

# Modelling Mobile Agent Applications in UML2.0 Activity Diagrams

Miao Kang\*, Lan Wang and Kenji Taguchi

*School of Computing, Leeds Metropolitan University\**

*Department of Computing, School of Informatics, University of Bradford*

## Abstract

*Mobile agent technology has gained more importance recently due to the rapid advance of high-performance mobile devices with high-speed wireless telecommunications technology such as 3G. However design methodologies for mobile agent applications are not mature enough to support their software development process.*

*We proposed an extension of Activity Diagrams in UML 1.5 for modelling mobile agent applications and presented a case study in [1]. However we found that the latest version of UML 2.0, which provides better model elements in Activity Diagrams, is more effective and flexible in modelling mobile agent applications. This observation prompted us to upgrade our work in UML 1.5 to UML 2.0.*

*In this paper, we will demonstrate how UML 2.0 Activity Diagrams can be used for modelling mobile agent applications and discuss their underlying computational models which capture mobility of agents.*

## 1. Introduction

The advent of network technology has prompted new computing and communication paradigm in which codes and processes can move over the network. Mobile agents are autonomous computing entities (codes/processes), which can autonomously decide where to move over the network and what tasks to perform at different hosts. A large number of programming languages, systems and APIs have emerged to support this paradigm, e.g., Telescript [6], Aglets [3], and JADE [5] to name a few.

There has been an emerging trend to use UML (Unified Modeling Language) for modelling agent-based systems in general. The most notable work in this arena is AUML [13] in which UML is extensively used to model all aspects of agent. Although we generally follow the rationale adopted in AUML, our work specifically focuses on design issues on mobile agents.

In our previous work [1], we have extended Activity Diagrams in UML 1.5 [4] to capture algorithmic

behaviour of mobile agents with location based on a new structural interpretation of swimlanes. Indeed, locality is the most crucial abstraction of mobile agents which imposes several constraints on their behaviours. For instance, the agent place model in [6] only allows local communications between agents at a common location, hence it is essential for agents to synchronise their moves as well as their communications at locations. Indeed, the mantra for mobile agents is “*location, location, location*”.

The latest enhancements of UML in 2.0 [2] will provide new concepts for modelling, and their associated new model elements which can be readily used for various system developments. In this paper, we will demonstrate how Activity Diagrams in UML 2.0 can be readily used to model dynamic behaviours of mobile agent systems and point out why they are effective for them from its underlying computational model. Since it is crucial to found a notation on the sound theoretical basis rather than on the simple yet visual appealing. If the underlying computational model is poorly understood and not well-founded, it is hard to use the notation in a concise and rigorous manner.

This paper is organised as follows: in section 2, we overview Activity Diagrams in UML 2.0 and illustrate how they can be used to model mobile agent applications by some example case studies. Section 3 compares related work and we will conclude our work in the final section.

## 2. UML 2.0 for mobile agent applications

### 2.1. Overview

In our previous work [1], we presented an effective new approach to model specific features of mobile agents by extending UML 1.5 Activity Diagrams [4]. We introduced a new stereotype <<Host>> + *parameter* for a swimlane, which represents the location with a unique name (address) as a parameter in order to capture mobility of agents. Other specific features of agents such as communications, cloning were defined by existing model elements with a new rule for subactivities. In that work, we also proposed a new notation *subswimlanes* in order to model nested locations.

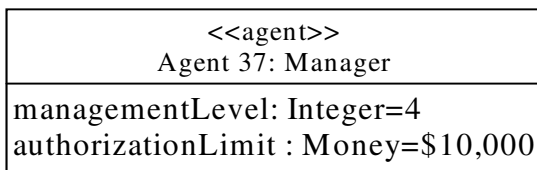
In this paper, we will use UML 2.0 Activity Diagrams to model the dynamic behaviour of mobile agents. Specific features of mobile agents, which we model, are mobility, cloning, messaging between agents. In this new approach we will extensively use activity partitions newly introduced in UML 2.0 under the different semantic interpretation.

An activity partition is a kind of activity group for identifying actions that have some characteristic in common. A partition may be represented by an attribute and its sub-partitions by values of that attribute. Hence locality is naturally captured in a partition as its attribute. And at the same time, the location attribute could be on the process class associated with an activity so that this is not an attribute of a partition [2].

This underlying semantics of activity partitions correspond to that of formalisms such as MobileUNITY [10] in which mobility is captured by the change of values in a particular variable for location. However, in real mobile agent applications, we should separate applications and run-time systems, which control and manage resources, security, communications and *addresses* for agents. Hence the location is better attributed to run-time systems rather than to agents. This semantic interpretation corresponds to mobile calculi such as  $D\pi$  [12] and Mobile Ambient [7] in which processes and locations are separate syntactic components. We will adopt the latter interpretation for activity partitions in this paper.

UML 2.0 further allows multidimensional hierarchical partitions in which one dimension represents a location and another orthogonal dimension represents an object. We will use this multidimensional hierarchical partition to model location and agent respectively.

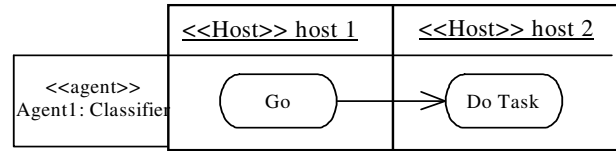
We will follow our previous work in which a location is represented by the stereotype `<<Host>>` and parameter as its unique name (address). We also need a particular notation which represents an agent as a class. We will follow [14], which proposes `<<agent>>` as a stereotype in the object notation to denote an agent class and its instance (see Figure 1).



**Figure 1. Agent Class (from [14])**

The following multidimensional partition models an agent Agent1, which moves from location “host1” to “host2” by using the “Go” activity. As we have adopted different interpretation of partition which represents a location, it is essential to represent mobility of agent by a

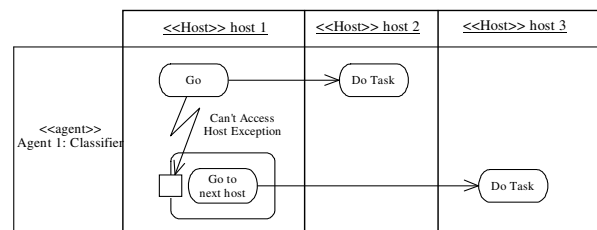
particular activity such as “Go”. This approach is not only reflected in its underlying semantics but also provides a smooth path to implementation, as almost all mobile agents programming languages, systems and APIs offer a programming primitive which controls mobility.



**Figure 2. “Go” action**

In our previous work, an Activity Diagram models the algorithmic behaviour of a single multi-threaded agent; hence it is required to compose Activity Diagrams in order to verify synchronization of moves and communications between agents at a location. However in UML2.0 Activity Diagrams, agents can be allocated to one dimension and locations are represented by orthogonal dimension so that the behaviour of multiagents are easily captured and visualized. Another example is given in Appendix 1, in which a master agent controls synchronized move of other slave agents from one host to another by exchanging messages.

In UML 2.0, exception handling is modelled by ExceptionHandler, which replaces jump handlers in UML 1.5. The following example, in which exception of failing to access to the destination host is modelled, depicts how this model element can be used.



**Figure 3. “Go” action with Exception captured**

We will use the same model elements for communications between agents in our previous work except object node, which basically represents data or objects flowing between control nodes. *Signal sending* and *Signal receipt* in UML Activity Diagrams enables agents to exchange messages, and synchronization of messaging is achieved only when the parameters (message description) are exactly the same between signal sending and expecting signal receipt.

Agents need communication protocol to exchange messages. JADE (Java Agent Development programming language) [5], which is compliant with FIPA, uses a class

ACLMessage, which represents Agent Communication Language (ACL) messages. This class defines a set of constants e.g., REQUEST, INFORM, QUERY that should be used to refer to the FIPA performatives.

In order to be able to model mobile agent applications in existing standards and in programming languages/systems/APIs compliant to those standards, it is essential to provide pre-defined stereotypes, which represent classes defined in them. Therefore we will propose to use particular pre-defined stereotypes; in this case, <<ACLMessage>> which denotes the message class is the ACLMessage, to explicitly specify what agent communication protocol is used in the model. The following example depicts that the message type of receiving and sending is the ACLmessage and its performative is "REQUEST".

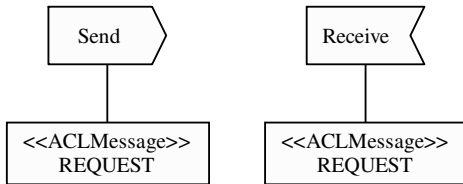


Figure 4. Communication between agents

Cloning is another important behaviour of mobile agents, which includes several subactivities. This activity allows an agent to clone it first, then to send the cloned agent to another host. This behaviour may be repeated in order to send the number of clones required.

We will use the invoking activities notation in UML 2.0 that have nodes and edges inside. The call of an activity is indicated by placing a rake-style symbol within the symbol. The rake resembles a miniature hierarchy, indicating that this invocation starts another activity that represents a further decomposition.

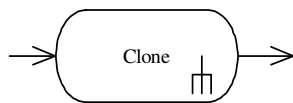


Figure 5. Clone Action

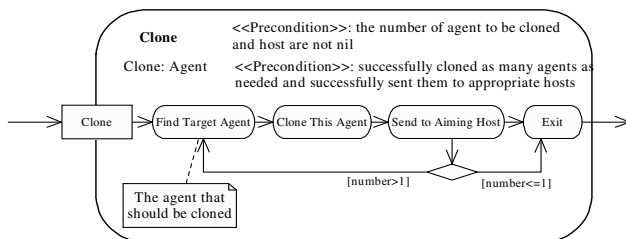


Figure 6. Details of Clone Action

The above clone action is modelled by subactivities in figure 6. This activity begins by finding the target agent to be cloned, which may be another agent or its own. The cloned agent will be sent to the aiming host. This is repeated until there is no agent needed to be cloned.

Another new notation which is used in our approach is the accept time event action in UML 2.0. This notation is used to capture the time out exception.

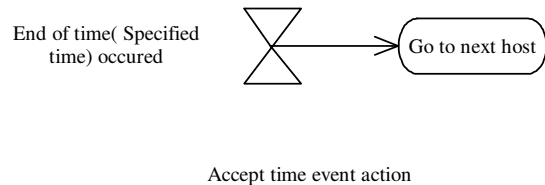


Figure 7. Time Out Notation

The above example shows that if the specified time elapsed, the agent moves to the next host.

## 2.2. Case Study

In order to demonstrate applicability of our approach as a design notation, we will present an auction system as a case study, which is designed for JADE. This case study consists of two agents; an Auctioneer agent, who is stationary, and a Bidder mobile agent. The Auctioneer agent resides at seller's host and manages a GUI interface to the human seller to show the state of the auction, and controls the whole process. The Bidder mobile agent will circulate round potential bidders while gathering bids by displaying it on GUI to human bidders.

For brevity's sake, we will attach the whole diagram in Appendix 2, and only show part of the activities of Bidder Mobile Agent, which are called part 1 and part 2 in Figure 8 and 9.

Every time the Bidder agent moves from one host to another, it needs to update the list of locations currently available. The figure 8 models that the Bidder agent moves to the host of one bidder, and requests a list of available locations. It then sends back an <<ACLMessage>> "REQUEST" type message to the Auctioneer agent, and waits for the information saved by that agent. It then waits for feedback from the bidder. After it finished this task, it will move to the next bidder's location.

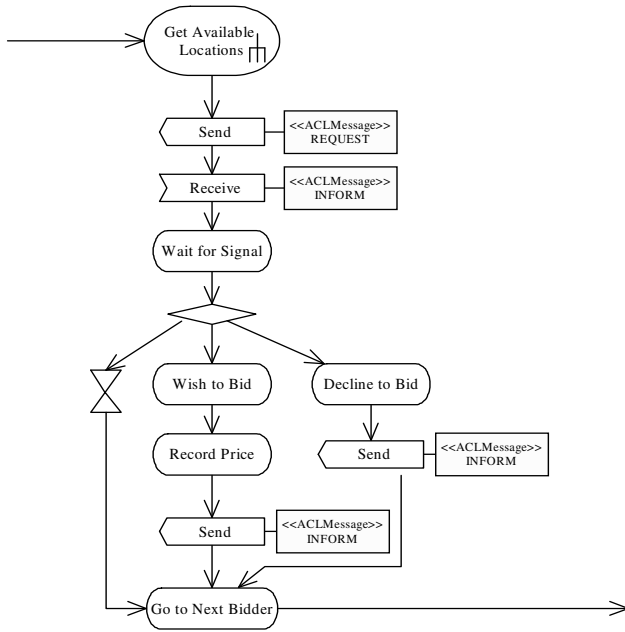


Figure 8. The activities for bid (Part1)

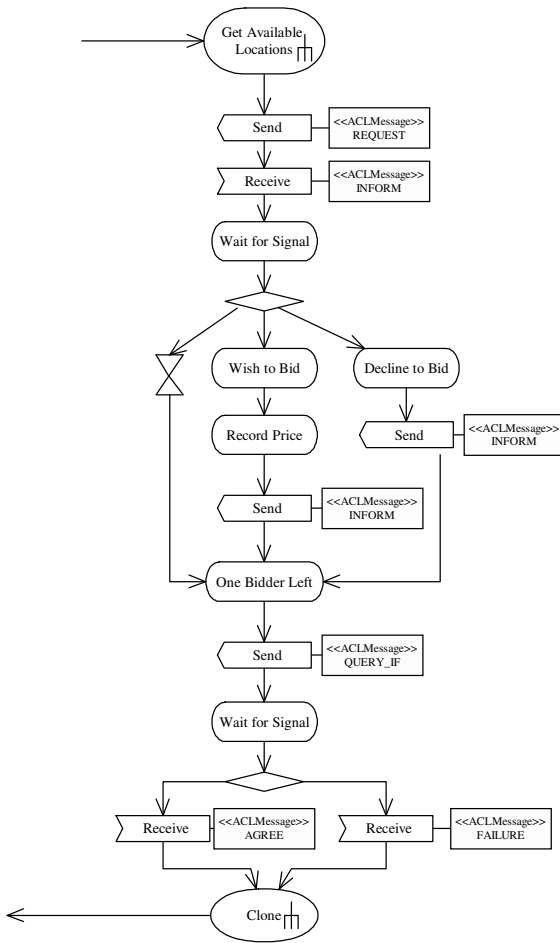


Figure 9. The activities for bid (Part2)

The figure 9 models the behaviour of the Bidder mobile agent, which is the same as in the figure 8 except how the last bidder on the list of locations is dealt with. The Bidder mobile agent moves to the bidder who is the last on the list of locations and is informed that only one bidder left. After receiving the feedback from the bidder, it will clone itself and send it to the location of the bidder who bade the highest price to inform the bid was succeeded.

### 3. Related work

UML is widely used in industry, but the most fundamental problem as a notation is its lack of rigorous semantics [18]. We will justify our approach in UML 2.0 Activity Diagrams by deriving its underlying computational model from an existing rigorous formalism.

There are several calculi such as Mobile Ambient [7] and  $D\pi$  [12], which have been proposed to model and analyze the mobile computing in general and mobile agent technology in particular. The main reason why they are hardly used as a modelling notation is that although those formalisms are elegant and can capture the semantics of mobile agent behaviours, they generally cannot scale up for modelling complex mobile agent systems which often have complex data and state properties as well as mobility of codes and/or processes. The third author of this paper proposed an integrated notation of mobile primitives and a state-based formal specification language Object-Z to remedy this problem [11].

UML provides structural diagrams for modelling state properties and behavioural diagrams for modelling dynamic behaviours of systems, which makes it the best tool for the system design. However we cannot ignore those formalisms for founding the semantics of semi-formal/graphical notation like UML. In fact, our approach in UML 2.0 Activity Diagrams is based on mobile calculi such as  $D\pi$  but some related works, which we will compare in this section, such as [8], [9] are based on Mobile Ambient.

An Ambient is a bounded place (location) where computation happens and can move as a whole while other ambient inside it can run in parallel. An Ambient can be nested and form a tree structure, which can be dynamically changed.  $D\pi$  is an extension of the  $\pi$ -calculus [17] and its network topology is flat. In  $D\pi$  only processes can migrate between places (locations), and communications are only allowed locally. Mobile Ambient is suitable for mobile computing systems in general, since it provides simple primitives to encode communications, security mechanisms, etc, but not readily used for mobile agent systems because of its nature as being a generic framework.  $D\pi$  is more faithful to the agent place model, which almost all implementations of mobile agent programming languages, systems and APIs use.

Baumeister, et. al., [8] extended UML Class diagrams and Activity diagrams to model mobile systems in which both mobile objects and locations can migrate to another location. They introduce two notational variants of Activity Diagrams for modeling mobility. One variant is location centered and focuses on the topology of locations. The other one focuses on actor responsible for an activity. In their notation, a swimlane is used to denote an object, and mobility of an agent is captured by an object denoted by a swimlane, which evolves to an object with a different location by an activity with a stereotype <<move>>.

Kosciuczenko [9] used Sequence Diagrams instead of Activity Diagrams for modelling mobile systems in general. His *Sequence Diagram for Mobility* models migration of objects, interaction between objects and the network topology of nested objects. The communication between two mobile objects can be expressed using the arrow symbol used in Sequence Diagrams. A jump arrow with a fixed location is used to model the migration of objects.

It must be pointed out that even their works are based on Mobile Ambient; they are not a faithful interpretation of Mobile Ambient in the sense that all primitives and constructs in Mobile Ambient are not encoded in those diagrams.

The main difference between our work and theirs is now clear. The underlying computational models are different, which are also reflected in the use of notations. Our notation can be readily used for mobile agent applications mainly due to a faithful match of underlying computational model between the modelling notation and implementations.

MobileUNITY notation and logic is the result of a careful re-evaluation of the implications of mobility on UNITY [10]. In addition to the structure of a UNITY (a declare section, an initially section and an assign section), mobile UNITY adds four constructs (Transaction, label, inhibitor and reactive statement) to model mobile system. Transaction is a set of assign-statements, which forms a nested relationship of activity. The combination of inhibitor and label avoids the execution of a statement when it is undesirable and captures the semantics of processing dependencies. At last reactive statement makes a program in an execution order as to execute to fixed-point (the reactive statement) after each an assignment statement and captures the semantics of interrupt processing.

As was mentioned in the previous chapter, the original interpretation of active partitions of UML 2.0 is similar to MobileUNITY in which mobility is captured by the change of values (constants for location) in a particular variable for location; this does not match the real implementations of mobile agent systems.

There are some other related works which are not clearly linked to underlying computational models.

The FIPA modelling standard in deployment and mobility of multiagent systems [15] uses Activity Diagrams and Deployment Diagrams as part of AUML. AUML Deployment diagrams statically models mobility of agents by incorporating a new association with a stereotype <<moves>> and nodes in a network in which agents are modelled as software components. Algorithmic behaviour of an agent is modelled in an ordinary Activity diagram, but mobility is captured by using symbols for a node and a decision which an agent will take in order to migrate different nodes in a network.

It should be pointed out that Deployment Diagrams used is not very helpful in capturing movement of mobile agents, since it can only model static links between hosts. The approach is weak at visualizing mobility and specifying algorithmic behaviour of mobile agents at the same time, which our notation can do.

Lind adopted UML Activity Diagrams to model agent interaction protocol without introducing any new modelling elements [16]. He used a swimlane as a construct which denotes a role of an agent by giving a stereotype <<role>>. Synchronous communications are described by allocating a swimlane with a stereotype <<channel>> and the fork/joint node in it. His approach uses existing UML concepts only and requires no additional modelling elements, which makes it easy for UML users to understand the notation without having learnt a completely new type of diagram. However, as mobile agent systems being inherently multiagent, it is hard to use his notation for more complex communication patterns, which require more channels between different agents.

## 4. Conclusion and future work

The main motivation of this paper is to upgrade our previous work [1] in UML 1.5 Activity Diagrams to UML 2.0 Activity Diagrams and to accommodate specific features of mobile agents in newly introduced model elements in UML 2.0, which provides an effective modelling method for mobile agent applications. We used multidimensional hierarchical partitions to model locations and agents respectively, which help to visualize the algorithmic behaviour of multiagent systems with locality.

On the contrary to AUML which uses UML to model all aspects of agent, our work specifically focuses on design issues on mobile agents, and we hope our work will supplement AUML in this respect.

We compared related works from their underlying computational models and pointed out that our interpretation of UML 2.0 Activity Diagrams is based on mobile calculi such as  $D\pi$ .

As our future work, we are planning to formalize our interpretation of UML 2.0 Activity Diagrams in a mobile calculus, which will provide not only the semantic foundations but also the mechanical verification procedure of the diagrams.

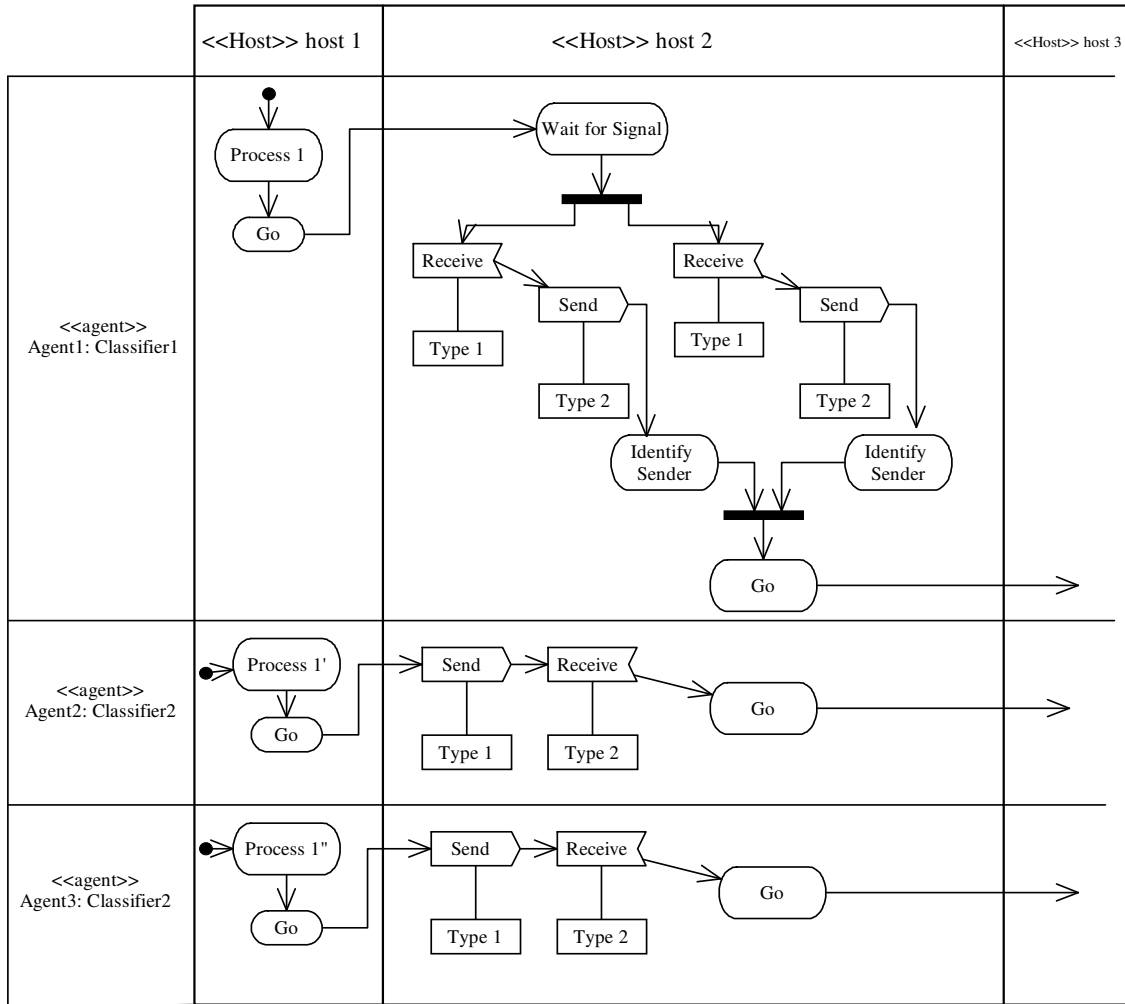
## 5. Acknowledgement

We would like to thank J. Odell for his kind advice to guide us to UML 2.0.

## References

- [1] M. Kang and K. Taguchi, "Modelling Mobile Agent Applications by Extended UML Activity Diagram", *To appear in Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS)'04*, Porto, Portugal, April 2004.
- [2] OMG, UML 2.0 Superstructure Specification, <http://www.omg.org/docs/ptc/03-08-02.pdf>, August 2003.
- [3] D. Lange, and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, London, 1999.
- [4] OMG., *Unified Modelling Language Specification* (version. 1.5), UML Revision Task Force, 2002.
- [5] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE: a FIPA2000 compliant agent development environment", *In Proceedings of the fifth international conference on Autonomous agents (Agent)'01*, Montreal, Canada, 2001, pp. 216-217.
- [6] J. White, "Mobile Agents". *In Software Agents*, J. Bradshaw, editor, MIT Press, 1996, pp. 437-472.
- [7] L. Cardelli and A. D. Gordon, "Mobile Ambients", *Theoretical Computer Science*, vol. 240/1, June 2000, pp. 140-155.
- [8] H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing, "Extending Activity Diagrams to Model Mobile Systems" *In Proceedings of International Conference NetObjectDay (NODe) '02*, LNCS 2591 Springer, Germany, Oct 2002, pp. 278-293.
- [9] P. Kosiuczenko, "Sequence Diagrams for Mobility", *In Proceedings of Advanced Conceptual Modeling Technique (ER) '02*, LNCS 2784 Springer, Finland, Oct 2002, pp. 147-158.
- [10] G. C. Roman, P. J. McCann, and J. Y. Plun, "Mobile UNITY: reasoning and specification in mobile computing", *In Transactions on Software Engineering and Methodology*, vol. 6/3, ACM, 1997, pp. 250-282.
- [11] K. Taguchi, and J. S. Dong, "An Overview of Mobile Object-Z", *In Proceedings of 4<sup>th</sup> International Conference on Formal Engineering Methods, (ICFEM)'02*, Shanghai, China, LNCS 2495 Springer 2002, pp. 144-155.
- [12] M. Hennessy, and J. Riely, "Resource access control in systems of mobile agents", in *Information and Computation*, vol. 173, 2002, pp. 82-120.
- [13] B. Bauer, J. P. Muller, and J. Odell, "Agent UML: A Formalism for Specifying Multiagent Software Systems", *In Proceedings of the First International Workshop on Agent-Oriented Software Engineering AOSE'00*, Limerick, Ireland, LNCS 1957 Springer , 2001, pp. 91-103.
- [14] J. Odell, R. Levy, M-P. Huget and M. Nodine, "FIPA Modelling: Agent Class Superstructure Metamodel", FIPA Modelling TC, 2004.
- [15] *FIPA Modeling Area: Deployment and Mobility*, FIPA Modeling TC, 2003.
- [16] J. Lind, "Specifying Agent Interaction Protocols with Standard UML", *In 2nd International Workshop on Agent-Oriented Software Engineering AOSE'02*, LNCS 2222 Springer, 2002, pp. 136-147.
- [17] R. Milner, *Communicating and Mobile Systems: the  $\pi$ -calculus*, Cambridge University Press, 1999.
- [18] A.S. Evans, R.B. France, K.C. Lano, B. Rumpe, "The UML as a formal modelling notation", *In The Unified Modeling Language, UML'98*, LNCS 1618 Springer, 1999, pp. 336-348.

# Appendix 1: Synchronized Move of Mobile Agents



## Appendix 2: An Auction System Case Study in Our Notation

