A mobile-agent platform and a game application specifications using M-UML

Kassem Saleh
Christo El Morr
Aref Mourtada and
Yahya Morad

1 Introduction

A mobile agent is a software component that is able to move across computer systems at which it can execute the code it carries (Cockayne and Zyda, 1988). Typical applications of mobile agents include electronic commerce (Kiniry and Zimmerman, 1997; Chavez and Maes, 1996), network management, and collaborative processing applications among others. Furthermore, a family of computer games can be suitably carried out using a combination of mobile and stationary agents, reducing network traffic by allowing the least number of interactions across platforms.

Mobile agents are created by a distributed application at a computer site and launched to another site using an underlying mobile-agent platform (MAP). An instance of the platform running at the remote site can receive the mobile agent and dispatch it to the distributed application running at that site. There are already some existing platforms like Aglets, Concordia, and Odyssey (Lange and Oshima, 1998). However, none of these platforms take care of two important aspects needed in mobile applications, that is, security and fault-tolerance, in addition, these platforms are not interoperable.

In this work, we report on our experience in: first, developing a prototype platform for mobile agents, and second, developing a game application that uses the developed platform. Our platform considers the guidelines described by the Foundation for Intelligent Physical Agents (FIPA) that allows the interoperability between agents running on different agent platforms[1]. Our platform specifications provide the application developer with an application programming interface (API) that allows the launching of mobile agents with two optional features, security and fault-tolerance. Using our API, new games can then be efficiently developed and deployed on the Internet.

The rest of the paper is organized as follows. Section 2 describes the functionality of the
MAP in terms of its interfaces with applications programs and other platforms. Section 3 describes a mobile game application that uses the mobile platform. This application is specified using Mobile Unified Modeling Language (M-UML). Finally, we conclude the paper and discuss some future extensions to this work.

2 M-UML

In this section, we describe M-UML (Saleh and El Morr, 2001, 2003) by going through each of the UML diagrams and explaining the modifications needed to describe the mobility aspects of the example voting system to model. Figure 1 summarizes the mobility extensions proposed in this section.

2.1 Use case diagram (UCD)

A UCD encapsulates actors, use cases and their relationships. Actors are external entities interacting with the system to model through use cases. An actor can be a software agent interacting with the system. A mobile actor is a mobile software agent interacting with the system while at a platform away from the one at which the agent was created (i.e. the base platform). The functionality of a system or subsystem to model is described by a group of use cases. A use case represents a functional requirement that may involve one or more interacting actors. A mobile use case represents a functional requirement that involves mobile actors. Therefore, at least one mobile actor is involved in a mobile use case. A mobile system contains both mobile and non-mobile use cases. A mobile case or actor will be presented with a box (m) in upper left corner.

2.2 Sequence diagram (SQD)

A SQD is one kind of interaction diagram that shows the interactions among agents while time progresses. Time is represented by a vertical timeline for each agent. An interaction is represented by an arrow starting from the initiating agent directed to the receiving agent. If the initiating agent is interacting while not at its base platform, the timeline from the time this interaction occurs to the time the agent returns to its platform will be filled with “M”. An interaction emanating from that agent’s timeline is called a mobile interaction. A mobile interaction that implies two agents located on the same platform is labeled with the stereotype <<localized>>. A dashed line on the sequence diagram is used to trace the source platform of a mobile agent initiating either a localized or remote interaction. This means that mobile interactions need not be co-located (i.e. labeled
with <<localized>>). For example, a mobile agent may move to a remote platform and starts to interact with other remote agents. However, a mobile interaction that implies a mobile agent and the two interacting agents are not located on the same platform will hold a box (R) at the intersection with the timeline of the initiating agent’s platform. Finally, the return of the agent to its base is represented by a directed arrow stereotyped with <<agentreturn>>.

2.3 Statechart diagram (STD)
A STD is a UML diagram that consists of states and transitions linking pairs of states. Normally, an STD describes the behavior of an actor in terms of state change. An actor or object can be in a particular state at a given time. A mobile state is a state reachable while the actor or object is away from its base platform. A unidirectional link emanating from one state, the source state, and reaching another state, the sink state, is called a transition. A mobile transition is a transition linking two states that are reachable by an agent while at two different platforms. Therefore, we may have a non-mobile transition linking two mobile states, a mobile transition linking a mobile state to a non-mobile state or vice versa, and a mobile transition linking two mobile states. Mobile transitions carry a box (M). However, if an agent can reach a state while it is either at its base or away, that state will carry a dashed box (M). Finally, a transition is labeled with the stereotype <<agentreturn>> if it corresponds to the return of the agent to its base while changing its state.

2.4 Collaboration diagram (COD)
A COD is one kind of interaction diagrams that shows the behavior of interacting objects using a sequence of exchanged messages. The static relationships among objects are well defined. Therefore, a COD shows both structural and behavioral properties of interactions among objects. A COD contains objects (or class instances) and lines connecting pairs of interacting objects or agents. Each interaction is labeled by the message sequence number, the message identifier, and an arrow indicating the direction of the message. Functionally, both collaboration and sequence diagrams are equivalent. Consequently, given a sequence diagram, we can derive the equivalent collaboration diagram and vice versa. If a mobile object participates in the interaction, a box (M) will be added to the object. If the interaction is occurring at the same platform, the interaction line will carry the stereotype <<localized>>. However, if the interaction involves two remote agents while one or both are away from their base platform, the interaction line will carry the box (R). A self-loop interaction line carrying the stereotype <<agentreturn>> at a mobile object represents the return of the mobile object to its originating platform in the particular order given by the sequence number attached also to the interaction.

2.5 Activity diagram (ACD)
An ACD shows the control flow between actions or complex actions called activities. It is basically a blend of finite state machines/statecharts and petri nets syntax and semantics. However, unlike a statechart, the states representing the activities in an ACD may normally span more than one object/agent. An ACD is able to model both the sequential and parallel flow among actions and activities involved in the modeling of a use case. Moreover, an action is a basic atomic computational step, unlike an activity which can be further described by another activity diagram hence creating a hierarchy of ACDs. Swimlanes can also be used to show the agent involved in performing the action or activity. A mobile action or activity involves at least one mobile agent executing at a different platform than its originating platform. A mobile use case involves one or more mobile actions or activities. If the action/activity is executed by an agent outside its originating platform, then this mobile action/activity is represented by an oval with a box (M) attached to its top left corner. However, if this mobile action/activity interacts with a remote agent, it will carry the box (R) instead. The return of an agent to its originating platform is represented by a mobile action stereotyped with <<agentreturn>>. In summary, the diagram will show, in addition to the sequencing of actions/activities, the relative placement of the agents while performing those actions/activities.
2.6 Class diagram (CLD)
A CLD shows the static structure of the system's software classes and describes all relationships among those classes, including the association, aggregation and generalization relationships. A mobile class is a class from which mobile objects/agents are created. A class inheriting from a mobile class inherits its mobility feature. Classes that are parts of another mobile class (in aggregation relationship) are not necessarily mobile classes. Similarly, classes associated with a mobile class are not necessarily mobile classes themselves. Classes from which mobile objects are instantiated are shown with a box (M) at the top left. However, other classes that contain behavior (methods) that is affected (communicating with) by mobile objects are shown with a dashed box (M). Moreover, a class containing behavior affected by a remote mobile agent is shown with a dashed box (R). Therefore, methods invoked by a mobile agent are labeled with either (M) or (R), depending on the location of the calling mobile agent. This is by default when displaying the complete class hierarchy, showing non-mobile classes associated with mobile classes. If a class contains methods of both types, it will carry both dashed boxes (M) and (R).

2.7 Object diagram (OBD)
An OBD shows objects and links among them. An OBD is derived from a class diagram and is based on the static structure of the model, but also shows multiple instances of some objects as illustrations or scenarios of the association relationships. Objects made from a mobile class are mobile objects. Those mobile objects are considered the mobile agents in the system.

2.8 Component diagram (COD)
A COD contains components, interfaces and various types of relationships like dependency, generalization, and association relationships. Components can be executable code, Web pages, documentations and library components. An executable component or a package containing mobile agent code carries a box (M), and an executable component containing an agent that interacts with a mobile agent at the same platform, and not mobile itself, carries a dashed box (M). However, an executable component containing an agent that remotely interacts with a mobile agent, and not mobile itself, carries a dashed box (R). Similarly, any other component related to mobile agents in the system will carry the dashed box (M). Finally, similar to a class, a component may carry more than one type of boxes.

2.9 Deployment diagram (DPD)
A DPD is useful for modeling distributed and client/server systems. The diagram shows the distribution of physical processing units or nodes and components residing on them. A mobile system includes one or more platforms or nodes at which mobile agents are created and dispatched. Such nodes will carry a box (M) at its top left corner. A node that receives a mobile agent will carry a dashed box (M) at its top left corner. Finally, a node that interacts remotely with a mobile agent will carry the dashed box (R) at its top left corner. A node may carry a combination of boxes, for example, a node may have dispatched a mobile agent (M), and can still receive other mobile agents (dashed (M)), or interact with a mobile agent at another node (dashed (R)).

3 Mobile platform specification
This section describes the MAP, its submodules, and its upper and lower interfaces. Two types of applications will be executing on the top of the MAP layer, namely, user applications like games, and management applications. Mobile agent applications (MAA) are responsible for requesting the creation and managing the participation of mobile agents involved in the mobile application. Platform management applications (PMA) are programs responsible for the overall supervision control and administration of the mobile platform. The MAP layer consists of several modules: the agent management system (AMS), the agent registry, the agents repository, the security module and the fault tolerance module. This layer design conforms to the design recommended by the FIPA platform specification. Figure 2 shows the MAP layer and its lower and upper interfaces, in addition to the intra-layer interfaces. In this paper, we
refer to MAP’s upper layer interface as the API. Figure 3 shows the structuring of the AMS within the MAP. The functionalities of each of the MAP interfaces are described and their input and output specifications are listed.

3.1 MAP/MAA interfaces
The MAP/MAA interface offers functionalities related to the creation, launching and acknowledgement of mobile agents:

- **Agent creation.** The MAA uses createAgent to request from the MAP the creation of a mobile agent which will encapsulate the software agent’s code and data. This request specifies: the object to encapsulate, the security and fault-tolerance levels required. The security level can be one of the following literal values: 
  NO_SECURITY, ENCRYPTION, DIGITAL_SIGNATURE, and ENCRYPTION_SIGNATURE. The fault-tolerance level specifies the degree of reliability and fault-tolerance during agent migrations. This level can take one of the following literal values: ACTIVATED, and DE_ACTIVATED. The MAP will return to the MAA agent identification (AID), the created AID as registered in the MAP.

- **Application agent launching.** The MAA uses a Launch to request from the MAP the launching of a mobile agent that was already created. It passes the AID along the request. The MAP will return an indication of success or failure of the launch.

- **Agent acknowledgment.** The MAP uses a Acknowledge to inform the MAA about the success or failure of the creation or termination of application agents. It is the only communication interface between agents and the application. It passes the AID along the request.

3.2 MAP/PMA interfaces
The MAP/PMA interface offers functionalities related to the management of agents, such as the suspension, resumption, termination, and callback of mobile agents:

- **Agent suspension.** The PMA uses agentSuspend to temporarily suspend the execution of a mobile agent. It passes along the AID and the time to be suspended. After which the agent will be automatically resumed, however if the time parameter is zero, the agent will be suspended until it is explicitly resumed by the PMA.

- **Agent resumption.** The PMA uses agentResume to resume the execution of a suspended mobile agent. It passes the AID along the request.

- **Agent termination.** The PMA uses agentKill to terminate a mobile agent. It passes the AID along the request.

- **Agent status.** The PMA uses the getAgentStatus to check the mobile agent status, moves and current position. It passes the AID along the request. The function will access the database and get the registered moves of the mobile agent and use the getFlagPosition to locate the
mobile agent and return a vector with the retrieved data.

3.3 MAP/MAP interfaces
The MAP/MAP (platform to platform) interface offers functionalities related to inter-
platform operations that either performed synchronously (periodically) or asynchronously
triggered by an application at the MAP upper interfaces (i.e. from the PMA or MAA):

- **Platform pinging.** The PMA periodically
  initiates the pinging operation to check the
  availability of a platform, whether it is up or
down. This is important in the fault
tolerance and the optimization processes. It
passes the map of destinations along the
request. The return type is a vector
containing Boolean values specifying the
availability of platforms.

The following are asynchronous operations
triggered by MAA or PMA:

- **Agent rollback.** The MAP uses the rollback
  in order to call the agent back to a specific
  platform. It passes along the AID and the
  platform address that agent has to return to.

- **Agent return.** The MAP uses the return to
  return the mobile agent to the platform it
came from. It passes the mobile agent
  object to be returned along the request.

- **Platform agent launching.** The MAP uses the
  p_launch to launch the mobile agent. It
  passes the agent object to be sent along the
  request.

3.4 Intra-MAP interfaces
The intra-MAP interfaces offer supporting
functionalities related to the upper and lower
MAP interfaces:

- **Map building.** The MAP uses the buildMap
to build the map of destination platforms. It
passes the AID that the map will be built for,
and the minimum and maximum
number of addresses in the map along the
request, if needed.

- **Launched agent registration.** The MAP uses
  the AMSRegisterAgent to register a new
  mobile agent in the AMS database of
  created agents, to list the agent in the log
  file and to store a copy of the agent in the
  Agent Repository. It passes the AID to be
  registered along the request.

- **Launched agent deregistration.** The MAP
  uses the AMSderegisterAgent to remove a
  mobile agent from the AMS agent
database, list it as finished agent in the log
  file, and remove it from the Agent
  Repository. It passes the AID to be
deregistered along the request.

- **Visiting agent registration.** The MAP uses the
  DRegAgent to register a visiting
  mobile agent in the registry database and to
  list the agent in the log file. If the mobile
  agent supports fault-tolerance, a copy will
  be stored in the Agent Repository. It passes
  the AID to be registered along the request.

- **Visiting agent deregistration.** The MAP uses the
  DFdeRegisterAgent to remove a mobile
  agent from the Registry agent database, list
  it as finished agent in the log file, and
  remove it from the Agent Repository if it
  exists. It passes the AID to be deregistered
  along the request.

- **Agent modification.** The MAP uses the
  modifyAgent to modify the attribute(s) of
  an agent. It passes the AID to be modified
  along the request. An agent object,
  containing modifications, will be passed, as
  well, to overwrite the specified mobile
  agent.

- **Searching for an agent.** The MAP uses the
  searchAgent to search for a specific agent
  within the platform, in the AMS and
  Registry database. It passes the AID to be
  searched for along the request. The agent
  object with the same AID will be returned if
  found. Otherwise, null will be returned.

- **Searching for agents.** The MAP uses the
  searchApplicationAgents to search for
  agents servicing a specific application, in
  the AMS and Registry database. It passes
  the application’s type, along the request. A
  list containing the agents’ objects will be
  returned.

- **Agent description.** The MAP uses the
  getAID to get the description of the
  service provided by the mobile agent, if
  registered in the registry. It passes the AID
  along the request. This function can be
  used by the PMA as well.

- **Agent encryption.** The MAP uses the encrypt
to apply encryption to the application agent
object. It passes the application object to be encrypted along the request. This method is important to maintain a required security level.

- **Agent decryption.** The MAP uses the decrypt to decrypt the application object captured from the mobile agent. It passes the encrypted application object along the request. This method is important to support a required security level.

- **Binding digital signature to agent.** The MAP uses the bindSig to bind a digital signature to the mobile agent. It passes the AID along the request. This method is important to maintain and provide added security for agent authentication.

- **Checking digital signature attached to agent.** The MAP uses the checkSig to check a digital signature attached to a mobile agent. It passes the AID along the request. This method is important to maintain and support added security.

- **Agent location update.** The MAP uses the getFlagPosition to provide the PMA with updates of the agent’s last platform identification. It passes AID along the request and returns a String containing the address of the platform.

A complete M-UML specification and design of the platform is provided in (Saleh and El-Morr, 2003).

4 Game application

In this section, we first provide a brief informal description of the mobile game application that uses the MAP specified earlier. Then, we specify the mobile game application using M-UML.

4.1 Brief description

This game applies to any card game involving at least four stationary players. A mobile agent plays the role of game coordinator and judge. The game rules are encoded in both mobile and stationary agents. However, the mobile agent enforces the rules and ensures the fairness in terms of managing the round robin based turn taking among players. The game requires human interactions at the stationary agents, but in some games, agents may carry enough intelligence to take their proper decisions in the game. The mobile agent is referred to as the mobile game coordinator (MGC). The players have to register with the game registry, another stationary agent in the system. The MGC initiates a game by first getting a time-prioritized list of registered players from the game registry (GR). The MGC will then visit the first four players to take their agreement to join a game session. Once four players agree to start a session, the MGC starts moving to each of the player’s platform to update them of the current state of the game and to pick their input at that particular state.

4.2 M-UML specifications

In this section, we will describe the basic specifications of the game application using M-UML. Due to the limited space, we will only provide the UCDs, SQDs and STDs.

4.2.1 UCDs

In the following, we show three basic UCDs needed for the initiation, operation and termination of the mobile game. Two types of use cases are used to support the proper initiation of games. First, players have to register themselves with the game registry. Then, the MGC will initiate the registration of a game session. The UCDs: Player registration (Figure 4) and game registration (Figure 5) show the use cases involved in providing this basic functionality. The UCD game operation and termination (Figure 6) is responsible for the

![Player Registration Diagram](image-url)
control and supervision of the mobile game. These diagrams show only the MGC as the only mobile actor in the game model.

4.2.2 SQDs
Figure 7 shows a SQD describing an optimistic scenario in which all the phases of game initiation, operation and termination were successful.

Additional non-functional requirements are provided by the MAP and are to some extent application-independent. In the following, we show four SQDs related to application fault-tolerance. Figures 8 and 9 show the recovery of the MGC while players were joining the game or playing the game, respectively.

Figures 10 and 11 show the case where a player's platform is down and other players accepting or refusing to continue with the game by admitting a replacing player, respectively.

4.2.3 STDs
In the following, we show three STDs in Figures 12, 13, and 14, describing the behavior.
of each of the agents involved in the game, namely, the player, the GR, and the MGC.

4.2.4 CLD

Figure 15 shows the MGC’s class design which includes the overall class relationships like associations, generalization, and aggregation relationships. In addition, the class diagram shows the attributes of each class.

5 Conclusions

In this paper, we have described the specification and architecture of a
MAP for launching and controlling mobile agents created by mobile agent-based software applications. We have then described using M-UML a mobile agent-based game application that takes advantage of the mobile platform.

Various fault-tolerance and security issues are delegated to the platform itself, therefore relieving the application designer from accommodating these features in the game design itself. This approach allows for the easy development and deployment of mobile game applications. In the future, we plan to experiment with different game categories in which more agent mobility is present. This will allow us to examine the robustness of our platform under various game scenarios.
Web site


References


Lange, D. and Oshima, Y. (1998), Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, Reading, MA.


About the authors

Kassem Saleh is Professor in Computer Science in the Department of Computer Science at the American University of Sharjah in the United Arab Emirates. He is a specialist in software engineering, distributed systems and mobile agents. E-mail: ksaleh@aus.ac.ae

Christo El Morr is an Assistant Professor of Management Information Systems in the MIS Department at the American University of Sharjah, United Arab Emirates. He is a specialist in picture archiving and communication systems (PACS) consisting of intra-hospital networks for medical image archiving and communication. His research includes e-health and e-commerce, intelligent agent applications and software engineering. E-mail: celmorr@aus.ac.ae

Aref Mourtada is a senior student in the Computer Science Department at the American University of Sharjah, United Arab Emirates.

Yahya Morad is a senior student in the Computer Science Department at the American University of Sharjah, United Arab Emirates.