International Journal of Software Engineering
and Knowledge Engineering
Vol. 15, No. 4 (2005) 719–746

World Scientific
www.worldscientific.com

A SECURITY BASED MODEL FOR MOBILE AGENT SOFTWARE SYSTEMS

HAIPING XU

Computer and Information Science Department,
University of Massachusetts Dartmouth, North Dartmouth, MA 02747-2300, USA
hxu@umassd.edu

ZHIGUO ZHANG* and SOL M. SHATZ†

Department of Computer Science, University of Illinois at Chicago,
851 S. Morgan Street, Chicago, IL 60607-7053, USA
*zzhang@cs.uic.edu
†shatz@cs.uic.edu*

Received 15 September 2004
Accepted 10 December 2004

Security modeling for agents has been one of the most challenging issues in developing practical mobile agent software systems. In the past, researchers have developed mobile agent systems with emphasis either on protecting mobile agents from malicious hosts or protecting hosts from malicious agents. In this paper, we propose a security based mobile agent system architecture that provides a general solution to protecting both mobile agents and agent hosts in terms of agent communication and agent migration. We present a facilitator agent model that serves as a middleware for secure agent communication and agent migration. The facilitator agent model, as well as the mobile agent model, is based on agent-oriented G-nets — a high level Petri net formalism. To illustrate our formal modeling technique for mobile agent systems, we provide an example of agent migration to show how a design error can be detected.

Keywords: Agent security; mobile agent; facilitator agent; CPV approach; Petri nets; agent-oriented G-nets.

1. Introduction

Software agents can be classified in terms of a space defined by the three dimensions of intelligence, agency and mobility [1]. The first dimension, *intelligence*, is rooted in artificial intelligence research dating to the 1950s, where intelligent agents can be classified according to their capabilities to express preferences, beliefs and emotions, as well as their ability to fulfill a task by reasoning, planning and learning techniques. The second dimension, *agency*, represents the degree of an agent's autonomy and authority, which is measured by the nature of its interaction with the environment. The third dimension, *mobility*, emerged in the 1990s, is motivated

by the rise and rapid growth of a networked computing environment, especially the Internet, and the need for techniques to locally exploit distributed resources. Within this dimension of software agent research, the goal is remote action and mobility of data and computation.

Current research on agent-based systems generally does not exploit all the capabilities classified by these three dimensions. For example, multi-agent systems (MAS) based on distributed artificial intelligence try to execute a given task using a large number of possibly distributed but static agents that collaborate and cooperate in an intelligent manner [2, 3]. On the other hand, research on mobile agents usually emphasizes agent mobility and agent coordination, and mobile agents are typically assumed to only have very limited or even no intelligence [4–6]. The development schema in the latter case is sometimes called a weak agent approach, which contrasts with the strong agent approach that involves artificial intelligence techniques [7].

In this paper, we consider an architecture that is based on two basic components: mobile agents and facilitator agents. We define both by introducing mobility into our previously presented framework for agent-oriented software. This framework has been designed to model intelligent software agents for multi-agent systems, and it supports design reuse by providing an *inheritance* mechanism [8]. The resulting mobile agent models explicitly support asynchronous message passing. A key property of our approach is that fundamental agent models are based on the agent-oriented G-net formalism, a formalism derived from an object-based Petri net model. This paves the way for formal analysis, as seen in earlier work [9]. In the work presented here, we focus on explicit consideration for some security issues encountered in mobile agent systems.

The rest of this paper is organized as follows. In Sec. 2, we describe related work and highlight the relationships to our research. In Sec. 3, we summarize the agent-oriented G-net model, which was first proposed in [8]. In Sec. 4, we propose the architecture for a mobile agent system, and describe how to design the principal agent system components: the mobile agents and the facilitator agents. We incorporate a CPV (Certificate, Passport, and Visa) approach for secure agent communication and agent migration. In Sec. 5, we provide an example of agent migration and show how a design error is detected using formal analysis. Finally, in Sec. 6, we summarize our contributions and discuss the future work.

2. Related Work

Previous work on multi-agent systems has fostered the concept of agent-oriented software [10, 11, 8], where agents are viewed as intelligent software that has the properties of autonomy, reactivity, pro-activeness and sociability. Corresponding agent-oriented design methodologies are also proposed to provide guidelines for agent specification and design. Examples of such work are the AAII methodologies [12] and the Gaia methodologies [11], which are extensions of object-oriented

methodologies. In our own previous work [9, 13], an *inheritance* mechanism, in terms of agent functionalities, is introduced into the development of agent-oriented software.

For mobile agents, the concern is with software agents that can migrate over computer networks. The concept of location has been one of the key features to characterize mobility in most theoretical models of mobile agents, such as the distributed join-calculus [14], which is an extension of the π-calculus that introduces the explicit notions of named localities and distribution failure. Additional typical formalisms for agent mobility modeling are summarized as follows. Mobile UNITY [4] provides a programming notation that captures the notion of mobility and transient interactions among mobile nodes. Inspired by Mobile UNITY, the concept of connectors [15] is explicitly identified to describe different kinds of transient interactions, and facilitate the separation of coordination from computation in mobile computing. The connectors are written in COMMUNITY, a UNITY-like program design language whose semantics is given in a categorical framework. MobiS [5], as an extended version of PoliS, is a specification language based on multiple tuple spaces. It can be used to specify agent coordination and architectures containing mobile components.

Although the above results formally model mobile agents in terms of their mobility, they are not built upon a framework that explicitly supports the intelligence feature of agents. Furthermore, they are weak in agent communication modeling. Typically, such models are reactive rather than pro-active. In other words, these models may simply act in response to their environment, but they are not able to exhibit goal-directed behaviors. Additional efforts, such as the MARS (Mobile Agent Reactive Spaces) project [6], attempt to introduce context-dependent coordination into agent models; however, without explicitly suggesting the communication mechanism among mobile agents. There are also some research efforts concerned with mobile agent communication mechanisms; however, they are not formally defined [16, 17].

Another drawback of the above formal modeling approaches is that they restrict their scope of applicability due to a lack of security measures. There is some previous work on solving security problems in mobile agent systems. Such problems include how to protect mobile agents from malicious hosts and how to protect hosts from malicious agents, as presented by Sander and Tschudin [18]. The security threats that an agent platform faces from a malicious agent have been discussed in a number of papers [19, 20, 21]. Farmer and his colleagues proposed a system architecture to model the trust relations between the principals of mobile agents systems. A unique aspect of the architecture is a *state appraisal* mechanism that protects hosts from attacks via state modification [19]. Gray and his colleagues addressed how to protect an individual machine and how to protect a group of machines in the context of D'Agents, a mobile agent system whose agents can be written in TCL, Java and Scheme [20]. Vuong and Fu proposed a security based

architecture and implemented a security system based on a novel mobile intelligent system, called Actigen [21]. They first proposed a *passport-visa* approach to simulate the activities of traveling abroad in real life. On the other hand, a malicious host might steal private information from a mobile agent, or modify the agent to compute the wrong result or to misbehave when it jumps to another site. Sander and Tschudin addressed this problem by identifying a special class of functions — polynomials and rational functions — together with encryption schemes that lead to a non-trivial example of cryptographically hiding a function such that it must be executed with an interactive protocol [18]. Based on Sander and Tschudin's work, Lee and his colleagues proposed an extension of mobile cryptography that provides a practical idea for implementing mobile cryptography [22].

From the above review, we can see that current work on mobile agents mostly emphasizes some particular features of the mobile agents, e.g., agent mobility or agent security. With the continuing improvement of agent technology, and the rapid growth of software system complexity, especially for Internet applications, there is a pressing need for a more general model of mobile agents, in which agents are not only mobile, cooperative and intelligent, but also supports secure agent communication and agent migration. There is some previous work that discusses intelligent mobile agents [23]; however, it does not consider a formal framework for intelligent mobile agent design. One notable effort that emphasizes a formal framework for mobile agents is the work of Xu *et al.* [24]. While this work considers the cooperation between mobile agents for the purposes of migration, it did not explicitly address security issues. We seek to incorporate the security issues into our modeling framework and address the following types of properties: when a remote host refuses a migration request from a remote mobile agent, the mobile agent should not be allowed to migrate. Our proposed security based model for mobile agent systems not only addresses the problem of protecting the host from malicious agents, but also protecting the agent from malicious hosts. A mobile agent's migration request is granted only if it passes the security checking by a remote facilitator agent; meanwhile, a mobile agent only migrates to a trusted host with a certified facilitator agent.

3. A Base Framework for Agent-Oriented Software

One of the most important software engineering principles is that a system should be composed of a set of independent modules, where each module hides the internal details of its processing activities and modules communicate through well-defined interfaces. The G-net model provides strong support for this principle [25, 26]. G-nets, first proposed by Deng and his colleagues [25], are an object-based extension of Petri nets [27]. Petri nets are a graphically defined model for concurrent systems, having the advantage of being visually appealing, while also being theoretically mature and supported by robust tools. Details about G-net models can be found in Refs. 25 and 26.

Although the G-net model works well in object-based design, it is not sufficient in agent-based design for the following reasons. First, agents that form a multi-agent system may be developed independently by different vendors, and those agents may be widely distributed across large-scale networks such as the Internet. To make it possible for those agents to communicate with each other, it is desirable for them to have a common communication language and to follow common protocols. However, the G-net model does not directly support protocol-based language communication between agents. Second, the underlying agent communication model is usually asynchronous, and an agent may decide whether to perform actions requested by some other agents. The G-net model does not directly support asynchronous message passing and decision-making; it only supports synchronous method invocations in the form of *ISP* (Instantiated Switch Place) mechanisms [25]. Third, agents are usually designed to determine their behavior based on individual goals, their knowledge and the environment. They may autonomously and spontaneously initiate internal or external behavior at any time. The G-net model can only directly support a predefined flow of control.

To support agent-oriented design, we need to extend a G-net to support modeling an agent class.[a] This extension is made in three steps. First, we introduce five special modules to a G-net to make an agent autonomous and internally motivated. As shown in Fig. 1, the five special modules are the *Goal* module, the *Plan* module, the *Knowledge* module, the *Environment* module and the *Planner* module. The *Goal*, *Plan* and *Knowledge* modules are based on the BDI agent model proposed by Kinny and his colleagues [12]. The *Goal* module consists of a goal set that specifies the goal domain and goal state. The *Plan* module consists of a set of plans that are associated with a goal or a subgoal. Each goal or subgoal may associate with more than one plan, and the most suitable one will be selected to achieve that goal or subgoal. A *Knowledge* module describes the information about the agent's internal state, its environment, and interaction protocols. The *Environment* module is an abstract model of the environment, i.e., the model of the outside world of an agent. The *Planner* module represents the heart of an agent that may decide to ignore an incoming message, to start a new conversation, or to continue with the current conversation. In the *Planner* module, committed plans are achieved, and the *Goal*, *Plan* and *Knowledge* modules of an agent are updated after the execution of each communicative act that defines the type and content of a message [28, 29], or if the environment changes. Second, different from the semantic of a G-net as an object or a module, we view the extended G-net, we call it an *agent-oriented G-net*, as a class model, i.e., the abstract of a set of similar agents. Third, we define the instantiation of the agent-oriented G-net as follows: when an agent-oriented G-net A is instantiated, we generate an agent identifier $A.Aid$ for the resulting agent object AO; meanwhile, the state of AO, i.e., any state variables defined in A, is initialized.

[a]We view the abstract of a set of similar agents as an agent class, and we call an instance of an agent class an agent or an agent object.

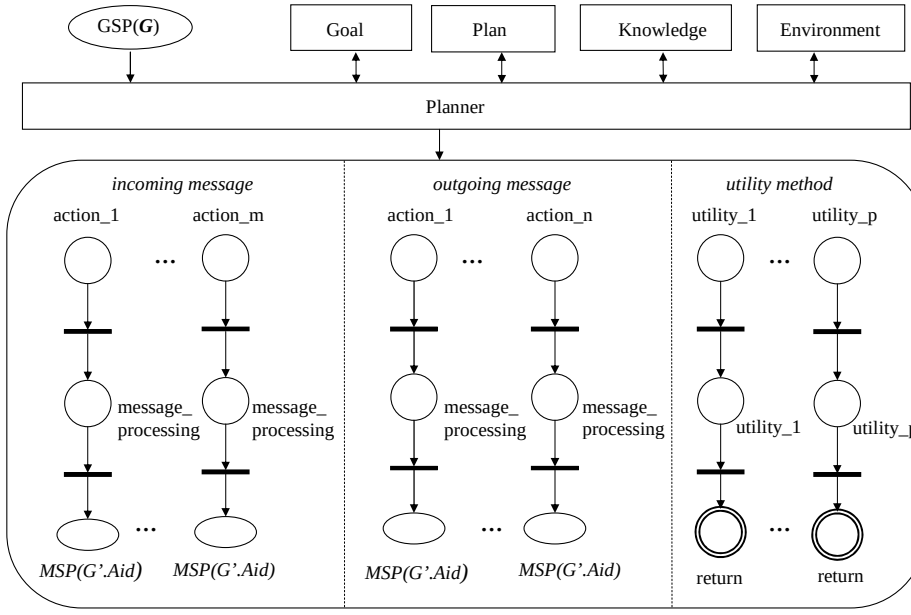


Fig. 1. A generic agent-oriented G-net model (adapted from [9]).

The *internal structure* (IS) of an agent-oriented G-net consists of three sections: *incoming message*, *outgoing message* and *utility method*. The *incoming/outgoing* message section defines a set of *message processing units* (MPU), which correspond to a subset of communicative acts. Each MPU, labeled as *action_i* in Fig. 1 is used to process incoming/outgoing messages and execute any necessary actions before or after the message processing. The *utility method* section defines a set of methods that can only be called by the agent itself.

Although both objects (passive objects) and agents use message-passing to communicate, message-passing for objects is a unique form of method invocation, while agents distinguish different types of messages and model these messages frequently as speech-acts and use complex protocols to negotiate [11]. In particular, these messages must satisfy the format of the standardized communicative (speech) acts, e.g., the format of communicative acts that defined in the FIPA agent communication language [30], or KQML [28]. Note that in Fig. 1 each named *MPU action_i* refers to a communicative act, thus our agent-oriented model supports an agent communication interface. In addition, agents analyze these messages and can decide whether to execute the requested action. As we stated before, agent communications are typically based on asynchronous message passing. Since asynchronous message passing is more fundamental than synchronous message passing, it is useful for us to introduce a new mechanism, called *message-passing switch place* (MSP), to directly support asynchronous message passing. When a token reaches an MSP (represented

as an ellipsis in Fig. 1), the token is removed and deposited into the *GSP* of the called agent. But, unlike with the G-net *ISP* mechanism, which invokes a method defined in the *utility method* section, and returns the result synchronously, the calling agent does not wait for the token to return before it can continue to execute its next step. Since methods defined in the *utility method* section can only be called by the agent itself, agent communications must take place asynchronously through the *MSP* mechanisms.

A generic template of the *Planner* module is shown in [9]. The function of the *Planner* module is to receive messages from the *GSP* place, make decisions and dispatch those messages to different *MPUs* according to the information contained in the *Goal*, *Plan*, *Knowledge* and *Environment* modules. In Sec. 4, we present and discuss a revised version of this component, which does not use inheritance but does provide support for secure mobility.

4. A Mobile Agent System Design

Today's users demand ubiquitous network access independent of their physical location. This style of computation, often referred to as *mobile computing*, is enabled by rapid advances in wireless communication technology [31]. The networking scenarios enabled by mobile computing range roughly between two extremes. At one end, the availability of a fixed network is assumed, and its facilities are exploited by the mobile infrastructure. We call this form of mobility *logical mobility*. At the other end, the fixed network is absent and all network facilities (e.g., routing) must be implemented by relying only on the available mobile hosts, namely *ad hoc* networks. This form of mobility is called *physical mobility*. Mobile agent technology is a new networking technology that deals with both forms of mobility. It offers a new computing paradigm in which a program, in the form of an intelligent software agent, can suspend its execution on a host computer, transfer itself to another agent-enabled host on the network, and resume execution on the new host. Here, as we will see in the next section, we define a host as either a stationary host or a mobile host that is situated in an *ad hoc* network.

4.1. *Mobile agent system architecture*

We propose an *agent world* (AW) architecture that provides the platform for execution and migration of mobile agents. A few key definitions for this architecture are now given as follows.

Definition 4.1. *Stationary Agent* (SA) and *Mobile Agent* (MA)
An agent A is a 3-tuple ($HOMEIP$, $CURIP$, AO), where $HOMEIP$ is the IP address of the host, on which agent A is created; $CURIP$ is the current IP address of the host supporting agent A; and AO is the agent object, with the general structure described in Sec. 3. If at all time, $CURIP = HOMEIP$, we refer to agent A as a *stationary agent* (SA); otherwise, we refer to agent A as a *mobile agent* (MA).

Definition 4.2. *Agent Virtual Machine* (*AVM*)
An *agent virtual machine* (*AVM*) Θ is a 4-tuple (*IFA, SMA, HOSTIP, ID*), where *IFA* is a stationary intelligent facilitator agent on Θ, which is responsible for recording information and providing services for mobile agents running on Θ; *SMA* is a set of mobile agents running on Θ; *HOSTIP* is the current IP address of the host that is supporting Θ; and *ID* is Θ's unique identifier.

Definition 4.3. *Stationary Host* (*SH*) and *Mobile Host* (*MH*)
A host Π is a 4-tuple (*SAVM, ACOM, HOMEIP, CURIP*), where *SAVM* is a set of *agent virtual machines* (*AVM*); *ACOM* is the communication protocol among AVMs in *SAVM*, and examples of such protocols are IPC and TCP/IP; *HOMEIP* is the original IP address of the host; and *CURIP* is the current IP address of the host. If at all times, *CURIP* = *HOMEIP*, we call the host Π a *stationary host* (*SH*); otherwise, we call it a *mobile host* (*MH*).

Definition 4.4. *Agent World* (*AW*)
An *agent world* (*AW*) is a 3-tuple (*WKFA*, *SHOST*, *HCOM*), where *WKFA* is a well-known stationary facilitator agent, which is responsible for recording the most recent addresses of all hosts in the agent world. *WKFA* is also responsible for recording the public keys of all facilitator agents and for issuing certificates to the facilitator agents in the agent world (we will introduce the concepts of public key cryptograph and certificate in Sec. 4.3). *SHOST* is a set of hosts in the agent world that can provide the services of an agent virtual machine and *HCOM* is the communication protocol among hosts in *SHOST*; an example of such protocols is TCP/IP.

Figure 2 shows a generic mobile agent system, and an example of agent migration. In the figure, Host-A and Host-B are two machines connected by a network. To make mobile agent platform independent of machine hardware, a mobile agent runs on an *agent virtual machine* (AVM), which provides a protected agent execution environment on a host. Each host may have a number of AVMs; however, to simplify matters, we only illustrate one AVM on each host in Fig. 2. Each AVM is responsible for hosting and executing any agents created on that AVM or that arrive over the network, and for providing APIs for agent programmers. We will discuss the sequence of steps for agent migration (illustrated by the directed arcs in Fig. 2) shortly.

Since in this paper we view mobile agents and facilitator agents as intelligent software agents, for the rest of our discussion a mobile agent or a facilitator agent always refers to an intelligent mobile agent (IMA) or an intelligent facilitator agent (IFA), respectively. In Fig. 2, when a mobile agent α on AVM Θ_{A} wants to migrate to another AVM Θ_B, it needs to contact the remote facilitator agent θ_{B} first, which resides on AVM Θ_{B} (step 1). In fact, the mobile agent α needs to know the address of the remote facilitator agent θ_{B} before the communication can begin. This could be done by querying the needed information from its local facilitator agent θ_{A},

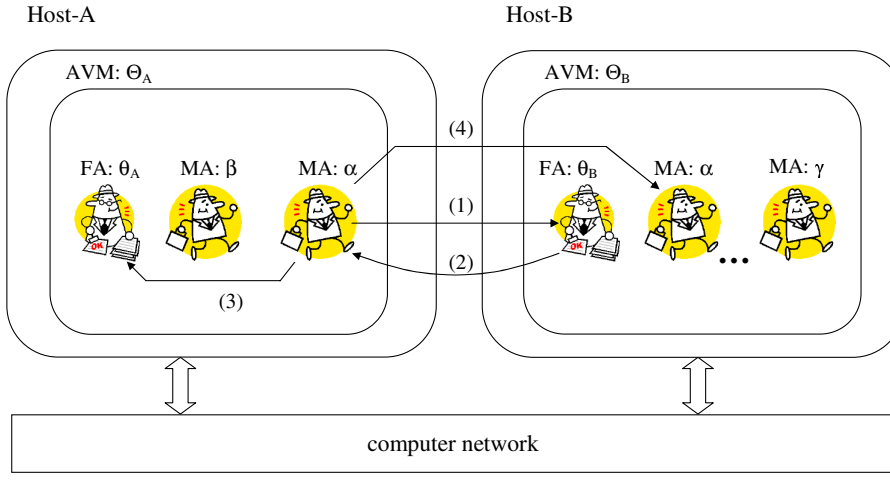


Fig. 2. Agent world architecture and an example of agent migration.

which resides on AVM Θ_A. If the local facilitator agent θ_A knows the address of the remote facilitator agent θ_B, it will provide such information to the mobile agent α; otherwise, it will contact the well-known facilitator agent *WKFA* (we do not show it in Fig. 2) for this information and forward the results to the mobile agent α thereafter. For simplicity, this procedure is omitted in Fig. 2. Based on security and resource criteria, the remote facilitator agent θ_B decides if the migration request is granted. If the migration request is granted (step 2), the mobile agent α notifies its local facilitator agent θ_A about its departure (step 3), and finally α moves to the remote AVM Θ_B (step 4).

The situation above is an example of logical mobility. For physical mobility, a host may, at some time, change its IP address or lose its IP address temporarily (detached from the network). In this case, the well-known facilitator agent *WKFA* is critical for recording this information. To successfully send a message to an agent, whose hosting of AVM has changed its *HOSTIP* address, the knowledge of the sender agent's local facilitator agent needs to be consistent with the latest network information. Further discussion about this issue is beyond the scope of this paper, which concentrates on logical mobility.

4.2. *Security considerations and design of IMA and IFA*

The scenario introduced in Sec. 4.1 seems practical for agent communication and agent migration; however, if we allow a mobile agent to communicate directly with a remote facilitator agent or with other mobile agents, mobile agents would then be responsible for checking authentications of all other mobile agents and all facilitator agents in the agent world. Meanwhile, a facilitator agent would be responsible for

checking authentications of any other facilitator agents and all mobile agents in the agent world. So, this approach would require agents to record all other agents' authentication information. This not only results in information redundancy, but also makes agent communication inefficient and unreliable. In order to reduce the information recorded by every agent, we consider each facilitator agent (FA) as a type of middleware for agent communication and agent migration, and a mobile agent (MA) on an AVM can only communicate directly with its local FA. When a mobile agent wants to communicate with a remote FA, it first sends a message to its local FA, and the local FA forwards the message to the remote FA. Similarly, when the remote FA replies to the mobile agent, the message is also forwarded by the local FA. In addition, any communications between two mobile agents (local or remote) also take place through local or remote FAs. Finally, when a mobile agent's migration request is granted by a remote FA, the mobile agent is serialized by its local FA and sent to the remote FA, which is responsible for resuming the execution of the mobile agent. Under the above scenarios, the authentications are limited to be used between local mobile agents and local facilitator agents, as well as between local facilitator agents and remote facilitator agents. In other words, all communications between mobile agents (local or remote), and all communications between a local mobile agent and a remote facilitator agent, must bridge through local FAs. Thus, a mobile agent is only responsible for checking authentication of its local FA; while a FA is only responsible for checking the authentications of its local mobile agents and any other remote FAs in the agent world.

Based on the above approach, the agent interaction protocol between mobile agents and local/remote agents can be described as in Fig. 3.

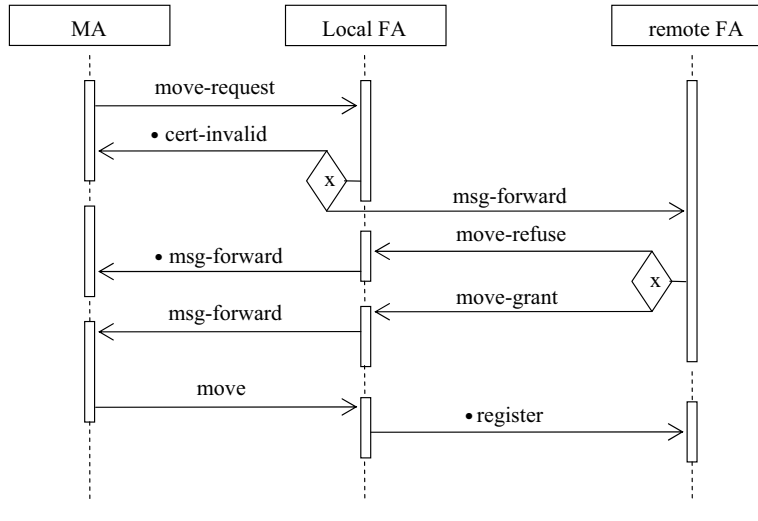


Fig. 3. Interaction protocol between a mobile agent (MA) and facilitator agents (FA).

As shown in Fig. 3, when a mobile agent α wants to migrate from its local AVM Θ_{A} to a remote AVM Θ_{B}, it first sends a `move-request` message to its local FA. The local FA checks the mobile agent's certificate as well as the certificate of its issuer to see if it is valid. If the certificate is not valid, then the local FA denies the request by sending a `cert-invalid` message back to the mobile agent. Notice that in Fig. 3 the end of protocol operation "•", put in front of the message name `cert-invalid`, marks that this message ends the conversation. If the certificate is valid, the `move-request` message will be forwarded to the remote FA on Θ_{B}. The remote FA checks both the local FA and the mobile agent's certificates. If there is anything suspicious, the `move-request` is denied, and a `move-refuse` message will be forwarded to the mobile agent via the local FA. Otherwise, the `move-request` is granted. In the same way, a `move-grant` message will be forwarded to the mobile agent via the local FA. Upon receiving the `move-grant` message, the mobile agent sends a `move` message to its local FA to confirm the desire to migrate. The local FA then serializes the mobile agent and sends it to the remote FA for registration.

We view each message in the interaction protocol as a communicative act with a format defined in some agent language-notation such as FIPA, or KQML [30, 28, 32]. Based on the communicative acts, we design the agent-oriented G-net model of mobile agents and facilitator agents as in Figs. 4 and 5, respectively. Notice that we define three *MPUs* in the incoming message section of the agent-oriented G-net model of mobile agents, which corresponds to the three communica-

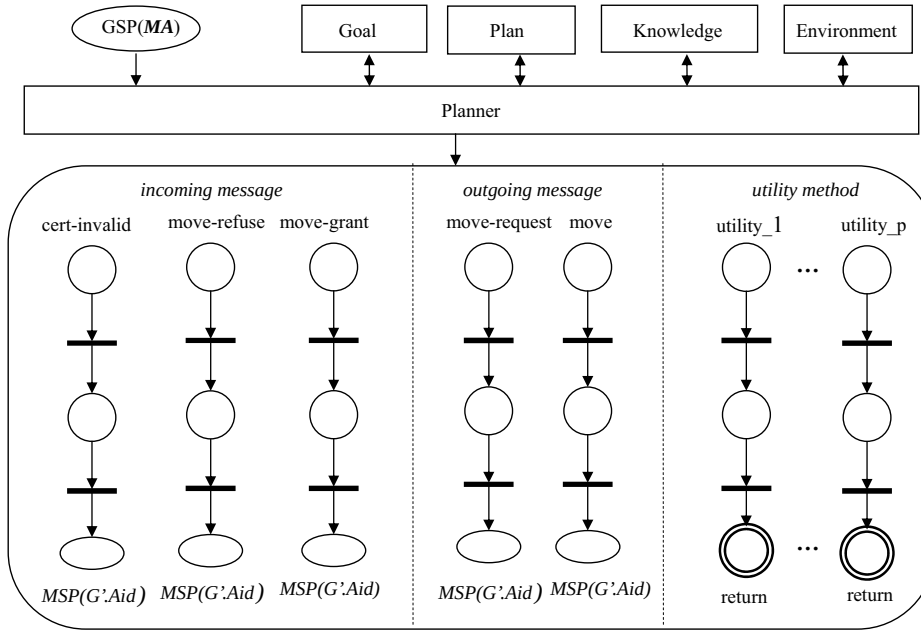


Fig. 4. Agent-oriented G-net model of mobile agent.

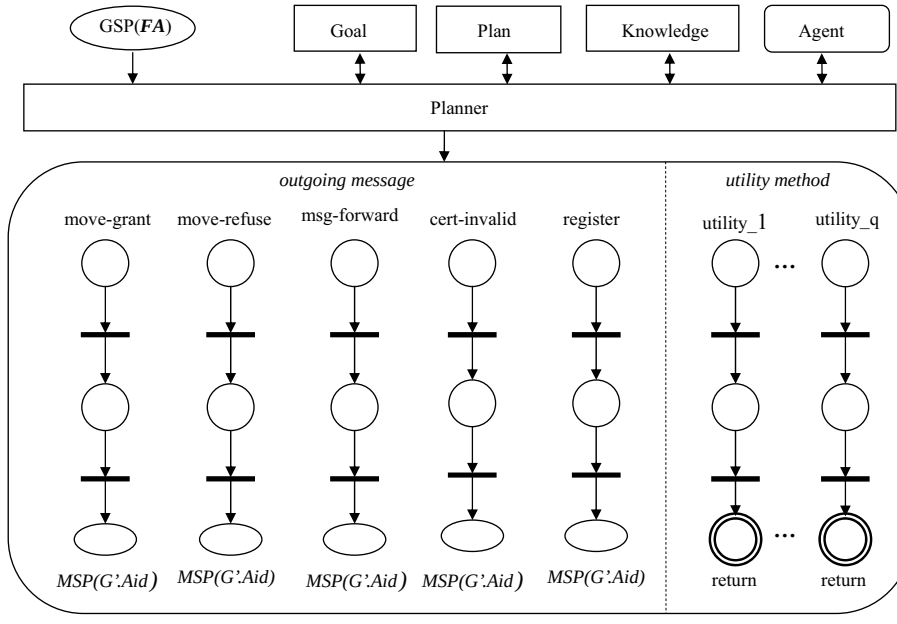


Fig. 5. Agent-oriented G-net model of facilitator agent.

tive acts (`cert-invalid`, `move-grant`, and `move-refuse`) in the interaction protocol. Similarly, we define two *MPUs* that correspond to the communicative acts of `move-request` and `move` in the interaction protocol. Details on how to design agent-oriented G-net models based on interaction protocols are provided in previous work [9]. The planner module of the mobile agent may reuse the general planner module as we defined previously [9], which support inheritance. The agent model for facilitator agents can be designed in a similar way. However, the general planner module is not sufficient for facilitator agents to support agent communication and agent migration, especially when security issues are taken into consideration. Therefore, we need to redesign the planner module for facilitator agents. Notice that the environment module in Fig. 5 becomes a module called *Agent* place (denoted as a rounded rectangle). The *Agent* place is an agent container for local mobile agents including agents that migrate from a remote AVM. With this facility, local mobile agents are under the control of a local facilitator agent. When a migration is granted, the mobile agent can be serialized and sent out to the *Agent* place of a remote facilitator agent when a migration request is granted. Also notice that, since there is only one facilitator agent defined on each agent virtual machine, the inheritance feature is not significant for modeling facilitator agents.

Unlike Fig. 4, Fig. 5 shows no *incoming message* section in the internal structure of the facilitator agent model. This is to simplify our facilitator agent model. As we will see in Sec. 4.4, all incoming messages to a facilitator agent are directly processed by the planner module of the FA.

Before we define the planner module for facilitator agents, we first introduce the cryptographic mechanism for secure agent communication, which is a key mechanism in designing the planner module of the facilitator agent model.

4.3. *The CPV approach*

Public key cryptograph is one of the most widely used encryption mechanisms, which involves a pair of keys — a public key and a private key [33]. The public key of the message receiver is used to encrypt a message, and the encrypted message can only be decrypted by the receiver using its private key. Although a message can be safely sent to a receiver using the receiver's public key, the receiver cannot guarantee if the message is actually sent by the claimed sender. To authenticate the message sender, the message must be digitally signed. The basic idea of digital signature is to use the sender's private key to encrypt a message [34]. If the receiver can recover the message using the sender's public key, then the receiver can be certain that the message is actually sent by the claimed sender because only the message sender can encrypt the message with its private key. Now, one more question arises: how can a message receiver know that the public key actually belongs to the claimed sender? This can be verified by using a certificate, which is a signed document that consists of the sender's public key and its identity. The certificate is signed by a certificate authority (CA), which is a third party, other than the message sender/receiver, that certifies the accuracy of the binding of a public key and an identity [34].

For our security modeling, we incorporate a CPV (Certificate, Passport and Visa) approach, which is based on the *passport-visa* mechanism first proposed by Vuong and Fu [21]. The basic idea of the *passport-visa* mechanism is to simulate the activities of traveling abroad in real life [21, 35]. In this paper, we define a passport as a certificate with additional visa pages, similar to an actual passport that contains visa pages with possible visa stamps. Each visa stamp on a visa page is also defined as a certificate, which is issued by a foreign facilitator agent. Our proposed method initially provides every mobile agent a certificate that is issued and signed by its owner/user, who creates the mobile agent. As a mobile agent requests to move from one AVM to another, the agent's certificate is replaced by a more specialized certificate — a passport, which is signed and issued by a local FA. When the migration request is approved by a remote FA, a visa stamp is put on one of the visa pages in the mobile agent's passport.

To ensure that a facilitator agent is a trusted agency, it must also have a certificate, which is signed by a certificate authority (CA). In our agent world architecture, the CA is the well-known facilitator agent, denoted as *WKFA*, that has a globally recognized public key. When a new AVM with a facilitator agent (FA) wants to enter the agent world, the FA will send its public key, its identity, valid time and other information to the *WKFA*. The *WKFA* will record the new FA's information, and assign a privilege to the new FA. This privilege will be used in the passport/visa issuing procedure. On the other hand, local users/owners of the AVM will assign

certificates to local mobile agents that are created on the local AVM. Because a local user does not have a public key recorded by any remote FA, the certificates issued by local users cannot be checked by a remote FA while communicating with a remote FA. Therefore, whenever a mobile agent has an intention to migrate by sending a *move-request* message to its local FA, the mobile agent's original certificate must be replaced with a passport that is issued by its local FA. The structures of the *Certificate*, *Passport* and *VisaPage* are defined as follows:

```
Struct Certificate {
      int serial_number;      // the serial number of the certificate
      String issuer_name;     // the issuer's name
      String name;            // the name of holder
      Privilege privilege;    // the privilege assigned by the issuer
      String public_key;      // the public key of the holder
      Time valid_time;        // the valid time for the certificate
      Signature signature;    // the encrypted value of the above items
                              // encoded by the issuer's private key
}

Struct Passport {
     Certificate passport;     // issued by a local facilitator agent
     Certificate CTofIssuer;   // issued by WKHOST
     VisaPage visaPages;       // defined as a linked list
}

Struct VisaPage {
     Certificate visaStamp;    // issued by a remote facilitator agent
     Certificate CTofIssuer;   // issued by WKFA
     VisaPage nextVisaPage;    // reference to the next visa page
}
```

For security considerations, any transferred message MSG is first encrypted by the sender's private key into $MSG' = \mathrm{E}(k_{\text{PRIV-S}}, MSG)$, where $k_{\text{PRIV-S}}$ is the private key of the sender. This encoded message is then combined with the sender's certificate (denoted as CT) as $MSG'' = (\mathrm{E}(k_{\text{PRIV-S}}, MSG), CT)$. Finally, the combination of the encoded message and the certificate is further encrypted using the receiver's public key into $E\text{-}MSG = \mathrm{E}(k_{\text{PUB-R}}, (\mathrm{E}(k_{\text{PRIV-S}}, MSG), CT))$, where $k_{\text{PUB-R}}$ is the public key of the receiver. Upon receiving the encrypted message $E\text{-}MSG$, the receiver first uses its private key $k_{\text{PRIV-R}}$ to decode the message into $MSG'' = \mathrm{D}(k_{\text{PRIV-R}}, E\text{-}MSG)$. It then reads the certificate CT of the sender from MSG'', and verifies its validity by checking the signature signed by the issuer. If the CT passes the security checking, the receiver then uses the public key of the message sender $k_{\text{PUB-S}}$ read from the CT to decrypt the message MSG' (extracted from MSG'') into $MSG = \mathrm{D}(k_{\text{PUB-S}}, MSG')$, which becomes the original message MSG. The structure of each message MSG is defined as follows:

```
Struct Message {
      AgentID sa;         // source agent identification
      AgentID da;         // destination agent identification
      Head mh;            // message head
      MessageBody mb;     // message body
}

enum Head {RMI, GOTO, REGISTER, LOCAL, METHOD};

Struct MessageBody {
      String str;         // string message
      File att;           // binary attachment
}
```

The message format consists of four fields: `sa, da, mh` and `mb`. The fields `sa` and `da` are agent identifications, representing the source agent and the destination agent, respectively; `mh` is a message head, representing the message type; and `mb` is the message body, which contains a string `str` and a binary attachment `att`. The string `str` in a message body describes text information of the message; while the binary attachment `att` can be a piece of code or a serialized encoding of an agent object. Both of the fields `str` and `att` can be `null` values. The message head `mh` can be a constant value of `RMI, GOTO, REGISTER, LOCAL` or `METHOD`. An `RMI` message is a message between a local mobile agent and a remote FA, which should be forwarded by a local FA; a `GOTO` message is a message sent by a local mobile agent to a local FA for migration request; a `REGISTER` message is used when a local FA sends a serialized mobile agent object to a remote FA for registration; a `LOCAL` message is a message sent by a local mobile agent to a local FA for any purpose other than migration. Finally, a message with the message head of `METHOD` represents a local method invocation on a local FA. Since all methods defined in a local FA's *method utility* section can only be invoked by the agent itself, a message with a message head of `METHOD` is sent by the facilitator agent using an *ISP* mechanism (representing a method call). Notice that all messages, except a message with a message head of `METHOD`, are transferred *asynchronously*; while transferring a message with a message head of `METHOD` is a *synchronous* message passing.

4.4. *The planner module of facilitator agents*

From the above description, we can see that the agent world architecture is divided into two layers, the facilitator agent (FA) layer and the mobile agent (MA) layer, where FAs serve as a middleware for agent communication and agent migration. We now design the FA's planner template, shown in Fig. 6.

In Fig. 6, the *Agent* place (denoted as a rounded rectangle) represents a container for a set of local mobile agents, including agents migrating from remote hosts. Each token in this place identifies a unique mobile agent. The *Knowledge/Plan/Goal* place represents a simplified combination of places for the *Knowledge, Plan* and

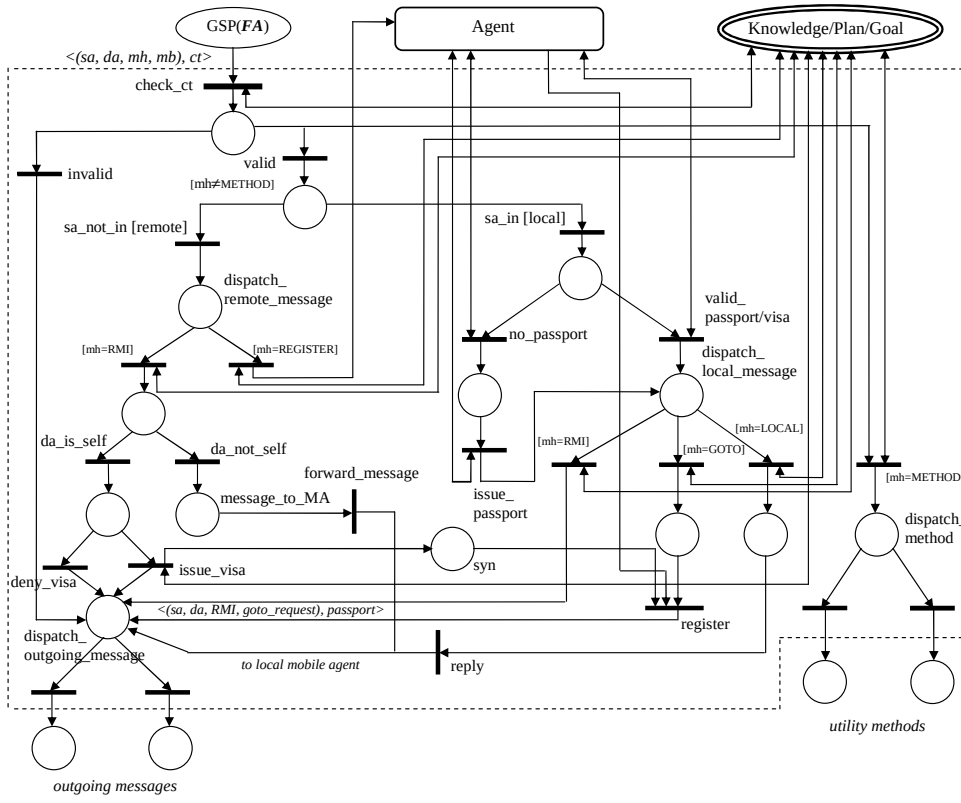


Fig. 6. A planner module of agent-oriented G-net model for facilitator agent.

Goal modules as introduced in Sec. 3. In particular, the *Knowledge* module contains the following three tables:

User table. This table contains information about registered users that can create/register agents on the local AVM. Each record contains the following data: user name, user privilege, and a public key. User name refers to the creator of a local mobile agent on a local AVM. User privilege is a privilege assigned to a user by a local administrator, and the public key is the user's public key issued by a local administrator. Remote FAs are also recorded in this table, and we view remote FAs as remote users.

Agent table. This table contains information about mobile agents residing on a local AVM. Each record contains the following data: agent ID, name, owner and description. Agent ID is the object identification of a mobile agent. Owner, for local mobile agents, is the user who creates that mobile agent; while for those mobile agents that exist due to a migration from another AVM, the owner field is the issuer of the passport, as we have discussed previously.

Policy table. This table contains security policy information about the AVM. Each record contains the following data: privilege and a method set. The privilege refers to the privilege level of a local FA; while the method set is the set of methods that can be invoked under that privilege level. When a mobile agent sends a method request to the local FA in a form of a `LOCAL` method, the local FA decides if the agent has the privilege to request that method based on the information from the user table and the policy table.

We now use a migration example to show how a FA's planner module works. As shown in Fig. 6, when an agent on a local AVM Θ_{A}, say agent α, wants to move to a remote AVM Θ_{B}, the agent α first sends an *RMI* message *move-request* to the *GSP* of the local FA θ_{A} with `mh = RMI`, `da` = θ_{A} and `mb` = "goto request". The actual message received by the local FA θ_{A} is *E-MSG* = $\mathrm{E}(k_{\text{PUB-R}}, (\mathrm{E}(k_{\text{PRIV-S}}, \textit{move-request}), CT))$, where $k_{\text{PUB-R}}$ is the public key of the local FA θ_{A} and $k_{\text{PRIV-S}}$ is the private key of agent α. Notice that to simply matters, we show the message as $\langle(sa, da, mh, mb), ct\rangle$ in Fig. 6. We also do not show the steps for how to decrypt the message *E-MSG* by the local FA θ_{A}, which have been described in Sec. 4.3. The transition *check_ct* models the action of checking the authentication information of a message sender by the FA. The security checking is based on the message sender's certificate extracted from the message and its issuer's information from the FA's *user table*. If the authentication is invalid, the transition *invalid* fires, and a message of *cert-invalid* will be sent to the mobile agent α. Otherwise, the attribute `sa` from the message *move-request* will be checked to see if the message sender is recorded in the *agent table*, i.e., to check if the message is from a local AVM or a remote AVM. In our example, since the RMI message is from the local mobile agent α, the transition *sa_in* fires. Now, if it is agent α's first time to communicate with its local FA θ_{A}, its certificate issued by its local user shall be replaced with a passport. This process is done by firing the transitions *no_passport* and *issue_passport*. After a passport is issued to the mobile agent α, the local message is dispatched for further processing according to its message head `mh`. Since `mh = RMI` in this example, the message is forwarded to the remote FA θ_{B} according to the attribute `da`. When combining the encrypted message $\mathrm{E}(k_{\text{PRIV-S}}, \textit{move-request})$ with agent α's certificate, its original certificate is replaced with its newly issued passport, where $k_{\text{PRIV-S}}$ is still the private key of mobile agent α. Notice that the local FA θ_{A} does not modify the encrypted message $MSG' = \mathrm{E}(k_{\text{PRIV-S}}, \textit{move-request})$; however, it combines the message MSG' with agent α's passport instead of its initial certificate, and encrypts the whole message with the public key of the message receiver, i.e., the remote FA θ_{B}.

When the remote FA θ_{B} receives the forwarded message *move-request*, it first checks the certificate (passport) of the message sender — agent α, as well as the certificate of its issuer, i.e., FA θ_{A}. If the security check fails, a *move-refuse* message (with `da` = α) will be generated by firing the transition *invalid*, and the message will be sent to the mobile agent α via FA θ_{A}. Otherwise, the attribute `sa` from the

forwarded message move-request will be checked. In this example, since agent α is not recorded in the *agent table* of FA θ_B, the transition *sa_not_in* fires. The message is then dispatched according to its message head (`mh = RMI`), and its attribute `da` is checked to see if the destination agent is the receiver agent itself. In this case, since `da =` θ_B, the transition *da_is_self* fires, and a decision will be made either to deny a visa or to issue a visa to the mobile agent α. If the visa is denied, a *move-refuse* message (with `mh = RMI`, `da =` α) will be generated by firing the transition *deny_visa*, and will be sent to the mobile agent α via FA θ_A. Otherwise, a visa will be issued by firing the transition *issue_visa* and a new message *move-grant* (with `mh = RMI`, `da =` α) is generated. Agent α's passport is updated by putting a visa stamp on one of its visa pages, and the updated passport is then attached to the message *move-grant* in its message body and sent back to agent α via FA θ_A. Notice that in Fig. 6, there is a place called *syn*, where a token containing agent α's identity information will be deposited right after a visa is issued. This mechanism ensures that the actual migration cannot happen before a migration request is granted.

Now that mobile agent α receives the *move-grant* message from FA θ_B via FA θ_A, it sends a `GOTO` message to its local FA θ_A. Upon receiving the `GOTO` message from agent α, FA θ_A dispatches it to the appropriate transition for processing according to its message head (`mh = GOTO`), and generates a `REGISTER` message to the remote FA θ_B. To generate the `REGISTER` message, the mobile agent α is first removed from FA θ_A's Agent place and then serialized and embeded in the `REGISTER` message's message body as an attachment. When FA θ_B receives the `REGISTER` message from FA θ_A, it dispatches the remote message according to its message head (`mh=REGISTER`), resumes agent α's execution, and adds the mobile agent α into its *Agent* place. This ends the agent migration process.

When a local mobile agent sends a message to its local FA for any purpose other than migration, the message head of the message is set to `LOCAL`. In this case, when the local FA receives the message, the message is dispatched according to its message head `mh = LOCAL`. By firing the transition *reply*, a reply message is generated and sent back to the local mobile agent.

Similarly, when a FA receives a message with a message head of `METHOD`, the message represents a synchronous method invocation from the FA itself. Under this circumstance, the method is dispatched to an appropriate utility method defined in the *utility method* section of the FA. After the method invocation, the result will be returned to the *ISP* place where the synchronous message is generated.

5. Case Study: Agent Migration

One of the advantages of building formal models for mobile agent system is to help ensure a correct design. A correct design should meet certain key requirements, such as liveness, deadlock freeness and concurrency. Also certain properties, such as the mobility, need to be verified to ensure correct functionality. Petri nets offer a promising, tool-supported technique for checking the logical correctness of a design

[27]. In this section, we consider an example of a simple mobile agent system that consists of only two AVMs with a local FA and a remote FA. The purpose for this case study is to demonstrate that agent communication protocols can be effectively traced in our net model, and more importantly, to verify that a mobile agent can successfully migrate to the *Agent* place of the remote FA if the migration is granted.

5.1. *A simplified net model*

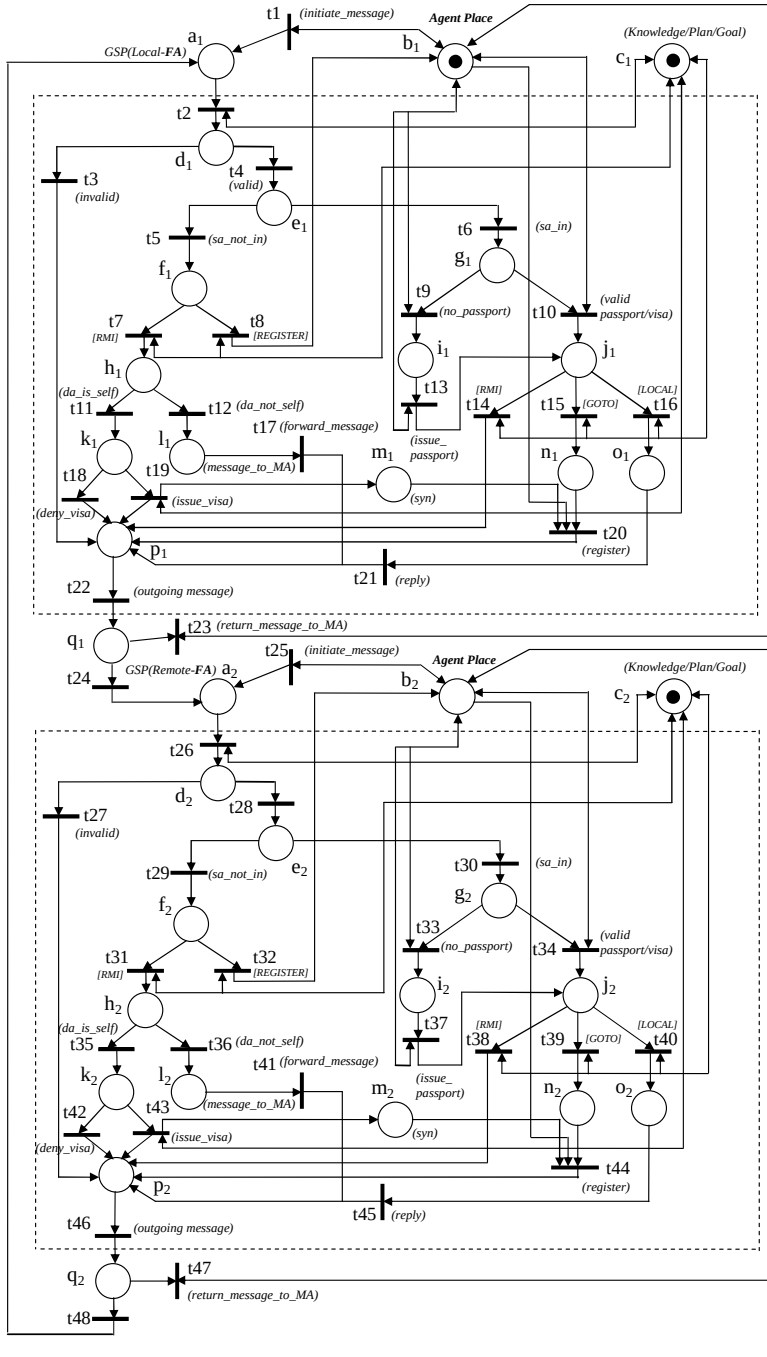
The interaction of a local FA and a remote FA can be modeled as a Petri net as shown in Fig. 7. In this model, there is only one mobile agent (represented as a black token) in the *Agent* place of the local FA. To derive this net model, we have simplified the net of the facilitator agent as follows:

1. Since the transition to dispatch a message with a message head of `METHOD` is only used for local resource access, it does not affect the agent communication and agent migration in the mobile agent system. Thus, to obtain our simplified net model, we omit this transition.
2. Since all asynchronous messages will be dispatched to an *MPU*, and sent asynchronously to the *GSP* place of another agent, we use net reduction to simplify the Petri net corresponding to *MPUs* into a single place. This is shown as the places of q_1 and q_2 in Fig. 7.
3. We use an ordinary token (black token) to represent a local mobile agent in the *Agent* place. The presence of such a token represents that a mobile agent, either created locally or migrating from other AVM, is under the control of its local FA.
4. All other tokens in the net model, including message tokens and tokens belonging to the place *syn*, are simplified as ordinary tokens. Since we have only one mobile agent in our example, such treatment would be sufficient for our purpose.
5. We add a new transition *initiate_message* to enable a mobile agent to send an asynchronous message to its local FA (shown as the transitions $t1$ and $t25$ in Fig. 7). We also add a new transition *return_message_to_MA* to enable a local FA to send an asynchronous message back to a mobile agent (shown as the transitions $t23$ and $t47$ in Fig. 7).

The resulting Petri net illustrated in Fig. 7 is an ordinary Petri net. The initial marking of this Petri net is as follows (with the ordering of place nodes as $a_1, b_1, c_1, \ldots, q_1, a_2, b_2, c_2, \ldots$):

$$M_0 = [0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T .$$

Notice that in the initial state of the Petri net, only transition $t1$ is enabled, which can fire to initiate a conversation between a mobile agent and a local FA.



Local Facilitator Agent

(on local AVM)

Remote Facilitator Agent

(on remote AVM)

Fig. 7. A transformed model of a local facilitator agent and a remote facilitator.

5.2. *Verifying secure agent communication and agent migration*

To verify the correctness of our agent models for secure agent communication and agent migration, we utilize some key definitions and theorems as adapted from [27].

Definition 5.1. *Incidence Matrix*
For a Petri net N with n transitions and m places, the incidence matrix $A = [a_{ij}]$ is an $n \times m$ matrix of integers and its typical entry is given by

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$$

where $a_{ij}^{+} = w(i, j)$ is the weight of the arc from transition i to output place j and $a_{ij}^{-} = w(j, i)$ is the weight of the arc from input place j to transition i.

Definition 5.2. *Firing Count Vector*
For some sequence of transition firings in a Petri net N, a *firing count vector* x is defined as an n-vector of nonnegative integers, where the ith entry of x denotes the number of times that transition i must fire in that firing sequence.

Definition 5.3. *T-invariant*
For a Petri net N, an n-vector x of integers ($x \neq 0$) is called a *T-invariant* if x is an integer solution of homogeneous equation $A^T x = 0$, where A is the incidence matrix of Petri net N.

Theorem 5.1. *An n-vector x is a T-invariant of a Petri net N iff there exists a marking M_0 and a firing sequence σ that reproduces the marking M_0, and x defines the firing count vector for σ.*

Definition 5.4. *Valid Firing Sequence*
Let $\sigma = \langle i_1, i_2, \ldots, i_p \rangle$ be some sequence of transition firings in a Petri net N with initial marking M_0, where i_k, for $k = 1 \ldots p$, is a transition of Petri net N. The firing sequence σ is *valid* if transition i_1 is enabled initially and each transition i_k is enabled after firing transition i_{k-1}, where $k = 2 \ldots p$.

Theorem 5.2. (*Necessary Condition*) *For a Petri net N with initial marking M_0, let $M_d = M_0 + A^T x$, where A is the incidence matrix of Petri net N and x is a firing count vector for firing sequence σ. The firing sequence σ is valid only if M_d is a nonnegative vector.*

The incidence matrix A of the Petri net in Fig. 7 is listed in Table 1. Now we begin to analyze and verify the following five cases for secure agent communication and agent migration, which show the key behaviors and properties of our models. Although the approach using state equation of Petri nets [27] is not new, we believe that it is useful here to demonstrate the correctness of our security based agent models.

Case 1: A *cert-invalid* message is sent by the local FA. In this case, when the mobile agent sends a *move-request* message to its local FA, the authentication is

Table 1. Incidence matrix A of the petri net in Fig. 7.

	a_1	b_1	c_1	d_1	e_1	f_1	g_1	h_1	i_1	j_1	k_1	l_1	m_1	n_1	o_1	p_1	q_1	a_2	b_2	c_2	d_2	e_2	f_2	g_2	h_2	i_2	j_2	k_2	l_2	m_2	n_2	o_2	p_2	q_2
t1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t2	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t5	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t6	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t7	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t8	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t9	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t10	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t11	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t12	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t13	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t14	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t15	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t16	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t17	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t18	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t19	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t20	0	-1	0	0	0	0	0	0	0	0	0	0	-1	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
t27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	1	0
t28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0
t29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0
t30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0	0
t31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0
t32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0
t33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0
t34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0
t35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0
t36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0
t37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0
t38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	1	0
t39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0
t40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	1	0
t41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	1
t42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1
t43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0
t44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	-1	-1	0
t45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
t46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t48	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1

detected as *invalid* due to a fake certificate of the mobile agent or its issuer. The transition t_3 fires and a *cert-invalid* message is sent back to the mobile agent. The firing sequence σ_1 corresponding to this case can be traced in Fig. 7 as follows:

$$\sigma_1 = \langle t1, t2, t3, t22, t23 \rangle .$$

From Definition 5.2, the firing count vector x_1 for the above firing sequence σ_1 can be calculated as follows:

$$x_1 = [1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T .$$

Since we have $A^T x_1 = 0$, from Definition 5.3, x_1 is a T-invariant of the Petri net in Fig. 7. From Theorem 5.1, for any T-invariant x of a Petri net, there exists a marking M_0 and a firing sequence σ, which reproduces the marking M_0, and x defines the firing count vector for σ. Obviously, the firing sequence σ_1 reproduce the initial marking M_0. This is desirable because the *cert-invalid* message ends the conversation (as shown in Fig. 3), and the Petri net should return to its initial state.

Case 2: A *move-request* message is forwarded to the remote FA by the local FA. If the mobile agent passes the authentication test, the *move-request* message shall be forwarded to the remote FA by its local FA. In this case, the transition $t17$ fires,

and the message token shall finally reach the *GSP* place of the remote FA, i.e., place a_2. The firing sequence σ_2 corresponding to this case can be traced in Fig. 7 as follows:

$$\sigma_2 = \langle t1, t2, t4, t6, t9, t13, t14, t22, t24 \rangle .$$

The firing count vector x_2 for the above firing sequence σ_2 can be calculated as follows:

$$x_2 = [1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T .$$

Based on the state equation for the Petri net in Fig. 7, if we assume that the firing sequence σ_2 is valid, we can calculate the destination marking M_1 as follows:

$$M_1 = A^T x_2 + M_0 = [0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T .$$

We find that the place marking $M_1(a_2) = 1$. This is desirable because it implies that the message token can be successfully forwarded to the *GSP* place of the remote FA. Notice that Theorem 5. 2 only states a necessary condition for the validity of a firing sequence, so we cannot use this theorem to prove that σ_2 is valid; however, the validity of σ_2 can be verified by simulating the firing sequence using an existing Petri net tool, e.g., INA (Integrated Net Analyzer) tool [36].

Case 3: A *move-refuse* message is forwarded to the mobile agent by the local FA. In this case, upon receiving a *move-request* message from a mobile agent, the transition $t42$ of the remote FA fires and a *move-refuse* message is sent to the local FA for forwarding. The corresponding firing sequence σ_3 and the firing count vector x_3 for σ_3 are listed as follows:

$$\sigma_3 = \sigma_2 \cdot \langle t26, t27, t46, t48, t2, t4, t5, t7, t12, t17, t22, t23 \rangle$$

$$x_3 = [1\ 2\ 0\ 2\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 2\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1]^T .$$

Notice that σ_3 is defined as the concatenation of sequence σ_2 and the sequence $\langle t26, t27, t46, t48, t2, t4, t5, t7, t12, t17, t22, t23 \rangle$, where the operator "$\cdot$" is the concatenation operator of two firing sequences.

Similar to Case 1, we have $A^T x_3 = 0$. This implies that if σ_3 is a valid firing sequence, it may reproduce the initial marking M_0. Obviously, this is true in Fig. 7, which is desirable because when the *move-refuse* message is forwarded to the mobile agent, the conversation ends, and thus the Petri net should return to its initial state.

Case 4: An attempt to migrate without a visa. In this case, a mobile agent attempts to migrate to a remote AVM without a visa issued by a remote FA. The mobile agent sends a *move* message to its local FA, and the local FA removes the mobile agent from its *Agent* place, serializes it and sends it with a *register* message to a remote FA for registration. Upon receiving the *register* message from the local FA, the remote FA resumes the execution of the serialized mobile agent and deposits

the mobile agent into its *Agent* place. In this case, the corresponding firing sequence σ_4 and the firing count vector x_4 for σ_4 are listed as follows:

$$\sigma_4 = \langle t1, t2, t4, t6, t10, t15, t20, t22, t24, t26, t28, t29, t32 \rangle$$
$$x_4 = [1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T .$$

Similar to Case 2, if we assume the firing sequence σ_4 is valid, we can calculate the destination marking M_2 as follows:

$$M_2 = A^T x_4 + M_0 = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T .$$

We find that the place marking $M_2(m_1) = -1$. From Theorem 5.2, we know that if M_2 is not a nonnegative integer vector, the firing sequence σ_4 is *not* valid. This contradicts our assumption that the firing sequence σ_4 is valid. Therefore, the firing sequence σ_4 is *not* valid. The marking M_2 also shows that some transition in the firing sequence σ_4 cannot fire due to a lack of token in place m_1. This is reasonable because the place m_1 is a synchronization place used to ensure that the actual migration cannot happen before a migration request is granted, i.e., a visa is issued by a remote FA.

Case 5: An attempt to migrate after a visa is issued by the remote FA. In this case, the process involves two steps. In step 1, upon receiving the *move-request* message from a mobile agent, the remote FA issues a visa, and sends a *move-grant* message with the visa as an attachment to the mobile agent, which is forwarded by the local FA. The corresponding firing sequence σ_5 is listed as follows:

$$\sigma_5 = \sigma_2 \cdot \langle t26, t28, t29, t31, t35, t43, t46, t48, t2, t4, t5, t7, t12, t17, t22, t23 \rangle .$$

Now, in step 2, when the mobile agent receives the issued visa, it sends a *move* message to its local FA. The local FA then sends the serialized mobile agent in a register message to the remote FA for registration. This step is exactly the same as the process in Case 4, so we have $\sigma_6 = \sigma_4$.

The firing sequence σ_7 for the whole process (step 1 and step 2) and its firing count vector x_7 can be listed as follows:

$$\sigma_7 = \sigma_5 \cdot \sigma_6$$
$$x_7 = [2\ 3\ 0\ 3\ 1\ 2\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 3\ 1\ 2\ 0\ 2\ 0\ 2\ 2\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1]^T .$$

It is expected that σ_7 is a valid firing sequence, and the token in place b_1 (the *Agent* place of the local FA) will be moved to place b_2 (the *Agent* place of the remote FA) after the firings. We now calculate the destination marking as follows:

$$M_3 = A^T x_7 + M_0 = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T .$$

We find that the place marking $M_3(m_1) = -1$. From Theorem 5.2, we conclude that σ_7 is *not* a valid firing sequence. This result indicates that there are some design errors in our agent model. The place marking $M_3(m_1) = -1$ shows that the synchronization token has not been successfully deposited into place m_1 after

step 1. Looking into the destination marking M_3 again, we find that $M_3(m_2) = 1$. This is not desirable because in our example the remote FA does not need any synchronization for agent migration. This shows that after step 1, the synchronization token has been deposited into a wrong place, i.e., place m_2 instead of place m_1. To correct this design error, we need to revise the planner module of the facilitator agents in Fig. 6 as follows:

(1) modify the label of transition *forward_message* from "forward_message" to "direct_forwarding";
(2) delete the arc from the transition *issue_visa* to the place *syn*;
(3) add a new transition *new_visa* to check if a new visa issued by a remote FA has been attached in a forwarding message;
(4) add a new arc from the place *message_to_MA* to the transition *new_visa*;
(5) add a new arc from the transition *new_visa* to the place *dispatching_outgoing_message*; and
(6) add a new arc from the transition *new_visa* to the place *syn*.

After the revision, when a local FA receives a message from a remote FA, which is to be forwarded to a local mobile agent, the message is first checked to see if a new visa issued by the remote FA is enclosed. If the new visa is enclosed, a synchronization token is deposited into the *syn* place of the local FA before the message is forwarded to the mobile agent; otherwise, the message is forwarded to the mobile agent directly.

Accordingly, we revise Fig. 7 by deleting two arcs: $t19$ to $m1$, and $t43$ to m_2; adding two new transitions: $t49$ and $t50$; and adding six new arcs: l_1 to $t49$, $t49$ to $m1$, $t49$ to p_1, l_2 to $t50$, $t50$ to m_2, and $t50$ to $p2$. The incidence matrix A of the Petri net model (shown in Table 1) is revised into A' by updating the rows for transitions $t19$ and $t43$, and adding two new rows for transitions $t49$ and $t50$. The revisions to incidence matrix A are summarized in Table 2.

Table 2. Revisions to incidence matrix A for the revised Petri net model in Fig. 7.

	a_1	b_1	c_1	d_1	e_1	f_1	g_1	h_1	i_1	j_1	k_1	l_1	m_1	n_1	o_1	p_1	q_1	a_2	b_2	c_2	d_2	e_2	f_2	g_2	h_2	i_2	j_2	k_2	l_2	m_2	n_2	o_2	p_2	q_2
t19	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	1	0
t49	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	1	0

After the revisions, we may verify Cases 1–5 again. Cases 1–4 can be verified in the same way and we get the same expected results. For Case 5, the firing sequence in step 1 is revised into σ'_5 as follows:

$$\sigma'_5 = \sigma_2 \cdot \langle t26, t28, t29, t31, t35, t43, t46, t48, t2, t4, t5, t7, t12, t49, t22, t23 \rangle .$$

The firing sequence in step 2 remains the same, i.e., $\sigma'_6 = \sigma_6$. Therefore, the firing sequence σ'_7 for the whole process (step 1 and step 2) and its firing count vector x'_7 can be listed as follows:

$$\sigma'_7 = \sigma'_5 \cdot \sigma'_6$$

$$x'_7 = [2\ 3\ 0\ 3\ 1\ 2\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 3\ 1\ 2\ 0\ 2\ 0\ 2\ 2\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0]^T .$$

Now, we calculate the destination marking for Case 5 again:

$$M3' = A'^T x'_7 + M_0 = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T .$$

In this case, we find that both place markings $M'_3(m_1)$ and $M'_3(m_2)$ are equal to zero, and the token in place b_1 (the *Agent* place of the local FA) is moved to place b_2 (the *Agent* place of the remote FA). By simulating the net with the firing sequence σ'_7, we find that the firing sequence σ'_7 is valid. This shows that the mobile agent in the *Agent* place of the local FA can be successfully moved to the *Agent* place of the remote FA.

6. Conclusion and Future Work

Agent-oriented software provides a new software engineering paradigm and the opportunities for development of new domain-specific software models. With the continuing improvement of agent technology, and the rapid growth of software system complexity, especially for Internet applications, there is a pressing need for general models of mobile agent systems. Such models can allow a structured approach for design of agent software systems and facilitate the application of formal methods techniques for design analysis and implementation synthesis.

We presented the design models of intelligent mobile agents and intelligent facilitator agents in a framework for agent-oriented software. Unlike previous work, which only models a particular feature of mobile agents, our agent models can serve as a general security based agent model that has the capabilities of mobility, cooperative behavior, and intelligence. Furthermore, our agent models are based on the agent-oriented G-net formalism, which can be translated into a standard form of Petri net (Predicate-Transition net, Pr/T net) [25]. Because the Petri net formalism is theoretically mature and supported by robust tools, our approach supports formal analysis and verification.

For future research work, we plan to implement a prototype following our formal design, by which we can show our approach supports rapid development of mobile agent systems. Finally, we need to explore various security issues in mobile agent design, and verify security mechanism in our models.

Acknowledgments

This material is based upon work supported by the U.S. Army Research Office under grant number DAAD19-01-1-0672, and the U.S. National Science Foundation under grant number CCR-9988168. We thank all anonymous referees for the careful review of this paper and the many suggestions for improvements they provided.

References

1. K. Rothermel and M. Schwehm, Mobile agents, in A. Kent and J. G. Williams (eds.), *Encyclopedia for Computer Science and Technology*, Vol. 40 — Supplement 25, Marcel Dekker, New York, 1999, pp. 155–176.

2. D. Kinny and M. P. Georgeff, Modeling and design of multi-agent systems, in *Proc. 4th Int. Workshop on Agent Theories, Architectures, and Language* (*ATAL-97*), 1997, pp. 1–20.
3. N. R. Jennings, K. Sycara and M. Wooldridge, A roadmap of agent research and development, *Int. J. Autonomous Agents and Multi-Agent Systems* **1**(1) (1998) 7–38.
4. G.-C. Roman, P. J. McCann, and J. Y. Plun, Mobile UNITY: Reasoning and specification in mobile computing, *ACM Trans. on Software Engineering and Methodology* **6**(3) (1997) 250–282.
5. C. Mascolo, MobiS: A specification language for mobile systems, in *Third Int. Conf. on Coordination Models and Languages*, P. Ciancarini and A. Wolf (eds.), LNCS Vol. 1594, Springer-Verlag, 1999, pp. 37–52.
6. G. Cabri, L. Leonardi, and F. Zambonelli, Engineering mobile agent applications via context-dependent coordination, *IEEE Trans. on Software Engineering* **28**(11) (2002) 1040–1056.
7. A. R. Silva, A. Romao, D. Deugo, and M. M. da Silva, Towards a reference model for surveying mobile agent systems, *Autonomous Agents and Multi-Agent Systems* **4**(3) (2001) 187–231.
8. H. Xu and S. M. Shatz, A framework for modeling agent-oriented software, in *Proc. 21st Int. Conf. on Distributed Computing Systems* (*ICDCS-21*), 2001, Phoenix, Arizona, pp. 57–64.
9. H. Xu and S. M. Shatz, A framework for model-based design of agent-oriented software, *IEEE Trans. on Software Engineering* (*IEEE TSE*) **29**(1) (2003) 15–30.
10. C. Argel Iglesias, M. Garrijo, and J. Centeno-González, A survey of agent-oriented methodologies, in *Proc. Fifth International Workshop on Agent Theories, Architectures, and Language* (*ATAL-98*), 1998, pp. 317–330.
11. F. Zambonelli, N. R. Jennings, and M. Wooldridge, Developing multiagent systems: The Gaia methodology, *ACM Trans. on Software Engineering and Methodology* **12**(3) (2003) 317–370.
12. D. Kinny, M. Georgeff, and A. Rao, A methodology and modeling technique for systems of BDI agents, in W. Van de Velde and J. W. Perram (eds.), *Agents Breaking Away: Proc. Seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, LNAI Vol. 1038, Springer-Verlag, 1996, pp. 56–71.
13. H. Xu and S. M. Shatz, ADK: An agent development kit based on a formal model for multi-agent systems, *J. Automated Software Engineering* (*AUSE*) **10**(4) (2003) 337–365.
14. C. Fournet, G. Gonthier, J. Lévy, L. Maranget, and D. Rémy, A calculus of mobile agents, in *Proc. 7th Int. Conf. on Concurrency Theory* (*CONCUR'96*), LNCS Vol. 1119, Springer-Verlag, 1996, pp. 406–421.
15. M. Wermelinger and J. L. Fiadeiro, Connectors for mobile programs, *IEEE Trans. on Software Engineering* **24**(5) (1998) 331–341.
16. J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel and M. Strasser, Communication concepts for mobile agent systems, in *Proc. 1st Int. Workshop on Mobile Agents* (*MA'97*), Springer Verlag, 1997, pp. 123–135.
17. T. Finin, Y. Labrou, and Y. Peng, Mobile agents can benefit from standards efforts in inter-agent communication, *IEEE Communications Magazine* **36**(7) (1998) 50–56.
18. T. Sander and C. Tschudin, Protecting mobile agents against malicious hosts, in G. Vigna ed., *Mobile Agent Security*, Springer-Verlag, 1998, pp. 44–60.
19. W. M. Farmer, J. D. Guttman, and V. Swarup, Security for mobile agents: Authentication and state appraisal, in E. Bertino *et al.*. (eds.), *Computer Security — ESORICS 96*, LNCS Vol. 1146, 1996, pp. 118–130.

20. R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, D'Agents: Security in a multiple-language, mobile-agent system, in G. Vigna, ed., *Mobile Agent Security*, LNCS Vol. 1419, Springer-Verlag, 1998, pp. 154–187.
21. S. T. Vuong and P. Fu, A security architecture and design for mobile intelligent agent systems, *ACM SIGAPP Applied Computing Review* **9**(3) 2001, pp. 21–30.
22. H. Lee, J. Alves-Foss, and S. Harrison, The use of encrypted functions for mobile agent security, in *Proc. 37th Annual Hawaii International Conference on System Sciences* (*HICSS'04*), Big Island, Hawaii, 2004.
23. H. Ku, G. W. Luderer and B. Subbiah, An intelligent mobile agent framework for distributed network management, in *Proc. IEEE Global Telecommunications Conference* (*GLOBECOM'97*), Phoenix, Arizona, November 1997.
24. D. Xu, J. Yin, Y. Deng, and J. Ding, A formal architectural model for logical agent mobility, *IEEE Trans. on Software Engineering* **29**(1) (2003) 31–45.
25. Y. Deng, S. K. Chang, A. Perkusich, and J. de Figueredo, Integrating Software Engineering Methods and Petri Nets for the Specification and Analysis of Complex Information Systems, in *Proc. 14th Int. Conf. on Application and Theory of Petri Nets*, Chicago, June 21–25, 1993, pp. 206–223.
26. A. Perkusich and J. de Figueiredo, G-nets: A petri net based approach for logical and timing analysis of complex software systems, *J. Systems and Software* **39**(1) (1997) 39–59.
27. T. Murata, Petri nets: Properties, analysis and applications, in *Proc. IEEE*, **77**(4) (1989) 541–580.
28. T. Finin, Y. Labrou, and J. Mayfield, KQML as an agent communication language, *Software Agents*, J. Bradshaw, ed., MIT Press, Cambridge, Mass., 1997.
29. M. J. Huber, S. Kumar, P. R. Cohen, and D. R. McGee, A formal semantics for proxy communicative acts, in *Proc. Eighth Int. Workshop on Agent Theories, Architectures, and Languages* (*ATAL-2001*), Seattle, Washington, August 1–3, 2001.
30. FIPA, The Foundation for Intelligent Physical Agents, FIPA 2000 Specification, 2000, http://www.fipa.org.
31. A. L. Murphy, G. P. Picco, and G.-C. Roman, LIME: A middleware for physical and logical mobility, in *Proc. 21st Int. Conf. on Distributed Computing Systems* (*ICDCS-21*), April 2001, Phoenix, Arizona, pp. 524–533.
32. J. Odell, H. Van Dyke Parunak, and B. Bauer, *Representing Agent Interaction Protocols in UML, Agent-Oriented Software Engineering*, P. Ciancarini and M. Wooldridge (eds.), Springer-Verlag, Berlin, 2001, pp. 121–140.
33. W. Diffie and M. E. Hellman, New directions in cryptography, *IEEE Trans. on Information Theory* **22** (1976) 644–654.
34. C. P. Pfleeger and S. L. Pfleeger, *Security in Computing*, 3rd edition, Prentice Hall, 2002.
35. S. Guan, T. Wang and S. Ong, Migration control for mobile agents based on passport and visa, *Future Generation Computer Systems* **19**(2) (2003) 173–186.
36. S. Roch and P. H. Starke, *INA: Integrated Net Analyzer*, Version 2.2, Institut für Informatik, Humboldt-Universität zu Berlin, April 1999.