Author(s): Mouratidis, Haralambos; Odell, James; Manson, Gordon.
Article title: Extending the Unified Modeling Language to Model Mobile Agents
Year of publication: 2002
Citation: Mouratidis, H. et al (2002) 'Extending the Unified Modeling Language to Model Mobile Agents' Proceedings Agent Oriented Methodologies Workshop, Annual ACM Conference on Object Oriented Programming, Systems, Languages (OOPSLA), Seattle – USA.
Link to conference website: http://www.oopsla.org/2002/fp/index.html

Extending the Unified Modeling Language to Model Mobile Agents

Haralambos Mouratidis[1], James Odell[2], Gordon Manson[1]
[1]*Department of Computer Science, University of Sheffield, England*
{h.mouratidis, g.manson}@dcs.shef.ac.uk
[2]*James Odell Associates, Ann Arbor, MI USA*
email@jamesodell.com

Abstract

Mobile Agents represent a crucial part in most agent-based systems. However, very little work has taken place in modelling such systems and, up to now, none of the existing agent oriented methodologies provide concepts and notations to fully capture mobile agents. In this work we are presenting extensions to the Unified Modeling Language (UML) to model mobile agents. We use three different scenarios to illustrate the proposed extensions.

1. Introduction

Analysing and designing complex computerised systems has proved to be a difficult task. Actually, it has been argued [1] that developing software for domains like telecommunications represents one of the most complex tasks humans undertake.

Agent Oriented Software Engineering (AOSE) introduces an alternative approach in analysing and designing complex computerised systems [1,2]. According to AOSE, a complex computerised system is viewed as a multi-agent system in which autonomous software agents (subsystem) interact with each other in order to satisfy their design objectives. AOSE provides designers with more flexibility in their analysis and design. The actual design of the system takes place by specifying a multi-agent system as a society, similar to a human society, consisting of entities that possess characteristics similar to humans such as mobility, and intelligence with the capability of communicating.

However, the concept of a software agent is not uniquely defined. Researchers have given definitions of the concept according to some typical characteristics [3], some operational characteristics [4] or some cognitive functions that agents should implement [5].

One of the most promising features of software agents is mobility. Mobile agents are software entities that can migrate autonomously throughout a network from host to host. This means they are not bounded to the platform they begin execution. Mobile Agents are emerging as an alternative programming-concept for the development of distributed applications [3,6,7]. So far, most of the work on the area of mobile agents has been focusing on the technology itself, and the development of agent frameworks to support mobility.

Many Agent Oriented Software Engineering methodologies have been developed during the last few years [8,9,10]. However, very little work has taken place in defining concepts and notations to capture and model mobile agents. Mobile agents are a crucial part in most agent-based systems and the lack of models to capture them restricts the usefulness of the existing methodologies. Only recently (2001) work was initiated trying to capture mobile agents during the analysis and design stages of the development. Very preliminarily work has been initiated by introducing some concepts to capture mobility at the MaSE methodology [11]. In addition, Klein et al have proposed some extensions to UML for mobile agents [12]. Although, these two works represent an initial approach on modelling mobile agents, more work must take place towards this direction. As proposed by Mouratidis et al [13] an approach to capture the concept of mobile agents, is to introduce concepts and notations (or use existing ones) to give answers to questions that arise from the use of mobile agents such as *why* a mobile agent moves from one platform to another, *where* the agent moves to, *when* the agent moves, and *how* it reaches the targeted platform.

In this paper we indicate how UML can be used to express agent mobility and we present extensions where might be necessary. Section 2 gives an overview on Mobile Agents, while Section 3 introduces the Unified Modeling Language (UML). In Section 4 we are describing the usage of UML to express mobile agents, and we propose extensions to model characteristics of mobile agents that are not adequately modelled using standard UML. In Section 5 we illustrate the extensions with the aid of 3 different scenarios, while in Section 6 we present related work and the differences of our approach. Finally Section 7 presents directions for future work and some concluding remarks.

2. Mobile Agents

Software Agents can be static or mobile. A static agent executes only in the system where it starts execution. If it needs information from another system or needs to interact with agents in another system, it

uses a communication mechanism such as messaging or remote procedure calling (RPC). A mobile agent, on the other hand, is not bound to the system where it starts execution [14] but it is able to transport itself from one machine to another amongst the hosts of the network. Thus, a mobile agent can move to a system that has the information that the agent wants to obtain. A mobile agent when it transports itself it transports its state and code [15]. The code of a mobile agent is the class code that the agent needs in order to execute, while the state of a mobile agent is the values that determine what the agent will do after it transport itself.

Mobile Agents have been promising in many application domains such as mobile computing, health care, military applications and information retrieval. It has been widely argued [6] that Mobile Agent technology provides many advantages, over traditional techniques, such as reduction of network load, better support of synchronous and asynchronous interactions and reduction of concurrency. For example, a mobile agent can move to a particular location on the Internet to obtain information on behalf of its user. Thus, it accesses the resources locally than sending multiple requests across the network. However, according to [15] the real advantage of mobile agents is the fact that there is no other alternative to all of the functionality supported by a mobile agent framework.

The last few years many researchers are focusing on mobile agents. Research topics on mobile agents include, amongst other, security [16] and communication [6,7]. Another major area of research on mobile agents involves the development of mobile agent platforms [17]. Many different mobile agent platforms have been developed, most of them based on Java. The choice of all those platforms has focused the development of a mobile agent-based system to the implementation stage. The usual approach towards the development of a mobile agent system is to define a mobile agent platform and start implementing the system. Very little or no design of the system takes place. The need to analyse and design a system before its implementation has been identified by many researchers [18,19].

3. Unified Modeling Language (UML)

During the seventies, structured programming was the dominant approach to software development. Along with it, software engineering technologies were developed in order to ease and formalize the system development lifecycle: from planning, through analysis and design, and finally to system construction, transition, and maintenance. In the eighties, object-oriented (OO) languages experienced a rise in popularity, bringing with it new concepts such as data encapsulation, inheritance, messaging, and polymorphism. By the end of the eighties and beginning of the nineties, a jungle of modelling approaches grew to support the OO marketplace. To make sense of and unify these various approaches, an Analysis and Design Task Force was established on 29 June 1995 within the OMG. By November 1997, a de jure standard was adopted by the OMG members called the Unified Modeling Language (UML).

The UML unifies and formalizes the methods of many approaches to the object-oriented software lifecycle, including Booch, Rumbaugh (OMT), Jacobson, and Odell [20]. It supports the following kinds of models:

- **static models**- such as class and package diagrams describe the static semantics of data and messages. Within system development, class diagrams are used in two different ways, for two different purposes. First, they can model a problem domain conceptually. Since they are conceptual in nature, they can be presented to the customers. Second, class diagrams can model the implementation of classes—guiding the developers. At a general level, the term class refers to the encapsulated unit. The conceptual level models types and their associations; the implementation level models implementation classes. While both can be more generally thought of as classes, their usage as concepts and implementation notions is important both in purpose and semantics. Package diagrams group classes in conceptual packages for presentation and consideration. (Physical aggregations of classes are called components, which are in the implementation model family, mentioned below.)
- **dynamic models**- including interaction diagrams (i.e., sequence and collaboration diagrams), state charts, and activity diagrams.
- **use cases**- the specification of actions that a system or class can perform by interacting with outside actors.
- **implementation models**- such as component models and deployment diagrams describing the component distribution on different platforms.
- **object constraint language (OCL)**- is a simple formal language to express more semantics within an UML specification. It can be used to define constraints on the model, invariant, pre- and post-conditions of operations and navigation paths within an object net.

4. Extending UML to Capture Mobile Agents

Compared to the traditional approach to objects, agents are autonomous and interactive. Based on internal states, their activities include goals and conditions that guide the execution of defined tasks. While objects need outside control to execute their methods, agents know the conditions and intended effects of their actions and hence take responsibility for their needs. Furthermore, agents act both alone and with other agents. Multiagent systems can often resemble a social community of interdependent members that act individually.

However, no formalism yet exists to sufficiently specify agent-based system development. To employ agent-based programming, a specification technique must support the whole software engineering process—from planning, through analysis and design, and finally to system construction, transition, and maintenance.

A proposal for a full life-cycle specification of agent-based system development is beyond the scope of this paper. Both FIPA [21] and the Agent UML (AUML) Working Group [www.auml.org] are exploring uses of and recommending extensions to UML. In this paper, we indicate how UML can be used to express agent mobility, as well as express where extensions to the standard UML (AUML) might be appropriate. It must be noticed that the proposed extensions are based on proposed UML 2 meta-structures.

4.1. AUML Deployment Diagram

As it was mentioned, mobile agents have the ability to migrate to different platforms within the network. We define the platform they begin execution as their *origin*, and the platform they stop execution, after finish their task, as their *destination*. We define the intermediate path between the origin and the destination as *mobility path*. To capture *mobility paths*, *origin* and *destination* we are extending UML deployment diagram and we call this extended diagram, an *AUML deployment diagram.*

In an *AUML deployment diagram* we capture the mobile agents of the system in a static nature (that means these diagrams do not capture dynamics (such as sequence) of the movement). Such a diagram provides notations to capture the mobile agents of the system, along with their *origin*, the *destination*, the platforms they might visit, and the mobile agent's *mobility paths.* An *AUML deployment diagram* provides answers to the *why* and the *where* of the before-mentioned questions.

Definition: An *AUML deployment diagram* captures mobility if for any link that is tagged «moves» the following holds:

- There is a corresponding Mobile Agent (MA)
- The *origin*, *destination* and *mobility path* of the mobile agent are specified.

The tag «moves» specifies the movement of the mobile agent from one platform to another, and it accepts a parameter that indicates the purpose (the *why*) of the movement. To indicate the *origin* of the mobile agent we introduce the stereotype «home». The stereotype «destination» is introduced to specify the *destination* of the mobile agent. It must be noticed that *destination* of a mobile agent can be its *origin*. For example when a mobile agent is sent somewhere, obtain some information and returns back to its *origin* to report the results. Finally we introduce the stereotype «visitor>> to indicate that the mobile agent visits a platform.

As in the deployment diagrams, nodes can represent types as well as instances of platforms. The mobile agents in each platform are represented as rectangles that include the name of the agent, and a stereotype that indicates the status of the mobile agent (*home, visitor, destination*). The paths are represented as dashed lines, with the arrows pointing towards the platform that the mobile agent moves to. A double arrow (both sides of the path) indicates the mobile agent moves both directions. When a «destination» tag is not used, it is assumed by default that the MA returns to its home (*origin*). In addition, in some cases, the mobile agent needs to send some *messages* to another platform. This is represented with the aid of dotted lines. A message accepts two arguments, the first argument being the platform in which the message has to be delivered and the second being the identification of the MA that sends the message.

For example, let us assume that a professional dispatches a mobile agent to book a ticket for her as shown in Figure 1(a). The *Professional Agent* might move from the *Professionals Laptop* to the *Reservation System,* but also it might move back from the *Reservation System* to the *Professionals Laptop* (to get maybe more information about the Professional, which it was not obtained before) so a double arrow represents this path. Another path could be for the *Professional Agent* to move to and from the *Ticket System.* However in this example we assume (for the purpose of illustration) that the MA only moves from the *Ticket System* to the Professional's laptop and not vice versa, so this path is indicated with an arrow pointing towards the professional's laptop.

4.2. UML Activity Diagrams

The extended deployment diagram helps to capture the *why* an agent moves to a different platform, and the *where* it moves. However it is not very helpful in capturing the *when* a mobile agent leaves a platform to move to another. This can be captured with the aid of

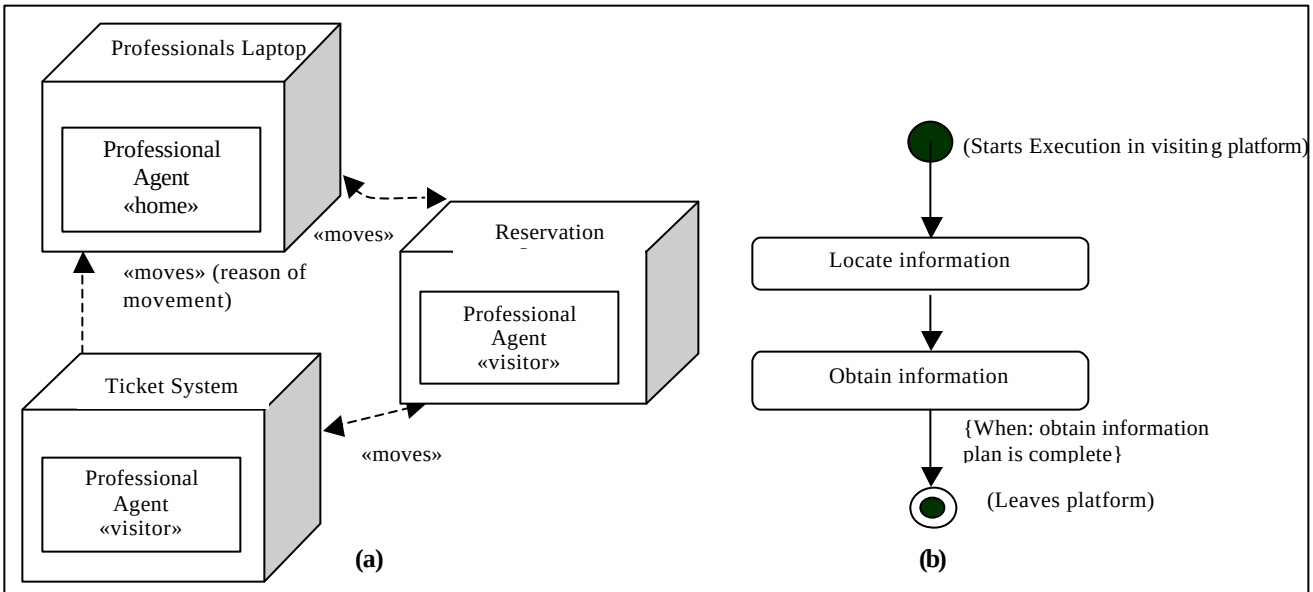

Fig. 1: (a) AUML Deployment Diagram, (b) Activity Diagram

activity diagrams. The activity nodes model plans, while the transitions model events. The starting point is the moment the agent migrates to the platform and the end point is the moment the mobile agent leaves the platform. The *when* the mobile agent moves from the platform is indicated as a parameter *(When: reason)* on the transition that leads to the end point (figure 1 (b)).

4.3. AUML Activity Diagram

The previous mentioned diagrams do not capture the dynamics of the mobility and thus fails to provide answers to the *how an agent reaches the targeted platform*. To give answers to this question we extend Activity Diagrams by using deployment nodes and we call the extended diagram *AUML Activity diagram.* This diagram helps to capture the dynamics of the mobile agents such as sequence, concurrency, and iteration. For example, from the *AUML deployment diagram* we can see that the mobile agent *Professional Agent* might move from the *Professionals Laptop* to the *Reservation System* and vice versa, but the *AUML deployment diagram* does not capture the sequence of the movement, the path that the MA follows, and the decisions that the MA takes in order to follow a specific path.

An extended Activity Diagram provides concepts to capture the sequence of the movement, the detailed mobility path (including the intermediate nodes between the home and the destination), and the decisions that drive the choice of particular intermediate nodes. The starting point of such a diagram is the despatch of the Mobile Agent from its home platform, and the destination point is either the return to its home platform or a platform in which the mobile agent finishes a particular task. In the case in which the agent returns back to its home, the starting and the destination points of the diagram are the same. The path of the mobile agent from the starting point to the destination is decomposed in different nodes that this agent might visit while trying to reach the requested platform (the platform that has the requested information). UML diamond notation is used to capture cases where the agent has to decide from a different number of nodes to visit (basically this is the case in which the designer is not sure about the path that the MA will follow in order to reach the requested platform). A simple diamond means there are only two possible nodes that the agent can move to, while a diamond with … (3 dots) means there are n possible nodes that the MA might choose from. Every time a diamond is indicated the designer must provide the agent with knowledge to decide which node it will move to. This knowledge is defined in terms of statements, such as *use the node with less traffic*. During the implementation stage these statements will be converted to code and they will be added to the knowledge database of the mobile agent (its beliefs).

Figure 2 illustrates the concept of such a diagram. A MA is dispatched from the professional's laptop. It moves from «home» to node 1. Then it has the option to move either to node 2 or to node 3. Let us assume it moves to node 2. There it has to decide from a different number of nodes (the number is not known to the designer). As it was mentioned above, this decision will be based on the knowledge of the agent, which has been indicated by the designer in terms of statements. After choosing a node, it finally reaches its destination. It must be noticed that throughout the path, the agent might find it useful to return to a previous node, in other words the mobile agent is moving to a node that has previously visited. We define this as the *return path* of

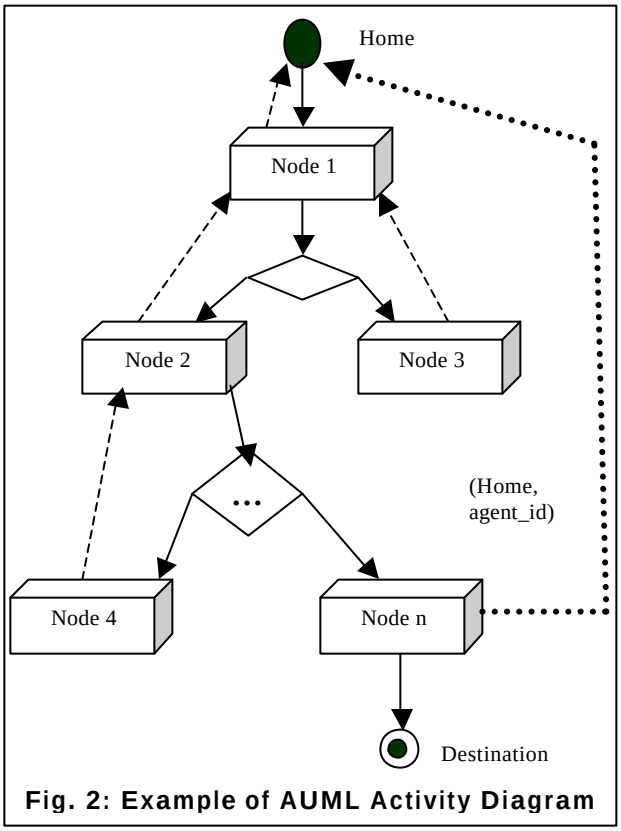

Fig. 2: Example of AUML Activity Diagram

the mobile agent. This can happen in cases in which the agent needs to get more information from the previous node, the judgment about choosing a particular node was not correct, or in case of *Bounce Failure* (the case where a MA tries to move to a particular platform but it is bounced off[1] (by the platform)). This is illustrated in the diagram with the aid of dashed arrow lines. In addition, in some cases a MA goes to a destination platform, it obtains some information, but instead of returning with the information on its home platform, it sends a message with the results (either in its home platform or another platform). This can be illustrated in this diagram with the aid of a dotted line and a statement that has two arguments, the first argument being the platform in which the message has to be delivered and the second being the identification of the MA that sends the message.

5. Scenarios

The following section illustrates with the aid of different scenarios the extensions introduced on the previous section.

[1] Reasons for Bounce Failure include (amongst others): (1) the Mobile Agent platform on the destination address is not operational; (2) the machine that the agent is moving from is isolated from the rest of the network; (3) The Mobile Agent Platform denies the move because of security or operational reasons

5.1. Scenario 1

A Mobile Agent (MA) moves (without moving to any intermediate node) to a platform (location depended information – location depended information means the agent will move to a particular instance) to retrieve some information and it returns to its original platform.

Real life application related to this scenario: A salesman despatches a Mobile Agent (MA) from his laptop to his *Company Server* to get information about his diary. The MA returns back with results.

We are using the extended deployment diagram proposed in the previous section to capture the *mobility paths*, *origin* and *destination* of the MA (figure 3). The same notation (underline) used in UML for capturing instances is employed here.

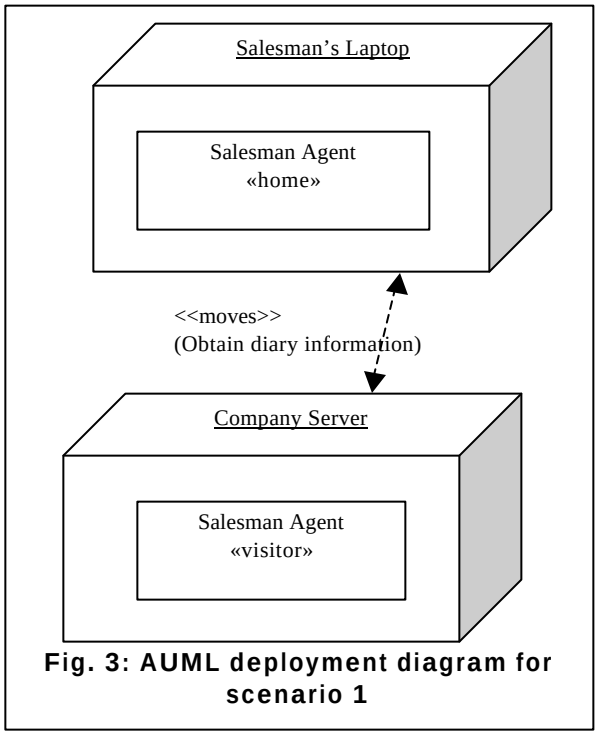

Fig. 3: AUML deployment diagram for scenario 1

Thus we can see from the diagram that the agent's *origin* is the Salesman's laptop and it moves to the *Company Server* in order to obtain the salesman's diary information. However, the above diagram is only a static representation. We also need to show the dynamics of the agent. Thus, we are using the proposed diagrammatic solution of the extended activity diagram with deployment nodes (figure 4). This diagram shows the detailed path that the mobile agent follows in order to reach the targeted platform (*Company Server*)

In addition, we need to show the *when* the MA will leave the *Company Server* and return to the Salesman's laptop, in other words we need to show the activity of the MA on the company server. From the moment it starts execution, until the moment it migrates back to the Salesman's laptop. This can be captured using activity diagrams (Figure 5). The following diagram (figure 5)

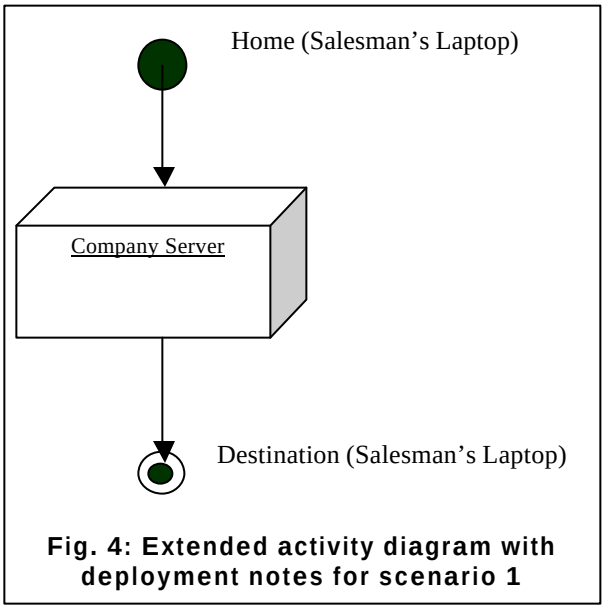

Fig. 4: Extended activity diagram with deployment notes for scenario 1

shows that the mobile agent, when in the company server, first tries to locate the diary information. When it obtains the diary information, it migrates back to the salesman's laptop.

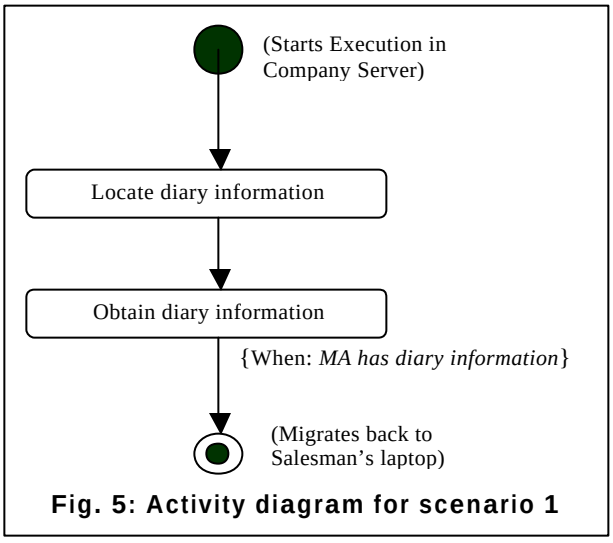

Fig. 5: Activity diagram for scenario 1

5.2. Scenario 2

A MA moves to a platform (non location depended information) to retrieve some information and returns to its origin.

Real Life Application: A traveller despatches a mobile agent to retrieve information about the weather.

In trying to capture the *mobility path* of the MA, differently than the previous scenario, in this scenario the MA is not looking for a specific instance of a platform but for a type of a platform as shown in figure 6.

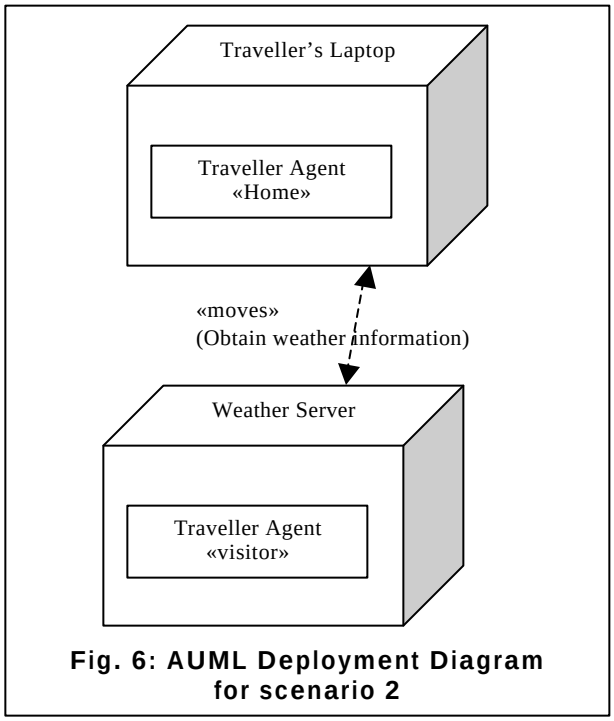

Fig. 6: AUML Deployment Diagram for scenario 2

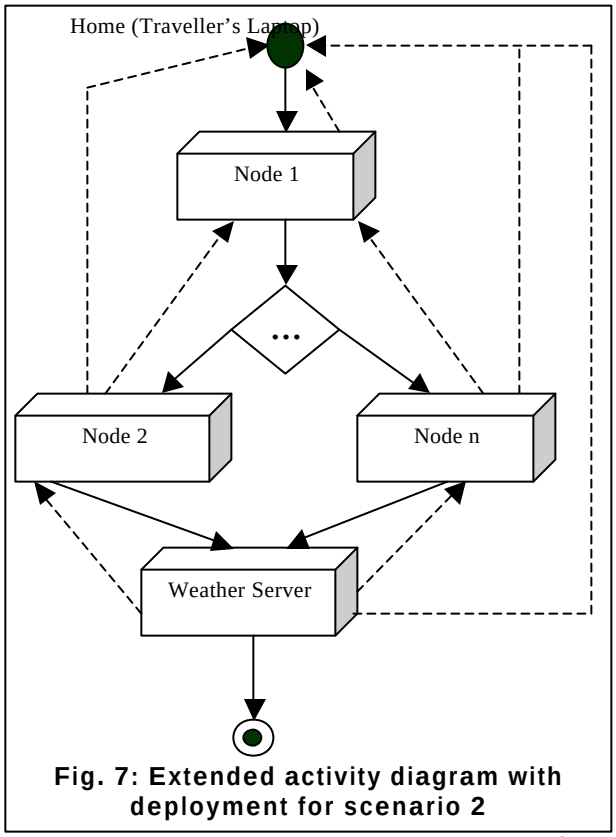

Fig. 7: Extended activity diagram with deployment for scenario 2

Although, the *AUML deployment diagrams* of this and the previous scenarios seem similar, many differences become obvious when trying to model the *detail mobility path* (sequence of the movement) of the MA using an *AUML activity diagram* (Figure 7). In

trying to move from the traveller's laptop to the *Weather Server*, the *traveller agent* might have to go via other nodes of the network. At the design stage we might not know how many nodes or what kind of nodes the *traveller agent* might visit but we know the reasons why the *traveller agent* will choose to visit a particular node over another.

In this scenario we have assumed that the *traveller agent* moves to node 1 from the traveller's laptop. In trying to move further to the network it has to choose from a range of nodes (2…n). The diamond with … (3 dots) in the middle means there are options from node [node the agent currently is +1] to node n. Thus, in our example there are options from node (1+1=2) to node n. The statements that make the agent choose a particular node over another must be indicated. In our scenario could be statements such as *choose node with less traffic*. It is important to note here that for each diamond appeared in the diagram, rules must be stated that affect the agent's decision. After moving through these nodes, the traveller agent reaches its destination node *Weather Server*. When it gets the information it returns to its *origin*. The dashed arrow lines used in the figure indicate the *return path* of the mobile agent.

The next step is to capture the activity of the MA in the *Weather Server* (figure 8). The MA starts execution in the *Weather Server*. First it tries to locate the requested information, and then to obtain it. When the MA has the requested weather information, it migrates back to the traveller's laptop.

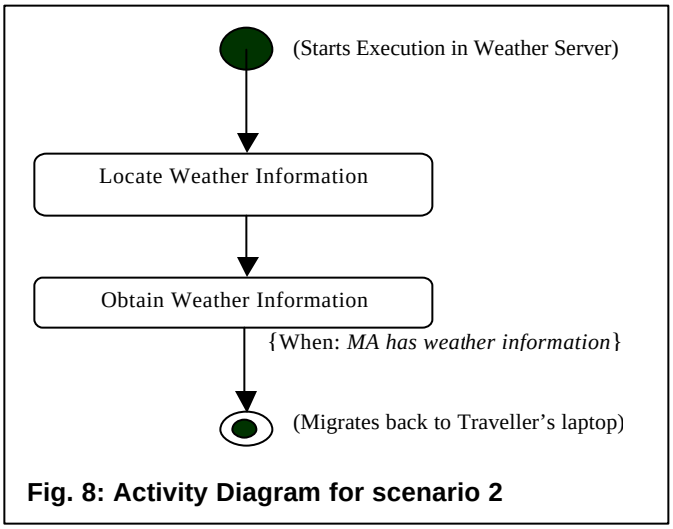

Fig. 8: Activity Diagram for scenario 2

5.3. Scenario 3

A MA moves to a platform and retrieves some information. Then it visits another platform and retrieves different information. Finally it moves to a third platform negotiates with agents on the platform and sends the results of the negotiation back to its user using a message.

Real Life Application: A traveller despatches a mobile agent to book her flight tickets. The agent first contacts the *air-company reservation system* to enquire about ticket prices, then it visits the traveller's *bank server* to get authorisation to pay for the ticket, and then

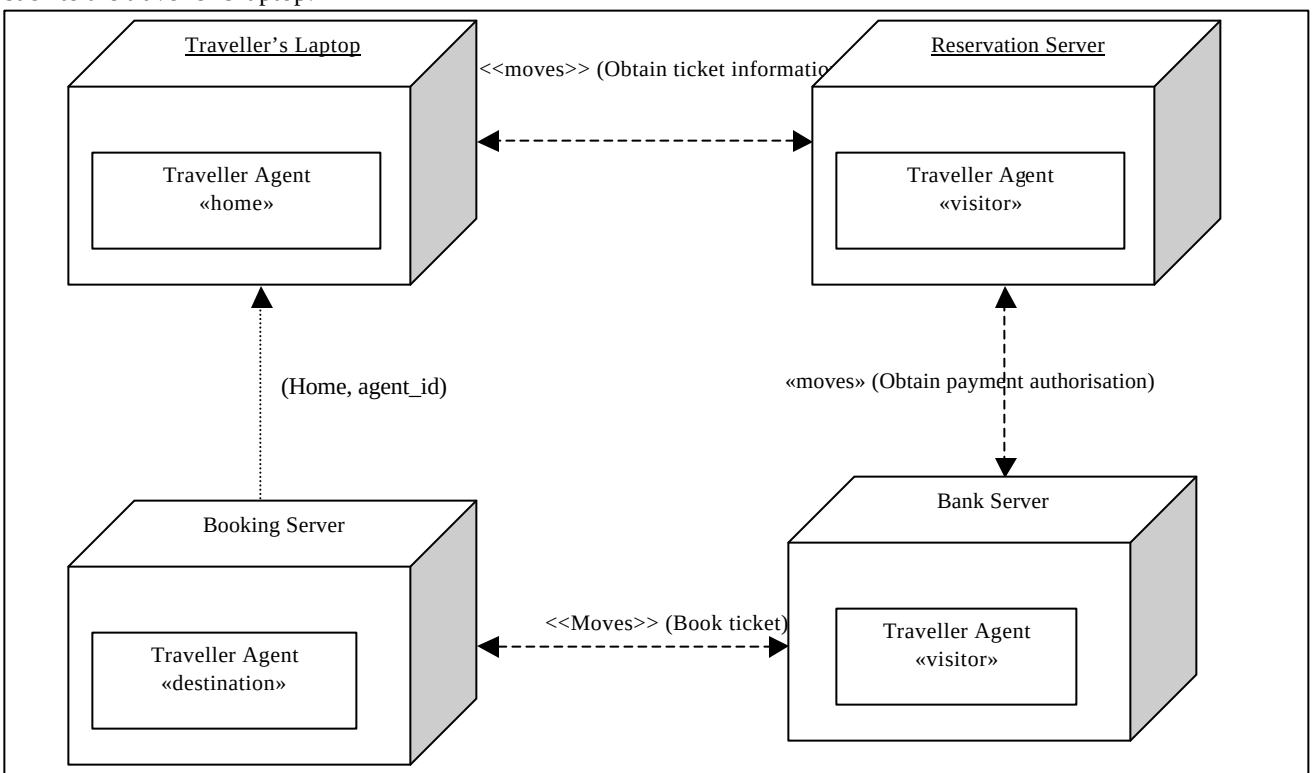

Fig. 9: AUML Deployment Diagram for scenario 3

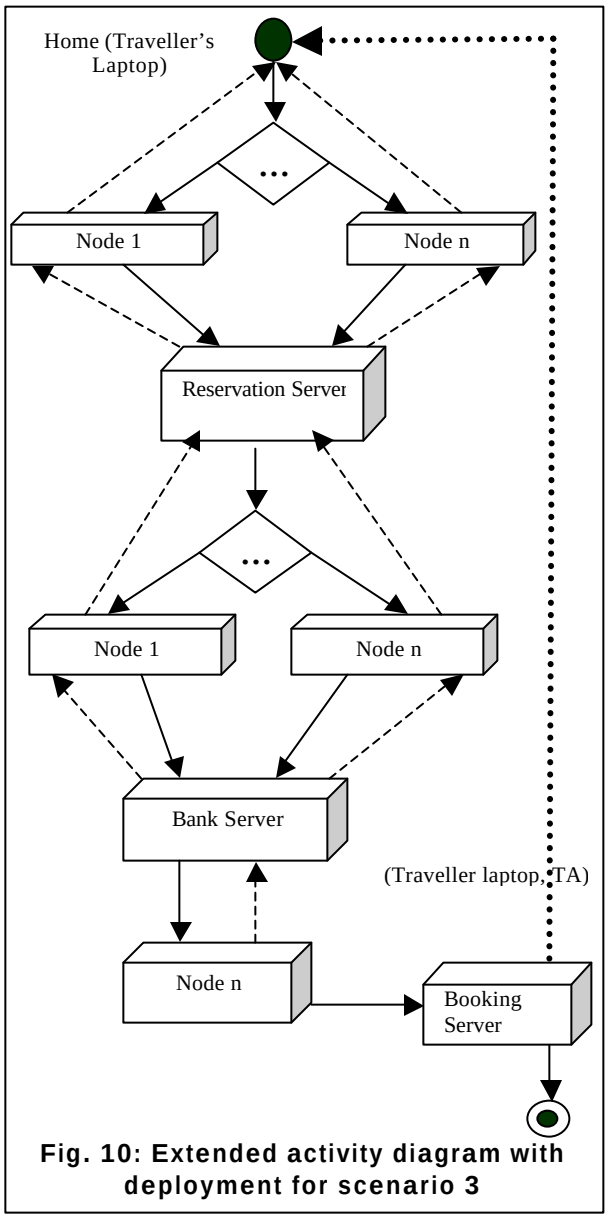

Fig. 10: Extended activity diagram with deployment for scenario 3

visits the *air-company's booking system* to book the ticket and pay for it. It then sends a message to the traveller with the flight details.

An *AUML deployment diagram* is employed to capture the *origin*, the *mobility path* and the *destination* of the *Traveller Agent* (*TA*). The *TA* first visits the *Reservation Server*, to obtain ticket information. Then it moves to the *Bank Server* to obtain payment authorisation and then it moves to the *Booking Server* to book the ticket. The booking server is the *destination* of the *TA*. In addition, the *TA* sends a message to the traveller to inform her about the booking details. The *AUML deployment diagram* of the system is shown in figure 9. Double arrow paths are indicated since we assume that the agent might need to move between the servers backwards in order to obtain some information that it is missing. For example, while in the *Reservation Server*, the *TA* might not be able to find an available flight within the traveller's proposed dates. Thus the *TA* will move back to its home and obtain a new set of proposed dates. It is worth mentioning that the *Booking Server* is considered to be the destination of the TA, so the tag *destination* has replaced the tag *visitor* in this platform.

When the designer knows the *origin*, *mobility path* and *destination* of the *Traveller Agent*, the next step is to indicate the dynamics such as the sequence of the movement, details of the path that the *TA* will follow, and the decisions that the *TA* will take in order to reach its *destination* (Figure 10).

Trying to move from the traveller's laptop to the *Reservation Server*, the *Traveller Agent* has to decide which path it will follow. Although at this stage the designer might not know if the agent will go to node 1 or node n, knowledge can be given to the agent to help in the decision. This knowledge, as previously mentioned, it comes in the form of statements. These statements will be implemented during the implementation stage to the agent's knowledge database (its beliefs). As an example the following statements can be used to determine the node selection in our example.

1. *Use node with less traffic*
2. *Use node with less round trip*
3. *Use node of a particular type*

It must be noticed that the rules in a decision statement must be outlined in terms of importance. Thus in our example, the agent first will check for the node with the less traffic. If it finds for example two nodes with exactly the same traffic, it will proceed to the next rule and so on.

When the agent will try to move from the *Reservation Server* to the *Bank Server*, it has again to decide which node to follow. This decision, might be taken by considering the rules that already have been defined, or the designer might add some rules according to the type of information and knowledge the *TA* will have gained from the *Reservation Server*. When the *TA* arrives at the *Booking Server*, it sends a message to its home. This message takes two parameters. The platform that the message will be delievered and the name of the agent. The message is illustrated in figure 10 as a dotted line.

So far we have captured the origin of the *TA*, *why* the *TA* moves from one platform to another, the platforms it will visit, its *mobility path*, indicating rules for helping *TA* to choose a node over another, and also its *destination*. However, we have not capture *when* TA moves from a platform to another, and the activities on each platform.

TA visits three platforms, the *Reservation Server*, the *Bank Server*, and the *Booking Server*. Thus three activity diagrams are employed to capture the activities of the *TA* in each of the platforms.

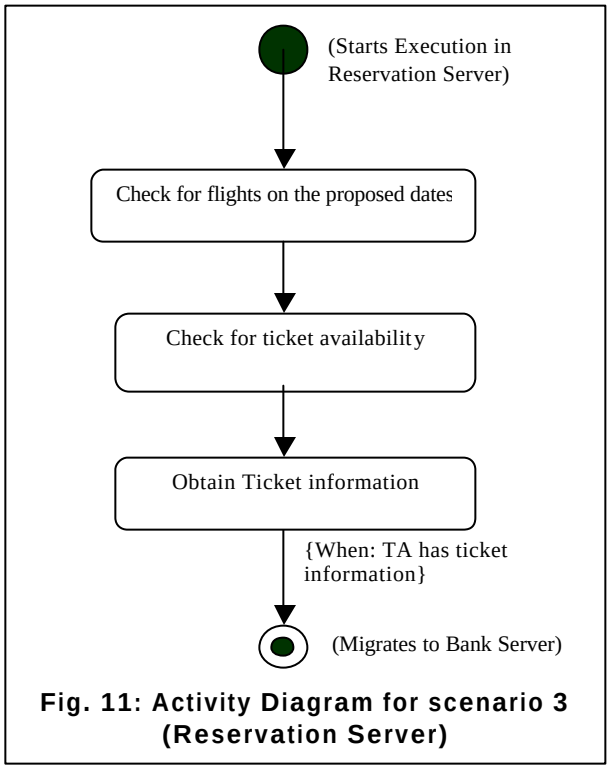

Fig. 11: Activity Diagram for scenario 3 (Reservation Server)

When the *TA* starts execution in the *Reservation Server*, it tries to identify tickets according to the proposed dates specified by the traveller. When such a ticket is found, it obtains more information about it, such as airline, price and so on. When it has obtained all the necessary information, it migrates to the *Bank Server*. Figure 11 shows the activities of the TA in the *Reservation Server*. It must be noticed that we have assumed (for reasons of simplicity) the *TA* finds a ticket available for the proposed dates.

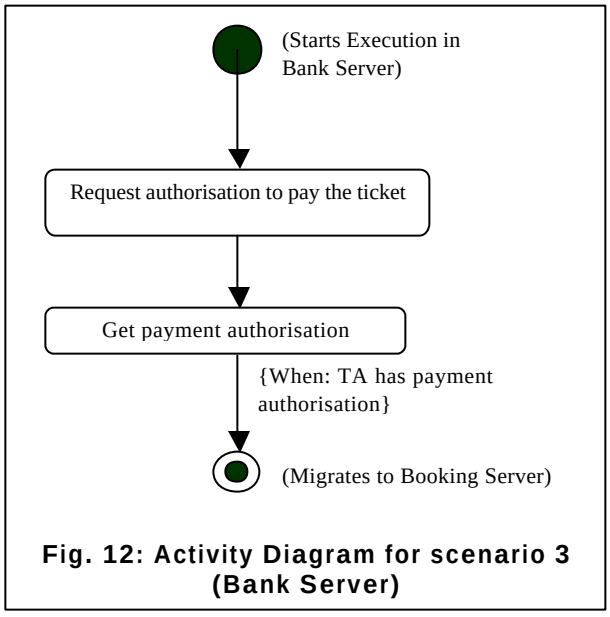

Fig. 12: Activity Diagram for scenario 3 (Bank Server)

When the *TA* leaves the reservation server, it will migrate to the *Bank Server*. In the *Bank Server* the *TA* requests an authorisation to pay the available ticket. When it gets the authorisation, it migrates to the *Booking Server*. We have assumed (for reasons of simplicity) that the authorisation is given to the *TA*. The activities of the *TA* in the *Bank Server* is shown in figure 12.

Finally the *TA* reaches its *destination*. It checks if the ticket is still available, and it starts the booking and payment procedures. When the ticket has been booked the *TA* sends a message to the traveller to inform her that the ticket has been booked and also it provides the details of the ticket. This is shown in Figure 13.

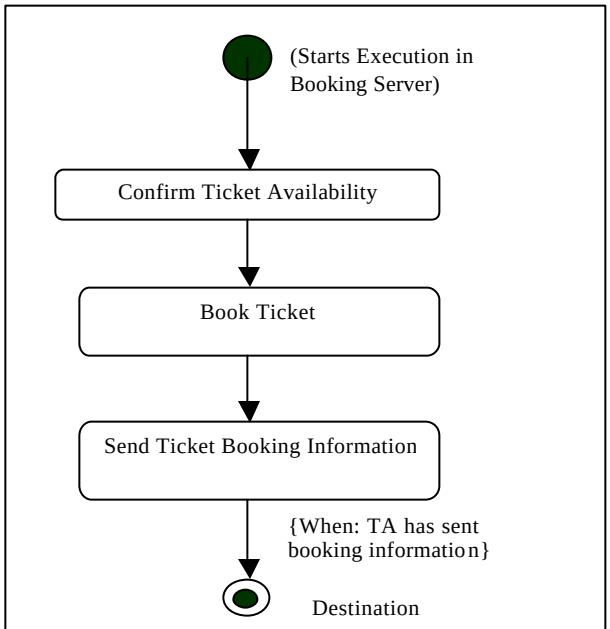

Fig. 13: Activity Diagram for scenario 3 (Booking Server)

6. Related Work

Research on providing concepts and notations in order to model mobile agents is very limited. As a matter of fact, the authors are aware of only two attempts.

In the first work [22], the MaSE [11] methodology was modified to allow for the analysis and design of multiagent systems using mobile agents. Different options for integrating mobility into the analysis and design phases of the methodology were examined and transformations were defined both for the analysis and design phases. The extensions were justified with the aid of an example scenario.

In addition Klein et al have proposed some extensions to UML for mobile agents [12]. In this work, they are extending the Unified Modeling language by providing some language concepts to model mobility during the analysis and design stages. Klein et al have partially extended the UML notation by introducing a number of different stereotypes that can be used on modelling mobile agents. The proposed extensions have been demonstrated using a prototype implementation of an advanced telecommunication system. It must be noticed, that the proposed concepts are mainly based on notation specific to the Grasshopper [www.grasshopper.de] mobile agents platform.

Differently than the above two approaches, in our work we have decided to provide more generic extensions that can be applied to many different cases and are not specific either to a methodology or a mobile platform.

7. Conclusions and Future Work

In this paper, we have indicated how UML can be used to express agent mobility, and we have also introduced extensions to the standard UML where appropriate. Our work involves extensions at the Deployment and Activity diagrams of UML to model issues related to mobile agents such as *why a mobile agent moves from one platform to another*, *where the agent moves to*, *when the agent moves* and *how it reaches the targeted platform*. The proposed extensions have been demonstrated with the aid of three different scenarios.

However, this work is by no means complete. More work is required in order to further justify the concepts and notations we are using in our extensions. In doing so, the proposed extensions must be employed in modelling different kind of systems and applications.

In addition, the need for a systematic approach towards the choice or not of mobile agents on a particular system have been recognized [12]. We are working towards this direction in order to provide a complete systematic process that will help developers to justify the reasons behind the choice or not of mobile agents at the development of a specific agent based system. The choice or not of mobile agents should be based not only on general advantages or disadvantages of the mobile agent technology but also on rules that will determine whether mobile agents should be used and with what functionality.

Also, concerns related to security are increased when using mobile agents. Thus, another important direction for future work is the definition of security on the analysis and design of mobile agent systems.

Taking into consideration the above-mentioned issues for future work, we believe it will advance current agent oriented methodologies and will provide designers with more functionality in developing agent-based systems.

8. Acknowledgements

The first Author would like to thank the RANK Foundation for the funding of his research project during which this work was carried out.

9. References

[1] N. R. Jennings, "An agent-based approach for building complex software systems", Communications of the ACM, Vol. 44, No 4, April 2001

[2] M. Wooldridge, P.Ciancarini, "Agent-Oriented Software Engineering: The State of the Art" In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*. Springer-Verlag Lecture Notes in AI Volume 1957, January 2001

[3] A. Caglayan, C. G. Harrison, *Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents*, John Wiley, 1997

[4] M. Wooldridge, N. Jennings, "Intelligent agents: theory and practice", *The knowledge Engineering Review*, 10, (2), pp 115-152, 1995

[5] J. Fox, "Understanding intelligent agents: analysis and synthesis", Proceedings of the Agents Applied in Health Care workshop, 15th European Conference on Artificial Intelligence, Lyon-France, July 2002

[6] W. Brenner, Z. Rüdiger, W. Hartmut, *Intelligent Software Agents: Foundations and Applications*, Springer-Verlag, pp. 55-67, Berlin, 1998

[7] J. E. White, "Mobile Agents", *Software Agents*, Jeffrey Bradshaw ed., MIT Press, Cambridge, MA, 1997, pp. 437-472.

[8] M. Wooldridge, N. R. Jennings, D. Kinny, "The GAIA methodology for agent-oriented analysis and design", *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, No 3, pp 285-312, 2000

[9] J. Castro, M. Kolp and J. Mylopoulos. "A Requirements-Driven Development Methodology," In Proc. of the 13th Int. Conf. On Advanced Information Systems Engineering (CAiSE'01), Interlaken, Switzerland, June 2001.

[10] C. Iglesias, M. Garijo, J. Gonzales, "A survey of agent-oriented methodologies", *Intelligent Agents IV*, A. S. Rao, J. P. Muller, M. P. Singh (eds), Lecture Notes in Computer Science, Springer-Verlag, 1999

[11] M. F. Wood, S. A. DeLoach, "An Overview of the Multiagent Systems Engineering Methodology (MaSE)", The First International Workshop on Agent-Oriented Software Engineering, 2000

[12] C. Klein, A. Rausch, M. Sihling, Z. Wen, "Extension of the Unified Modeling Language for Mobile Agents",*In Unified Modeling Language: Systems Analysis, Design and Development Issues*, edited by Keng Siau and Terry Halpin; Idea Group Publishing Book, 2001

[13] H. Mouratidis, P. Giorgini, G. Manson, I.Philp, "Using Tropos Methodology to Model an Integrated Health Assessment System", Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Toronto-Ontario, May 2002

[14] R. Gray, D. Kotz, S. Nog, D. Rus, G. Cybenko, "Mobile agents for mobile computing", Technical Report PCS-TR96-285, Department of Computer Science, Dartmouth College, 1996

[15] D. Chess, C. Harrison, A. Kershenbaum, "Mobile Agents: Are They a Good Idea?", IBM Research Report (RC 19887), 1995

[16] G. Vigna, *Mobile Agents and Security*, Lecture Notes in Computer Science 1419, Springer-Verlag, 1998

[17] J. Altmann, F. Gruber, L. Klug, W. Stockner, E. Weippl, "Using Mobile Agents in Real World: A Survey and Evaluation of Agent Platforms", Second Workshop on Infrastructure for Agent, MAS and Scalable MAS, Autonomous Agents 2001

[18] A. Macro, J. Buxton, *The craft of software engineering*, International Computer Science Series, Addison-Wesley Publishing Company, 1990

[19] Ian Sommerville, *Software Engineering* – Sixth Edition, Addison-Wesley Publishing Company, 2001

[20] J. Martin, J. Odell, *Object-Oriented Methods: A Foundation*, (UML edition), Prentice Hall, Englewood Cliffs, NJ, 1998.

[21] B. Bauer, "Extending UML for the Specification of Interaction Protocols", submitted for the 6th Call for Proposal of FIPA, 1999.

[22] A.L. Shelf, "Design and Specification of Dynamic, Mobile and Reconfigurable Multiagent Systems", Thesis, Graduate School of Engineering and Management of the Air Force Institute of Technology, AFIT/GCS/ENG/01M-11, March 2001