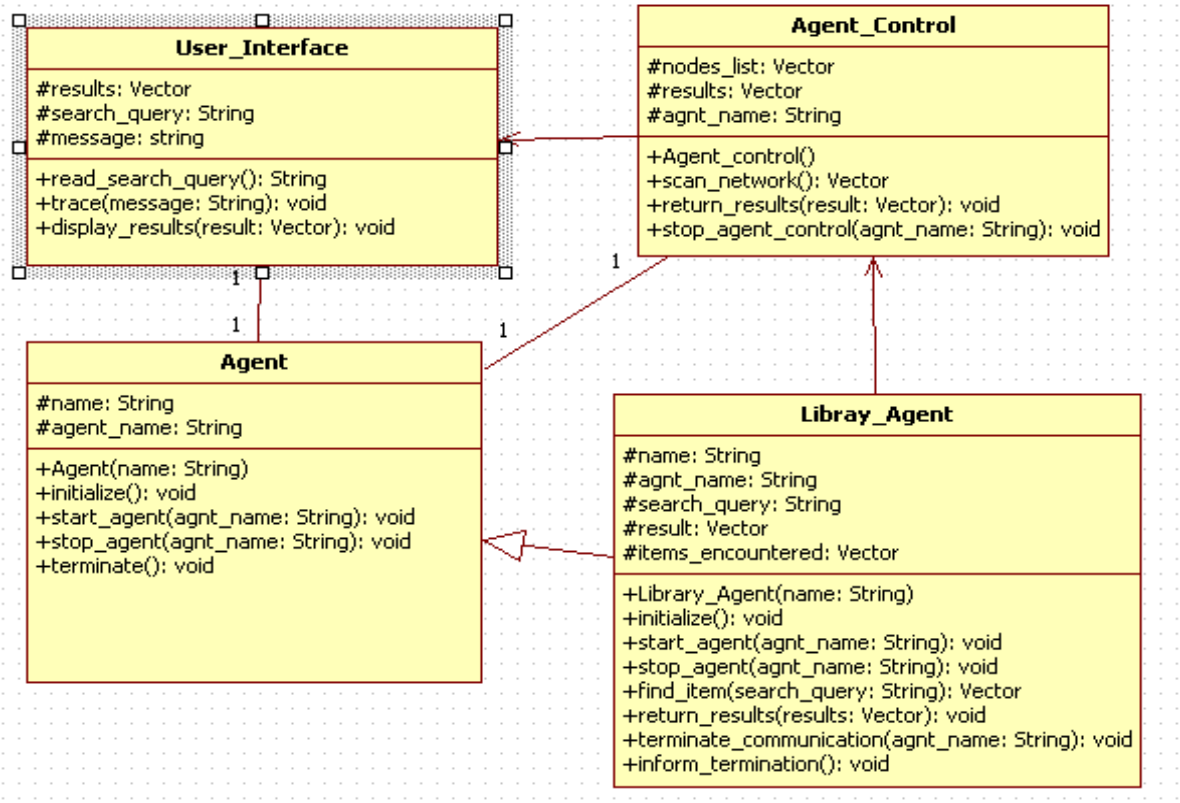

Figure 4. Agent-class diagrams applied to example.

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

7. Evaluation and Conclusion

This paper presents a Systematic Approach for Agent Design to support the modeling and the implementation of 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 preferences in order to assist them 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 the trip, 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 extensions as a result: Firstly, 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. Secondly, 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 star UML package to develop software for our case study by generate a code from

star UML software package, this software can generate a code by more than languages such as Java, C++, and other.

As a future work we are looking to implement MA-UML diagrams. Also looking to the design and the implementation of a mobile agent based A Systematic Approach for modeling Agent Mobility with other UML Diagrams.

References

[1] 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

[2] Beck, K. *Extreme Programming Explained*. Addison Wesley, 1999.

[3] Birgit B: "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.

[4] Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, 1998.

[5] Brazier F. M., Jonkers M., and Treur J. "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.

[6] Burgin, M.: *Super-recursive Algorithms*, Springer, New York/Berlin/Heidelberg (2005).

[7] FIPA Modeling TC, "FIPA Modeling Area": *Deployment and Mobility*,2003/05/13, http://www.auml.org/auml/documents/DeploymentMobility.zip

[8] Garijo F. J., and Boman M.. "Multi-Agent System Engineering". *Proceedings of MAAMAW'99*. Springer, ed., 1999.

[9] Herlea D. E., et al. "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.

[10] Hsu C., Goldberg H.S.: "Knowledge-mediated retrieval of laboratory observations", *Proc. JAMIA*, v.23:809--813(1999)

[11] Hyes-Roth, B.: "An Architecture for Adaptive Intelligent Systems". *Artificial Intelligence: Special Issue on Agents and Interactivity*, 72, 329-365 (1995).

[12] IBM's *Intelligent Agent Strategy white paper*, http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm.

[13] Iglesias C. A., Garijo M., and González J. C.. "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

[14] Ivar Jacobson, Grady Booch, James Rumbaugh: *The Unified Software Development Process*, Addison Wesley, 1998.

[15] Jansen, J.: *Using Intelligent Agents to Enhance Search Engine Performance*, *Firstmonday*, No.2/3, http://www.firstmonday.dk (1996)

[16] Jonker C. M., and Treur J.. "Compositional Verification of Multi-Agent Systems": a Formal Analysis of *Pro-activeness and Reactiveness. Proceedings of International Workshop on Compositionality* (COMPOS'97), Springer, 1997.

[17] Judson, O.P.: "The Rise of the Individual-based model in Ecology", *Trends in Ecology and Evolution*, 9, 9-14 (1994).

[18] Kinny D, and Georgeff M.. "Modelling and Design of Multi-Agent Systems". *Intelligent Agents II*I, Springer, 1996.

[19] Kinny D, Georgeff M, 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.

[20] Lanzola G., et al ."A framework for building cooperative software agents" *in medical applications, Artificial Intelligence in Medicine*, 16, 223--249. (1999)

[21] *MESSAGE web site*: http://www.eurescom.de/public/projects/P900-series/p907/

[22] 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

[23] *Model-driven Architecture*: http://www.omg.org/mda/

[24] OMG, *Unified Modeling Language: version 2.0 (UML 2.0), Final Adopted Specification*, 2003, http://www.uml.org/#UML2.0

[25] *Prometheus home page*: http://www.cs.rmit.edu.au/agents/SAC/methodology.shtml.

[26] Russel, S.J. and Norvig, P.: *Artificial Intelligence: A Modern Approach*, *Prentice-Hall, Englewood Cliffs*, N.J. (1995).

[27] Schreiber A. , et al "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 Brussel*s, 1994.

[28] Wooldridge M., Jennings N. R. and Kinny D.. "The Gaia Methodology for Agent-Oriented Analysis and Design". *International Journal of Autonomous Agents and Multi-Agent Systems*, 3, 2000.

Momtaz Alkholy Was born in Bascara, Algaria, on May, 1975 he received B. Sc. Degree in Systems and Computer Engineering in 1998 from Faculty of Engineering Al-Azhar University Nasr City, Egypt, He is M. Sc student in computer engineering at the same university. His area of Research is Mobile Agent, Security, Networks.

Ahmed Khalifa born in, Egypt, on April 18, 1958. In 1981, he received his Bachelor Degree in Systems and Computer Engineering from the Faculty of Engineering Al-Azhar University, Cairo, Egypt. In 1987, he received his Master Degree in Computer Engineering from Manhattan College, New York. In 1989, he received a second Master Degree in Computer Science from the City university of New York (CUNY) USA, his PhD Degree in Computer Science from the City University of New York (CUNY) in 1993. He is currently an Associate Professor in Systems and Computer Engineering Dept. Al-Azhar University Nasr City, Egypt. His research interests include Information Security, Wireless Networks, Network Security, and Web Services Technologies and Security.

Mohamed El-Saied Was born in, Egypt, on 25/07/1960. He received his Ph.D. jointly conducted between University of Colorado at Boulder, USA, & AL-Azhar University, Cairo, Egypt 1995. Thesis title was "Visualization of Software Design Environments". From 1991 to 1994, he works as a Research Assistant as well as a member of Human Computer Interaction group in the computer science department, university of Colorado at Boulder, USA. From 1994 until now, as a Faculty member in the Systems and Computer Department, Faculty of Engineering, Al-Azhar University, Egypt, Cairo. His research interests include Web technologies, E-learning, and software Engineering.