A Secure Communication Framework for Mobile Agents

Suat Ugurlu and Nadia Erdogan

Istanbul Technical University, Computer Engineering Department,
Ayazaga, 34390 Istanbul, Turkey
suat@suatugurlu.com, erdogan@cs.itu.edu.tr

Abstract. Communication, a fundamental concept in computing, allows two pieces of software to interact and to exchange information. It is an important aspect of mobile agent systems because mobile agents generally need to coordinate their activities through some type of communication. Using mobile agent technologies provides potential benefits to distributed applications; however, an agent's ability to move introduces significant security risks. Consequently, a mobile agent system should provide a safe and secure communication infrastructure along with other security management and maintenance activities. This paper describes the communication framework of a new mobile agent platform, Secure Mobile Agent Platform (SECMAP) that provides mobile agents a flexible and secure communication environment with both synchronous and asynchronous messaging facilities.

1 Introduction

Intelligent agents and multi-agent systems bring in a new approach to the design and implementation of complex distributed systems. Several multi-agent systems have been implemented either as commercial products or in various research projects, with varying success [1] [2] [3] [4] [5] [6] [7]. Reasons for the growing recognition of agent technology are the innovative solutions it provides to problems of more traditionally designed distributed systems through mobility of code, machine based intelligence, and improved network and data-management possibilities.

Using mobile agent technologies provides potential benefits to applications, however, an agent's ability to move introduces significant security risks. Both mobile agents during their life times and hosts executing mobile agents are under security threats [8], [9]. The attacks against mobile agent security can be divided into three categories: attacks by hosts against agents, attacks by agents against hosts and attacks between agents. There may also be more complex attacks that agents and hosts may be exposed to. Consequently, a secure mobile agent system is a firm requirement especially when designing and implementing industrial or e-business applications. Mobile agents generally need to coordinate their activities, and do so by passing messages between them in a location transparent manner. Therefore, a mobile agent system should provide a safe and secure communication infrastructure along with other security management and maintenance activities.

This paper describes the secure communication framework of a new mobile agent platform, Secure Mobile Agent Platform (SECMAP). Unlike other agent systems, SECMAP proposes a new agent model, the *shielded agent model*, for security purposes. A shielded agent is a highly encapsulated software component that ensures complete isolation against unauthorized access of any type. SECMAP provides mobile agents a flexible, location transparent communication environment with both synchronous and asynchronous secured messaging facilities.

2 SECMAP Approach to Security

SECMAP treats every agent as a distinct principal and provides protection mechanisms that isolate agents. The system differs from other mobile agents systems in the abstractions it provides to address issues of agent isolation.

SECMAP agents are light-weight implementations as threads instead of processes. Each agent is an autonomous object with a unique identification and agents communicate via asynchronous message passing. A Secure Mobile Agent Server (SMAS) resident on each node presents a secure execution environment on which new agents may be created or to which agents may be dispatched. SMAS provides functionalities that meet security requirements and allow the implementation of the *shielded agent model*. A shielded agent is a highly encapsulated software component that ensures complete isolation against unauthorized access of any type. On a request to create a new agent, SMAS instantiates a private object of its own, an instance of predefined object *AgentShield*, and uses it as a wrapper around the newly created agent by declaring the agent to be a private object of AgentShield object. This type of encapsulation ensures complete isolation, preventing other agents to access the agent state directly. An agent is only allowed to communicate with its environment over the SMAS engine through the methods defined in a predefined interface object, *AgentInterface*, which is also made the private object of the agent during the creation process. The interface provides limited yet sufficient functions for the agent to communicate with SMAS. All variables of agents are declared as private and they have corresponding accessor methods. Agents issue or receive method invocation requests through asynchronous messages over the secure communication facility of SMAS. Thus, a source that is qualified for a particular request, for example, that has received the rights to communicate with a target agent, is granted to pass its message.

SECMAP employs cryptographic techniques to meet security constraints. Each SMAS owns a certificate which is used to identify its identity and to encrypt and decrypt data. A requests from a SMAS is not processed until the validity of the SMAS identitiy is verified. A SECMAP agent's code and state information are kept encrypted during its life time using Data Encryption Standard (DES) algorithm. They are decrypted only when the agent is in running state on the host's memory. To protect agents during migration over the network, agent code and state data are encrypted as well while in transfer and can only be decrypted on the target host after retrieving the appropriate DES key from the security manager. SECMAP employs a policy based authorization mechanism to permit or restrict agents to carry out certain classes of actions. SECMAP also monitors, time stamps and logs all agent activity in a file, in order to be later analyzed to determine the actions an agent carried out on the host.

2.1 SECMAP Architecture

A brief overview of SECMAP architecture is necessary before the description of the secure communication framework. We have used Java for the implementation of the execution environment because it offers several features that ease the development process. Figure 1 shows the SECMAP architecture. The main component of the architecture is a Secure Mobile Agent Server (SMAS) that is responsible of all agent related tasks such as creation, activation, communication, and migration. The system comprises of several SMAS executing on each node which acts as a host for agents. A SMAS may operate in three modes according to the functionality it exhibits. It can be configured to execute in any of the three modes on a host through a user interface. A SMAS on a node can also operate in all three modes at the same time.

Standard Mode (S-SMAS): S-SMAS provides standard agent services such as agent creation, activation, inactivation, destruction, communication, and migration. It also includes a policy engine that checks agent activity and resource utilization according to the rules that are present in a policy file, which has been received from a Security Manager SMAS. In addition, S-SMAS maintains a list of all active agents resident on the host and notifies the Master Browser SMAS anytime an agent changes state. Keeping logs of all agent activities is another important task S-SMAS carries out. Log content may be very useful in the detection of certain kinds of attacks which are difficult to catch instantly.

Master Browser Mode (MB-SMAS): When agents are mobile, location mappings change over time, therefore agent communication first requires a reference to the recipient agent to be obtained. In addition to supporting all functionalities of S-SMAS, MB-SMAS also maintains a name-location directory of all currently active agents in the system. This list consists of information that identifes the host where an agent runs and is kept up to date as information on the identities and status (active/inactive) of agents from other SMAS is received.

Security Manager Mode(SM-SMAS): In addition to supporting all functionalities of S-SMAS, SM-SMAS performs authentication of all SMAS engines, handles policy management, and maintains security information such as DES keys and certificates. Any SMAS engine in the system has to be authenticated before it can start up as a trusted server. SM-SMAS holds an IP address and key pair for each of SMAS engine that wants to be authenticated. If the supplied key and the IP address of the requesting SMAS engine is correct then it is authenticated. The authenticated SMAS engine gets a ticket from the

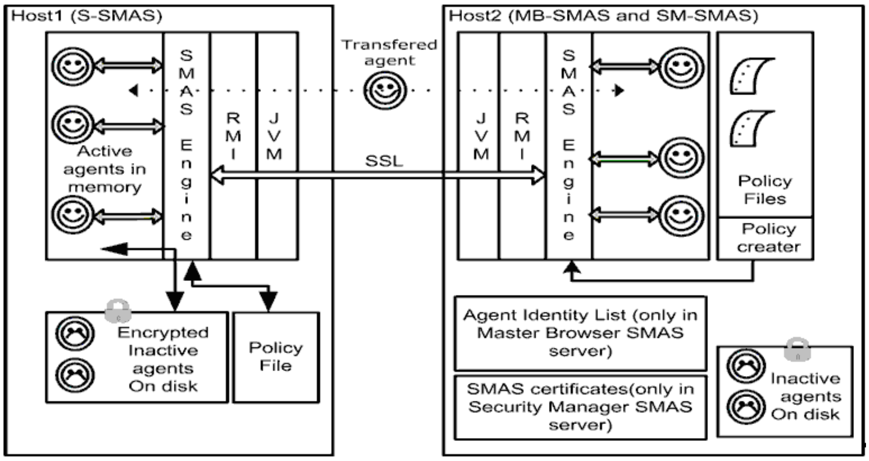
Host1 (S-SMAS)
Host2 (MB-SMAS and SM-SMAS)
Transfered agent
SMAS Engine
RMI
JVM
SSL
Active agents in memory
Encrypted Inactive agents On disk
Policy File
Policy Files
Policy creater
Agent Identity List (only in Master Browser SMAS server)
SMAS certificates(only in Security Manager SMAS server)
Inactive agents On disk

Fig. 1. SECMAP Architecture

SM-SMAS and uses this ticket when communicating with other SMAS engines. A SMAS that receives a request from another SMAS refers to SM-SMAS to verify the validity of its ticket before proceeding with the necessary actions to fulfill the request. Every SMAS, regardless of its mode, creates a private-public key pair once. Next, it creates its certificate and sends it to the SM-SMAS. Any SMAS can receive the certificate list from SM-SMAS before authentication and store it in its key store in order to use SSL communication with other SMAS engines in the system. From then on, all agent-to-agent and SMAS-to-SMAS communication is established through SSL.

2.2 SECMAP Agents

SECMAP requires agents to conform to a software architectural style, which is identified by a basic agent template. The agent programmer is provided a flexible development environment with an interface for writing mobile agent applications. He determines agent behavior according to the agent template given and is expected to write code that reflects the agent's behavior for each of the public methods. For example, code for the *OnCreate()* method should specify initial actions to be carried out while the agent is being created, or code for the *OnMessageArrive()* method should define agent reaction to message arrival.

```
public class Main extends Agent{
public void OnMessageArrive(){...}
public void OnCreate(){ ... }
public void OnActivate(){...}
public void OnInactivate(){... }
public void OnTransfer(){... }
public void OnEnd(){... }}
```

An instance of class *AgentIdentity* is defined for the agent on an initial creation. All agents in the system are referenced through their unique identities, which consist of three parts. The first part, a random string of 128 bytes length, is the unique identification number and, once assigned, never changes throughout the life time of the agent. The second part is the name which the agent has announced for itself and wishes to be recognized with. While the first two parts are static, the third part of the identity has a dynamic nature: it carries location information, that is, the address of the SMAS on which the agent is currently resident, and varies as the agent moves among different nodes. This dynamic approach to agent identity facilitates efficient message passing.

3 Communication Security

Communication security is an important aspect of mobile agent systems. The messages exchanged between agents or between an agent and its owner may be confidential, or can contain sensitive information. It should be possible to detect if messages are tampered with. Also, it should be possible to verify the target entity to which a message is being directed and to verify if a message received really originates from a given entity. Therefore, the communication framework of an agent system should provide facilities for the fulfillment of the following security requirements [10]:

- *Confidentiality*
- *Data integrity*
- *Authentication of origin*
- *Non-repudiation of origin*
- *Non-repudiation of receipt*

SECMAP meets the first three requirements through application of the SSL protocol and cryptographic techniques. All SMAS engines are authenticated before their requests are processed. The implementation relies on Java RMI and it is enhanced to support SSL communication. Non-repudiation of origin and receipt ensures that agents can not deny their actions. SECMAP keeps logs of all agent activity, which can later be analyzed for execution tracing if need arises on conflicts or denial of certain actions.

4 SECMAP Secure Communication Framework

SECMAP agents communicate via messages. SMAS supports asynchronous message exchange primitives through methods of *AgentInterface*. Agent communication is secured by transferring encrypted message content through SSL. Agents are provided with a flexible communication environment where they can question the results of send message requests, wait for responses for a specified period of time, and receive messages or replies when it is convenient for them. Figure 2 shows the communication framework and how a request to send a message proceeds. During agent creation, SMAS, while instantiating a shield object for the

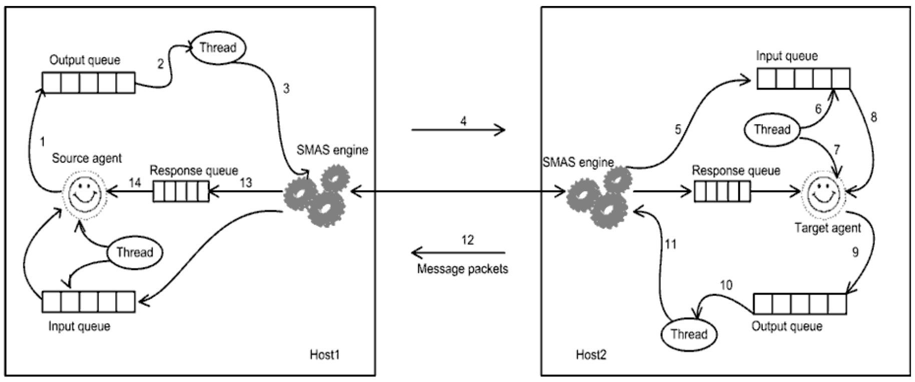

Fig. 2. SECMAP Communication Framework

agent, also creates three queues: one for outgoing messages, one for incoming messages and one for reply messages. The input and output queues are monitored by two threads which are spawned on agent activation. The thread monitoring the input queue alerts the agent if a message arrives, while the thread monitoring the output queue alerts the SMAS engine to route messages to their destination.

When an agent issues a send message call through the *AgentInterface*, the message is placed into the output queue by the agent shield and the call returns. From then on, the agent may continue with its operations. It may question the result of the send request, or, if it expects a response, it may retrieve the reply message at any point suitable in its execution path. The thread monitoring the output queue alerts the SMAS engine to route the message. After the SMAS on the recipient host places the message into the input queue of the target agent, the input queue thread alerts the agent of the arrival of a new message via a call to its *OnMessageArrive()* method,using Java Reflection feature. Subsequent to being alerted, the receiver agent can issue a call to receive the message at any time. Reply messages are also routed as regular messages are. The only difference is that the SMAS engine sending the reply sets the acknowledgement field at the end of the message packet object to true so that the message can be placed in the reply queue of the agent to which the call returns a result. The reply can be retrieved at any time.

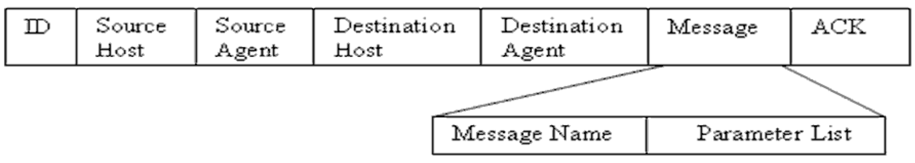

Fig. 3. Message Packet

Figure 3 shows the format of a message packet. Each message is assigned an id which is later used to query the result of a request. Before an agent can send a message to another agent, it needs to learn the name of the receiver agent. An agent learns the identity of the target agent via a call to the SMAS, which cooperates with MB-SMAS to return the required information, an object of type is *AgentIdentity*. Messages are created as instances of the Message class and consist of two parts. The first part is the name of the message, while the second part consists of a parameter list. Parameters can be of any type that can be serialized.

4.1 Agent Communication Interface

SECMAP provides communication security transparently at a lower level and agents are not aware of it. An agent is only allowed to communicate with its environment over the SMAS engine through the methods defined in a predefined interface object, AgentInterface. The methods of AgentInterface related to agent communication are listed below.

sendMessage(String strAgentHostName, AgentIdentity agentidentity, Message message): Send a message to an agent whose identity is known. The call returns the identifier of the message packet, which the agent can later use to query the result of the send operation.

sendBroadcastMessage(Message message): Send a broadcast message to all agents running on the same host. **receive() :** Read and then remove a message packet from the input queue.

sendReply(Packet packet, Object reply): Send the reply of a message which has been received before.

Sent(PIdentifier id): Returns true if a message send request has been carried out successfully, that is the message has been placed into the input queue of the receiver.

ReplyReady(PIdentifier id): Query to learn if the reply of the message has arrived. An agent may not always expect a reply. As communication is asynchronous, in case a reply is expected, the agent queries its arrival at a time convenient for it.

waitForMessage(long ms): Make a blocking call to listen on the input message queue for a specifed period of time.

waitForReply(PIdentifier id, long ms): Make a blocking call to listen on the reply message queue for a specifed period of time and returns true if the reply is ready. If issued right after a send, it leads to synchronous messaging.

getVisibleAgentIdentity(String strIdentifier): Learn the identity of the agent with a specific name running on the same host.

getAllVisibleAgentIdentities(String strIdentifier): Receive a list of the identities of all agents with a specific name running on the whole of the system.

Below are the code fragments of two agents. One of the agents has announced itself with the name *"calculator"* and, on receiving a message with the name *"cal-*

culate" and a parameter list in the form of an arithmetic expression, computes the result and sends it as the reply message. The second agent, a client, wishes to have the result of an arithmetic expression to be computed. In its *OnActivate* method, it constructs a request message, inquires if any *"calculator"* agents are currently active on the system and, if it receives identities of "calculator" agents, scans the list to send its request until a result is obtained.

"calculator" agent:

```
public void OnMessageArrive(){
Packet packet =getAgentInterface().receive();
Message message=(Message)packet.getObject();
if (message.getMessageName().equals("Calculate")) {
Object[]parameters = message.getParameters();
String par1 =(String)parameters[0];
Calc calculator = new Calc(); String result = "";
try\\
{result = calculator.calculate(par1);}
//details of calculator class not included here.
catch (Exception ex){ }
getAgentInterface().sendReply(packet, result);}}
public void OnActivate(){
getAgentInterface().setVisibleOn("calculator");}
```

"client" agent:

```
 Enumeration calcagentlist=
 agentinterface.getAllVisibleAgentIdentities ("calculator");
 //acquire identities of all agents on the system
 //who have announced themselves with the name "calculator"
 while (calcagentlist.hasMoreElements()){
 AgentIdentity agentidentity = (AgentIdentity)
                                 calcagentlist.nextElement();
 Message message = new Message
                     ("Calculate",CalculateInput.getText());
 PIdentifier id = agentinterface.sendMessage
   (agentidentity.getAgentHostName(),agentidentity,message);
if (!agentinterface.waitForReply(id,5000))
    System.out.Println("Timed out.."+"\n");
else{ String result = (String) agentinterface.getReply(id);
System.out.println(CalculateInput.getText()+" = "+result+"\n");
break();}}
```

4.2 Location Transparence

The system is managed with a decentralized control; several MB-SMAS and SM-SMAS may currently be active and they cooperate for a smooth execution.

They share their data and communicate messages to keep it coherent. When initializing an S-SMAS on a node, the programmer specifies the addresses of the MB-SMAS and the SM-SMAS it should register itself to. Next, S-SMAS sends its agent list to MB-SMAS and, in return, receives the identities of all other agents active on the system. All MB-SMAS and SM-SMAS in the system share their data. We call those S-SMAS that a MB-SMAS or a SM-SMAS cooperates with as its partners. When a MB-SMAS gets a request to return an agent identity, it cooperates with its partners to obtain the current agent identities. A similar mode of processing is true for SM-SMAS. If an SM-SMAS can not authenticate a request, it directs it to its partners for possible authentication. Additionally, when an S-SMAS communicates with its MB-SMAS and SM-SMAS, it obtains the addresses of their partners and saves them, in order to use as a contact address in case its communication to its MB-SMAS or SM-SMAS fails. This approach adds robustness against network or node failures. Figure 4 shows MB-SMAS integration to allow location transparent agent messaging.

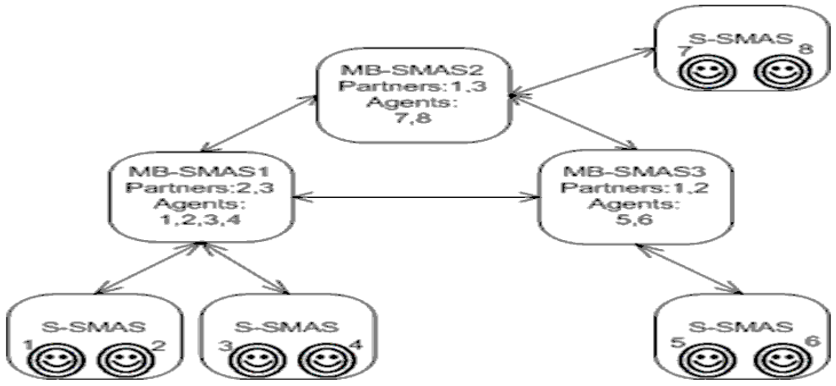

Fig. 4. MB-SMAS integration

5 Related Work

Several mobile agent systems have been proposed and developed up to now. They all have their software agent specific features. These agent systems allow agents to communicate with each other and some have better and more flexible features over the others. SECMAP has the advantage of having a scalable, secure and location transparent messaging architecture. SSL provides message privacy while origion of the message are provided to be safe by authenticating SMAS servers installed on each agent server on the network. Using queues and managing agent queues with agent shields also provides a scalable messaging architecture. Alerting the agent when a message arrives for the agent gives the programmer to write the agent code in an easier way.

6 Conclusions and Future Work

This paper describes the secure communication infrastructure of SECMAP. Mobile agents are provided with a flexible and secure communication environment where they can benefit both synchronous and asynchronous messaging facilities. Confidentiality, integrity, authentication of origin and non-repudiation of messages are assured through security techniques. SMAS are authenticated by several SM-SMAS through certificates, thus introducing trusted nodes to which mobile agents can migrate when required, or send sensitive information. Location tranparency is provided by integrating more than one MB-SMAS into the system, which, at the same time, adds to the robustness of the system. SECMAP keeps logs of all agent activity, such as creation, activation, migration, and message exchange, which can later be used for execution tracing, either for debugging purposes or analysing agent behaviour. Currently, the system does not possess a message buffering feature. Our future work includes adding such a feature into the system so that while an agent is in transit from one host to another, all messages sent to the agent can be kept in the system and the agent be alerted immediately when it is activated on the destination host. Additionally, remote broadcasting support is another feature we plan to add into the system. At its present state, SECMAP supports local broadcasting, that is, a message can only be broadcast to all agents running on the same host as the sender.

References

1. Voyager, http://www.recursionsw.com/products/voyager/voyager.asp
2. Aglets, http://www.trl.ibm.com/aglets/
3. http://www.genmagic.com/technology/odyssey.html
4. JATLite, http://www-cdr.stanford.edu/ProcessLink/papers/JATL.html
5. http://www.cs.dartmouth.edu/ dfk/papers/gray:security-book.ps.gz
6. Bryce C., Vitek J.: The JavaSeal Mobile Agent Kernel, Autonomous Agents and Multi-Agent Systems, 4, 359-384, (2001)
7. Concordia, http://www.merl.com/projects/concordia
8. Sander T., Tschudin C.: Protecting Mobile Agents Against Malicious Hosts, in: Giovanni Vigna (Ed.), Mobile Agent Security, LNCS 1419, (1998), Springer, 44–60
9. Varadharan V., Foster D.: A Security Architecture for Mobile Agent Based Applications, World Wide Web:Internet and Web Information System, 6, (2003), 93–122
10. Borselius N.: Mobile Agent Security, Electronics & Communication Engineering Journal, October (2002), Volume 14, no 5, IEE, London, UK, 211