On Securing Mobile Agent Servers

Javad Chamanara
Software system R&D Lab., Computer Engineering Department, Amirkabir University of Technology
Tehran, Iran.
E-mail: javad@chamanara.com

Mohammad Reza Razzazi
Software system R&D Lab., Computer Engineering Department, Amirkabir University of Technology
Tehran, Iran.
E-mail: razzazi@ce.aut.ac.ir

***Abstract* − In this paper a secure agent server named J-SAS is introduced. J-SAS has an Agent Virtual Machine with the similar functions as a traditional host machine. By assigning passport to agents, authentication, verification of passports, and translation of code to safe version, J-SAS provides a secure environment for executing the agents. Using resource allocation algorithms, ACL and authentication, J-SAS shows a good defensive mood against unauthorized read/ write, denial of service, and masquerading attacks.**

Keywords* − *Security; Mobile Agent; Agent Server; Secure Server; Agent Attack

I. INTRODUCTION

Mobile agents in general terms, are autonomous entities (almost always software) that can run their execution code upon settling on a proper host. They have ability to move from one host to others freely, based on their internal decision making mechanism. Also, they can memorize their mental state before immigration and restore them on the target host to continuing their execution.

Many researches are carried out on the various aspects of mobile agents in recent years and many applications, problems and solutions are provided. Although some mobile agent systems are implemented and have attempted to solve these problems in different ways, unfortunately none of the offered solutions could support all the mobile agent issues in large scale, real systems that act in wide areas like Internet [1], [2].

With regards to various specifications and applications of the agent technology and daily improvements in its efficiency, we will attempt in the present article to offer a server model by which we can receive and execute agents.

Our goal is building a server so that it can execute received agents with their requirements in a proper way, regarding to agent to host attacks prevention.

As we are interested in server-side security utilizing a virtual machine similar to java virtual machine, we have named our proposed server as Java allied Secure Agent Server (J-SAS) in this article.

II. AGENT'S DEFINITION AND SPECIFICATIONS

Our definition of agent is based on Wooldridge-Jennings [3] and SodaBot [4] agents, but with some improvements. Bellow is our exact definition:

Agents are entities with some capabilities such as: calculation, dialogue, negotiation, analysis, decision-making, and cooperation. Also agents enjoys the following properties:

- *Autonomy*: agents operate without the direct intervention of human and others, and have some kind of control over their actions and internal state.
- *Social ability*: agents interact with other agents (and possibly humans) via some kind of inter agent communication language to exchange their experiences, beliefs, and assigned tasks.
- *Reactivity*: agents perceive their environments, and respond in a timely fashion to changes that occur in it.
- *Pro-activeness*: agents do not simply act in response to their environment; they are able to exhibit goal-oriented behavior by taking initiative.
- *Mobility*: agents can halt on a host and resume their execution on another one to satisfy their goals

Based on the above definition, we will discuss the main specifications of our desired agent. These specifications help us in designing a proper agent structure as follows:

A. Mobility

An agent should be able to transfer from one host to another. This transfer can be mentioned explicitly in the agent, or be apparent to the host machine while executing.

Stability, extendibility, low cost and common use of Internet and especially its web service are a proper situation for agent exchange gateways implementation. Using POST method of HTTP is a simple and effective scenario to transfer the agents, so that the whole agent is delivered to agent exchange gateway by POST method as a HTTP Request [5].

Fortunately the existing web servers like MS Internet Information Server (MS IIS) and Apache Web Server running on various operating systems utilize this capability. Thus a module could be designed and embedded on HTTP servers to receive delivered agents and provide them to the server.

B. State Dependency

State dependency means that an agent when arrives in each server, has a primary state, probably acquired from the previous server. That is, run time agent variables have certain values and server should initiate them prior to agent execution.

As Agent State is included in agent structure, the expected server should extract and adjust the run time variables primary values before the agent's execution [6].

C. Communication

An agent may intend to communicate with its surrounding environment [7], [1]. This communication often takes place for these reasons:

- Environment sensing: collect information in order to recognize its environment.
- Inter Agent Cooperation: exchange of information in order to cooperate with other agents in the network.
- Agent Advertisement: distribute information about itself in order to make others aware of its presence and capabilities.

The above-mentioned issues are main specifications of the agent that this article attempts to support them in the next sections.

III. Agent Primary Model

In order to design the primary model of an agent server, a basic structure of the agent should be defined.

Any agent may confront various problems in its life cycle. In order to classify the agent parts, we will do the following:

As from the agent definition, it should be authorized to execute its code, so one of the agent's main structural parts will be "Agent Code". Also when an agent enters or leaves the server, it has a state vector that shows its primary and final "State" respectively. Each agent has some natural specifications like owner, producer and etc. In order to save this information we add another part to the agent structure named "Agent Specification".

As mentioned, agents have the ability to move over various servers. Therefore servers see the agents as external entities that trusting them could not be reasonable. So servers require an agent authentication method and certificates by which they can trust it. To reach this end, we add another part to the agent structure that has all required "Certificates". As result the general structure of an agent will be as in figure 3.1.

Before going to detail of J-SAS, in the next section we will describe general misbehaviors that agents can do against the servers.

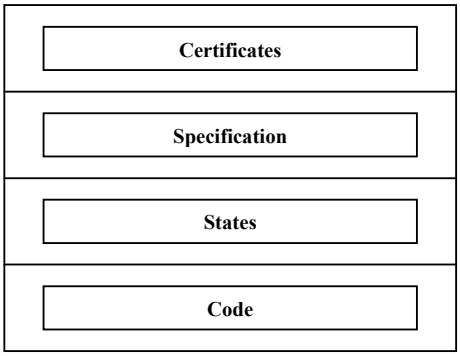

Figure 3.1. Agent general structure

IV. Agent to Host Attacks & Defences Model

We concentrate on providing server security against malicious agents. To do so, we define the malicious agents by describing misbehaviors that agents can do against any server.

Generally the abnormal behaviors occurring by an agent can be classified as below [8], [9], [10]:

- *DIS_ATK*: (Unauthorized disclosure) By this behavior, agent can read existing data, without necessary permissions.
- *MOD_ATK*: (Unauthorized Modification) By this behavior, agent may change or destroy existing data without necessary permissions.
- *DOS_ATK*: (Denial Of Service) By this behavior, agent may unaccustomedly consume server's shared resources, so that it halts other services. Also it can disable other agents by sending them numerous messages.

- *MQR_ATK*: (Masquerading Attack) By this behavior, agent could enjoy the advantages of a trusted user and execute its code, neglecting the server. In other words, agent may masquerade as other agents.

Security model, which we are seeking, attempts to prevent the above-mentioned behaviors throughout agent life cycle. To reach this end, we briefly explain how the server reacts to such behaviors.

A. DIS_ATK

This attack can be applicable only when an agent is capable to read any data from resources. We must establish the access to the resources through a controllable procedure.

It is possible to use ACL[1] mechanism to manage the access to the resources.

So in order to prevent such attacks, a SPM[2] is established in the server. Using SPM, an ACL set is assigned to every useable resource. SPM can assign access level to the entered agents and respond to the security related requests issued by different parts of the server. All the agent requests for accessing the resources will be analyzed by SPM and it will be accepted or rejected after inspection.

To prevent the MOD_ATK, a similar approach can be applied.

To assign ACL to the server resources and the entered agent, following general operations should be accomplished by SPM:

- SPM assigns an access list to each resource. Each list determines in what way agents/ producers/ owners/ users can use the resources.
- After authentication, the entered agent will be delivered to the SPM along with its resulted passport. According to its available ACLs, SPM attaches an ACL to the agent.

After authentication of the delivered agent it will be sent to SPM. SPM will determine its access level. Then the agent will be executed, but in order to access any resources, it must pass SPM policies. Therefore, SPM can monitor the access of the agents to server resources. The above method can be employed to prevent DIS_ATK and MOD_ATK.

B. DOS_ATK

As said before, this attack will unaccustomedly consume the server resources so that other available services in the server won't be able to continue their execution and in some cases these services are stopped by the Operating System due to the lack of the resources. To prevent such attack, some tasks should be done as below:

The first step is to observe the received agent code to find ambiguous cases. Ambiguous cases include using unsafe classes, trying to access objects without proper permission and so on. Using code verification [11], code rewriting and resource accounting [12] algorithms we can over come this problem.

The next step is to design an RFI[3] in the run time virtual machine through which all the agent requests to access to the server resources are passed, before they reach the SPM. Employing the SPM security policies, RFI realize how, when and to what extend, an agent would access the resources, so it decides on the agent request accordingly. This is a technical decision, In other words, RFI examines the request to see whether the needed resources are available in the server and whether the agent quota is remained or not.

Final step is that, agent code commonly contains instructions, which create needed objects for the agent on the server. (In Java, JVM creates the intended object and delivers it to the applicant.) Then applicant may use the properties and methods of that object. This can cause DOS_ATK also. In order to prevent such attacks, server should consider the followings:

- Agent code contains just one class. This means that agent code is not authorized to request the creation of the objects that are along with the agent or they are in an area beyond SPM scope.
- Existing classes in server CLASS-PATH are revised in a way that they do not threaten the server. These classes are called "Safe Class".
- SPM bears the list of safe classes and attaches proper ACL to them.

As discussed above, a certain RFI performs the agent requests on the agent side in coordination with SPM. Therefore, the object creation requests are performed under SPM inspection. So the server will assure that:

- No agent can exceed its allocated quota (RFI determines the maximum accessible resources for each agent).
- No agent is able to contain instruction, which leads to object creation out of safe classes (this function is performed through code verification).
- No agent has direct access to the resources.
- The agents will be executing in the isolated threads.

The above conditions assure that agents will not cause DOS_ATK.

[1] Access Control List
[2] Security Policy Manager
[3] Resource Feasibility Inspector

C. MQR_ATK

As mentioned earlier, the agent contains specifications related to manufacturing company, owner, user and its sending source. The sender may be offline as the agent is received or it may happen that the intermediate server sends the agent. So in order to prevent MQR_ATK following tasks must be performed:

- *Agent Authentication*: this prevents forging the name and specification of the agents through controlling the signature of the manufacturing company.
- *Owner Authentication*: this prevents stealing the agent, forging owner's name or abusing owner's confidence. Authentication can be done by owner or Retailer Company signature.
- *Code Authentication*: we suppose the sender machine (original sender machine and intermediate servers) will have signed agents' code as it leaves the machine. So to authenticate the code, it is possible to use the signatures of original and intermediate machines.
- *User Authentication*: while sending the agent, user signs its own part. This signature will be evaluated in receiving server. In the case of approval, the pre-specified access level in SPM will be devoted.
- *Providing access list*: SPM has the responsibility to provide all resources accessible to agents with appropriate access lists categorized as agent/ owner/ user, and maintain them. After authentication of the agent/ owner/ user/ code, SPM creates and attaches an access control list to the agents. For any resource, this list carries the minimum right among the agent/ owner/ user rights.

Therefore, using the signatures of manufacturing and retailer companies, owner, user and intermediate machines as well as a proper and secure signature pattern, arrival agent will be authenticated, receive their access control lists, and can be executed in a controlled environment.

V. OPERATIONAL SERVER MODEL

The final model of a desirable server is divided into two distinct units. This division helps the stability, extendibility and security of the system. These units as discussed in the sections VI.A and VI.B, have their own specific functional features and can be implemented on different hosts.

A. Agent Gateway

AgG acts as a gateway to the outside world. Its responsibility is to receive, format, authenticate and locate arrived agents. To describe the responsibilities of AgG, its model is shown in figure 5.1and its subunits will be discussed as below:

a) HTTP Server

HTTP server is an application which is already prepared and installed on an operating system and performs HTTP protocol operations. Windows family operating systems use IIS and UNIX family operating systems use Apache as the most common HTTP servers.

To develop the users intended capabilities IIS offers ISAPI (Internet Server Application Programming Interface) technology. Apache can also be developed through designing and implementing modules. So servers are able to send and receive the agents as HTTP POST and GET methods. This helps AgS and AgG to act in a heterogeneous network without changing the existing platforms.

b) Agent Receiver

This unit can be implemented as an ISAPI extension or an Apache module. Also it is supposed to coordinate with HTTP server in order to get the received agent, separate agent code from HTTP header and deliver the agents to the next unit.

c) Preprocessor and Formatter

If necessary, this part is able to check the agent format, different parts of the agent, data integrity, etc. If different agents have different formats, it is also able to form them in a way that local servers could recognize them. Moreover, this unit is responsible for determining the encryption method of agent body if exist.

d) Authenticator

Its responsibility is to extract agent certificates part and authenticate the agent/owner/user/code and previous server, which has sent the agent. Based on needs, capabilities, costs, and standards, different authentication methods could be applied. Using PKI is our main approach. The sender of the agent may generally encrypt and send the agent. If so, the authenticator first decrypts the agent by using a proper algorithm, then it will decide to authenticate it.

Having authenticated different parts of the agent, authenticator issues a passport to the agent. It writes the specifications of the agent/user/code, authentication time and results, its own specifications, and time interval of passport credit on it and signs the passport. This passport is just used internally and it is possible to add AgS list, which can be met by the agent. So the agents are sent to certain AgSs, and can only be run on those AgSs. Regarding the internal state of passport credit scope, authenticator is capable to use Hashing algorithms such as MD5 to sign the passports. So a shared secret key is offered to AgGs and AgSs to be used to calculate and accept or reject the signature. Thus while the agent is being run in AgS, the related units could react properly to its requests using agent passport.

e) Agent Mediator

The responsibility of this unit is to prepare the agent and its passport and coordinate with one of existing AgSs in the network in order to send the agent to it. This unit is aware of the presence of AgSs and AgGs in the network. To balance the load and select proper AgSs, it utilizes all AgGs load information. Having chosen the AgS, agent mediator communicates with it and requests the agent delivery. If AgS accept, agent mediator will send agent to it. Otherwise, agent mediator should repeat this procedure in order to find another AgS. To select the AgS, existing scheduling algorithms can be used.

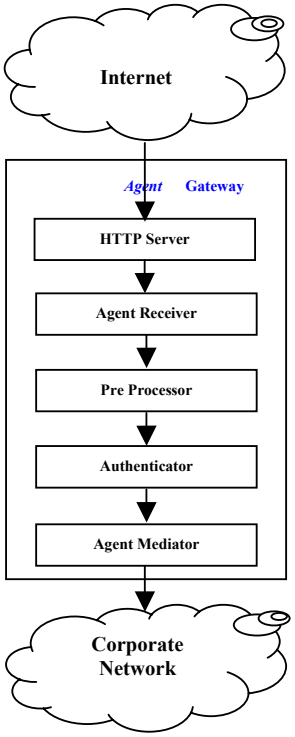

Figure 5.1. Agent Gateway model

AgG and AgS have other communications in addition to receiving and delivering the agents. Therefore, we attempt to use a simple communication model. To do so, AgS is considered as a service, which can communicate through TCP/IP protocol suite. Therefore, AgG or any other consistent tool can communicate with AgS to exchange data.

AgS makes an appropriate structure for received data and according to that structure, AgGs connect to the related service, afterward they will be capable to deliver their data. For this purpose, sender puts one byte pattern at the initial part of the dispatched stream and sends it to AgS. AgS also decides about it's meaning based on that initial pattern.

It is clear that AgS can be applied for exchanging the management and control messages of the system and the messages among the agents in addition to agent transfer, through AGENT_PORT. The reverse procedure can also be applied, that is AgS may need to send a message to AgG to do certain activities. In that case, AgS pack the massage, put the byte pattern on its initial part and send it to AgG. AgG uses HTTP port to receive its packets. In fact, AgS sends its messages to Agent Receiver. The details will be explored in Inter Agent Communication in this article. The following stream will be sent by AgG to AgS in order to deliver the agent.

AgG_i->Send (Code, passport_len, agent_len, Passport, Agent) TO AgS_j

That Code = 0 shows that delivered stream is an agent. Passport contains information and signature of AgG_i and Agent is the received agent by AgG_i, which is supposed to be performed on AgS_j. The passport_len and agent_len are passport and agent lengths respectively in bytes. In this unit, TCP/IP will make communication protocol simple and easy to improve.

As AgS and AgG are kept behind the firewall, there won't be any IP attacks in AgS, as both firewall and the service itself are capable to prevent unauthorized machines to access it. Furthermore, using a passport makes AgS capable to perform AgG authentication.

B. Agent Server

AgS is an application, which can be viewed as a service on the server. On starting, it can be bind to an IP address in private scope and listen to a specific TCP port like AGENT_PORT. This port provides Agent Virtual Machine Handler (AVMH) with all received data so that it decides based on them. In some cases, AgS may need to communicate with outside world so it will deliver its requests to intended AgG and receive the proper answer. Agent server farm is configured in a way that each AgS and AgG obtains a unique name in the network and also each AgG obtains a unique name in the Internet. To reach this end, it is possible to use DNS convention. In this model all formal agreements as choosing domain name, allocating general IP scope, getting related certifications will be called as AgGs.

As shown in figure 5.2, AgS consists of different units as below:

a) Agent Controller

This unit receives the entered agents under the control of AVMH and takes the following steps:

- Checking data integrity of the agent
- Checking data integrity of the agent passport
- Authentication of sending AgG
- Controlling the signature of sending AgG
- Making necessary controls on certain requests

- Signing outgoing requests or agents which are prepared to be sent

b) Safe Code Generator

The responsibility of this part is to check the agent to figure out if there is a reference to a class rather than SPM verified classes or not. If such references exist, SCG[1] attempts to change them to safe references (replace them by safe versions). If there is not a safe class in AgS for a reference, SCG will introduce the agent to AVMH as an unsafe agent.

The operations, from which all agents are forbidden, are taken into account by this part.

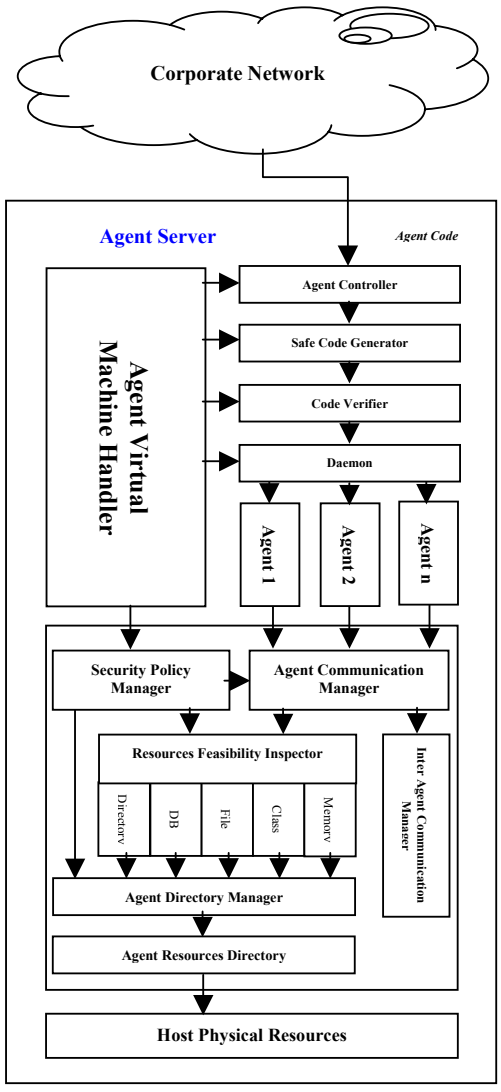

Figure 5.2. Agent Gateway model

[1] Safe Code Generator

c) Code Verifier

This unit is similar to JVM code verifier, but it is optimized to check the agent code. Among its responsibilities are:

- Jumps and loops addresses checking
- Type checking
- Code evaluation in order to estimate intended memory and processor
- Evaluation of the default agent state values with acceptable variables values

d) Daemon

After receiving the agent, this unit checks the possibility of its execution using RFI. In the case of RFI acceptance, agent and its specifications will be added to the active agents list. This task is taken in order to enable SPM to monitor the consumption of the resources, as in each AgS, only SPM is aware of the limitations imposed on agents' operation procedure, maximum number of agents that can be run, accessible resources and so on.

As a list of the running agents is kept in AgS, a data structure named Active-Agent would keep agent information, the general form of which is as below:

```
Structure Active-Agent
{
      Agent-Name
      Arrival-Time
      Arrival-Gateway
      Departure-Time
      Departure-Cause
      Thread-Name (Address)
      Agent  (include code,
      certificates, specifications
      and state)
      Agent-Passport
      Agent-ACL
      Structure Active-Agent *Next,
      *Previous
};
Declare Active-Agent-List AS List of
Active-Agent;
```

It includes the minimum cases, which are considered to manage the agent, and may be changed in practice.

After adding an agent to Active-Agent-List, daemon creates a separate thread (Agent-Thread) and assigns a unique name to it. In this stage daemon initialize the data items of the active agent data structure with appropriate data. Daemon provides passport with SPM in order to receive ACLs of current agent and SPM extracts ACLs of agent from Agent Directory Manager (ADM) and send them back to daemon. Finally, after the above-mentioned activities are done successfully, daemon prepares the execution of the agents by passing their entry point addresses to Agent-Thread.

e) Agent Virtual Machine

AVM provides an execution environment, which includes processor, stack, register bank, program counter, I/O, and so on [11]. In this way the agent assumes that it is run in an actual machine. This machine contains certain instruction set, which is optimized to run agents and facilitate their communication. As this article attempts to utilize the current technologies, we utilize Java Virtual Machine instructions to implement AVM. Each instruction will be compiled in a series of native commands by local machine so that all of them will perform the following sequence of ACTION at the run time:

- To evaluate resource allocation feasibility through RFI.
- To check the security permissions through SPM and ACLs.
- To perform given tasks.
- To update the values of registers, stack, PC and etc.
- To log the related events in order to perform auditing and accounting.

AVM starts to initialize objects and variables by the use of agent state and wherever necessary it instantiates objects. It then resets PC and other registers. After initialization, AVM starts to run the agent instructions line by line. Agent executive model in AVM as can be presented as below:

```
Do {
   Fetch next Opcode;
   If (Opcode HAS Operand) Fetch
   Operands;
   Execute the ACTIONS related to
   Opcode;
} While (There is more to do)
```

In this manner, after reading every Opcode and its Operands, AVM performs their sequence of ACTIONS.

f) Agent Virtual Machine Handler

This unit is basically responsible for coordinating the activities of the other units. Moreover, this unit may be responsible for the following:

- Garbage collection
- Error Handling
- Event logging
- Communicating with AgGs
- Communicating with other AgSs
- Load monitoring

In the following the methods of SPM and RFI activities such as mechanisms of resources access control, access permission adjustment, and offered communicative mechanisms by AgS, are discussed.

g) Access control mechanism

In order to control access to the resources, we suppose that each AgS is running on a specific host, which allocates certain resources to AgS such as memory, processor, file, network and etc. To manage the existing resources, AgS makes a directory that can maintain resources list and its attributes. The directory is introduced in AgS as ARD[1]. ARD is accessible and can be managed through another unit called Agent Directory Manager.

ADM[2] offers a number of functions to add, delete, edit and search ARD entries. Using ADM, other units can request data items such as ACLs, safe class names, files and other resources or manipulate ARD's entries.

So there is a list of accessible resources and their specifications associated with the lists of access control, where them could be managed through SPM. Using the above-mentioned information, we explain how an agent requests a resource:

Agent-Thread reads an instruction from an agent code, which contains the request for using a resource. Agent-Thread sends the request to ACM[3]. By evaluation of the request, ACM is informed that the requesting agent needs the access to a local AgS resource, so it sends the request to RFI.

After assessing the request type, RFI checks whether there are available resources to be allocated. In case of acceptance, RFI will deliver it to SPM to check its security. SPM controls the agents' ACLs and sends the results back to RFI. If all the above evaluations are passed successfully, the resource will indirectly be given to the agent through RFI. "Indirectly" means the agent have no access to resources directly but there is a Resource Allocation Table in RFI which shows that the resource is assigned to which agents with what permissions. Finally, the agent is provided with the index of related item in this table.

It is worth mentioning that considering the continuous development and modification of AgS resources, a policy is made in RFI to support this capability.

To reach this end, we divided RFI into two sections, static and dynamic. The static part is considered to do basic functions and preserve tables and states, while dynamic parts are able to maintain the specific module of each resource. So RFI can recognize the request type and send it to its specific modules. The related module

[1] Agent Resources Directory
[2] Agent Directory Manager
[3] Agent Communication Manager

will do the requested operations and return the results to RFI. The modules for accessing the memory, safe classes, file, network, database and etc, are of this kind. In addition, these modules can operate the actual actions such as reading and writing according to instructions.

In order to implement the above model, we can realize ARD and ADM units by means of LDAP[1]. ARD and ADM are considered as LDAP server. Other units that need to utilize this service (i.e. SPM and RFI) are considered as LDAP Client.

Using LDAP protocol makes it simple to design and apply ARD, ADM, and RFI based on a standard pattern. Moreover, considering that Windows and UNIX families' operating systems support this protocol, it decreases the server development cost.

h) Inter Agent Communication

As mentioned earlier, all the participating members in the model are considered as agents. So you can see ACM section in a way that this unit is supposed to fulfill the communication requests of active agents in AgS. This unit must be able to distinguish the requests related to resources, communications with local agents, and communication with remote agents.

The offered ACM in this article divides requests into two main groups:

The first group includes requests related to local resources and the second group includes the requests related to communication between agents.

ACM will provide RFI with all relevant requests to access the resources and it will direct all communication requests to IACM[2]. If IACM receives a request, will locate the destination. Then in cooperation with SPM, IACM will contact target AgG and will perform the related operations. Finally it will inform the source agent of the request result through ACM.

If the target agent is alive, it can be found in one of the following locations:

- In local AgS
- In one of the AgSs in local network
- In other active AgSs in Internet

IACM can evaluate the first task, as it can easily access the Active-Agent-List. If local AgS lacks the agents, IACM should search the local network AgSs. Requesting it from local network AgG can do this. If local network lacks the agent, IACM should find it in a world as big as Internet. To do this, some algorithms like Agent Broker and Agents Advertisement are offered which will not be discussed here.

[1] Lightweight Directory Access Protocol
[2] Inter Agent Communication Manager

VI. Related Work

We considered major agent platform that are referenced in [2]. As mentioned in this article a mobile agent platform must provide mechanisms for mobility, state dependency, code altering, and communications in a secure way. **Java seal** [13], [8] use special kernel and Java sandbox mechanism and no support for code altering. **AgentSpace** have no built-in security mechanism [14]. IBM Japan **Aglets** have security mechanism that includes authentication, data integrity, confidentiality, and authorization. Aglets utilize ATP[3] for inter agent communications [15]. Aglets are dependent on underlying Java system. Security of the aglets against runtime class loading, when aglets utilize more than one class, is not clear. The definition of **SOMA** allows enforcing layered security policies: the domain defines a global security policy that imposes general rules; each place can apply restrictions to the domain-level set of permissions. SOMA protects hosts against potentially malicious agents by supporting agent authentication and authorization at both domain and place level. SOMA authorization service is designed to perform flexible and fine-grained access controls that depend on both static and dynamic attributes utilizing Ponder [2]. **FarGo** is a development and runtime environment for wide-area distributed applications. FarGo provides a monitoring facility that can measure performance and resource consumption, and can notify the application upon the occurrence of various events [16]. Java Agent Template, **JATLite** facilitates construction of agents using the emerging standard communications language, KQML and provides a basic infrastructure in which agents register with an Agent Message Router facilitator using a name and password, send and receive messages and transfer files [17]. **aZIMAs** (almost Zero Infrastructure Mobile Agents system) that is a mobile agent platform, based on existing Apache web servers and the HTTP protocol utilize Apache and JVM security model [5].

VII. Conclusion

We presented the design and implementation issues that must be addressed by mobile agent platforms. The J-SAS offers strong isolation of agents, efficient and transparent communication and mobility facilities, load balancing over agent servers, fault tolerance against failure of nodes, distribution over network, OS independency, and layered security mechanism. While the agent server is responsible for isolation, internal communication, safe code generation, run time code alteration, and policy based resource control and the agent gateway is responsible for agent interchange, authentication, load balancing, and external communication. The J-SAS mobile agent system is based on a security policy manager architecture providing the necessary security features for commercial

[3] Agent Transfer Protocol

mobile agent applications. The agent gateway and agent server can be implemented in C++ or Java, but because of similarity of AVM and JVM, it is better to implement the agent server in Java. J-SAS is prepared for resource accounting and auditing, which could be used for reporting, billing and intrusion detection.

VIII. REFERENCES

[1] Wayne A. Jansen, "Countermeasures for Mobile Agent Security", National Institute of Standards and Technology Gaithersburg, MD 20899, USA.

[2] Xiaotong Zhuang, "Survey of the security aspects in mobile agent systems".

[3] Michael Wooldrige, Nicholas R. Jennings, "Inteligent Agent Theory and Practice", Department of Computer Science, University of Liverpool, Liverpool, United Kingdom, 1995.

[4] GeoAgent Project, Department of Geography, San Diego State University, USA, 2002.

[5] Amar Nalla, Abdelsalam (Sumi) Helal and Vidya Renganarayanan, "aZIMAs – almost Zero Infrastructure Mobile Agent System", University of Florida, 2002.

[6] Wayne Jansen, Tom Karygiannis, "Privilege Management of Mobile Agents", National Institute of Standards and Technology, National Information System Security Conference, October 2000.

[7] Alberto Silva, José Delgado, "The Agent Pattern for Mobile Agent Systems", 3 rd European Conference on Pattern Languages of Programming and Computing, EuroPLoP'98, INESC & IST Technical University of Lisbon, Portugal, 1998.

[8] Jan Vitek, Ciaran Bryce, "The JavaSeal Mobile Agent Kernel", Proceedings of the Joint Symposium ASA/MA'99, First International Symposium on Agent Systems and Applications (ASA'99) and Third International Symposium on Mobile Agents (MA'99) Palm Springs, California, October 3 - 6, 1999.

[9] Wayne Jansen, Tom Karygiannis, "NIST Special Publication 800-19 – Mobile Agent Security", National Institute of Standards and Technology, August 1999.

[10] Danny B. Lange, Yariv Aridor, "Agent Transfer Protocol -- ATP/0.1", IBM Tokyo Research Laboratory, 1997.

[11] Tim Lind Holm, Frank Yellin, *The JavaTM Virtual Machine Specification*, Second Edition, Sun Microsystems, 1999.

[12] Grzegorz Czajkowski and Thorsten von Eicken, "JRes: A Resource Accounting Interface for Java", ACM Conference on Object Oriented Languages and Systems (OOPSLA), Vancouver, Canada, October 1998.

[13] Walter Binder, CoCo Software Engineering GmbH,"Design and Implementation of the J-SEAL2 Mobile Agent Kernel", 2001 Symposium on Applications and the Internet (SAINT 2001), San Diego, CA, USA, 2001.

[14] AgentSpace: A Next-Generation Mobile Agent System, Telematics Systems and Services Group - INESC, Technical University of Lisbon, PORTUGAL, 1998.

[15] Mitsuru Oshima, Guenter Karjoth, Kouichi Ono, "Aglets Specification 1.1 Draft", IBM Tokyo Research Laboratory, 1998.

[16] Issy Ben-Shaul, Ophir Holder, Hovav Gazit, Boris Lavva, Yoad Gidron, FarGo: Mobile Agent Development Environment, Technion - Israel Institute of Technology, 1998.

[17] Heecheol Jeon, Java Agent Template, Stanford University, 1996.