Towards Secure Agent Distribution and Communication
Korba, Larry

Towards Secure Agent Distribution and Communication

Larry Korba[1], *Member, IEEE*
National Research Council of Canada
Larry.Korba@iit.nrc.ca

Abstract

Development of acceptable agent-based, distributed systems requires secure techniques for agent migration and agent communication. Several Java-based agent frameworks provide these functions to greater or lesser extents. This paper describes an agent distribution framework for intercommunicating agents using Java which targets several key goals: 1) Provide an agent dispatch mechanism to selected nodes from secured agent repositories by one or more authenticated agent deployment applets or agents, 2) Adjust the operating environment for each agent depending upon its role-based authenticated security level. 3) Provide inter-agent communication by way of its own published objects as well as a multicast event service. 4) Provide encryption for communications depending on the security requirements. 5) Provide a distributed naming directory of agents populating the network. 6) Monitor agents proactively for appropriate activity. 7) Keep the environment lightweight, making agents small in size, and the package easy to use.

***Index terms*—Agent, Java, Object Communication, Mobile Agent, Security**

1. Introduction

With the advent of the web and more recently the Java programming language, the potential for distributed agent applications is exploding. To further this movement, several groups have developed mobile agent systems in the Java Programming language for many different target applications. Some basic requirements for an agent environment are:

- The ability to distribute agent entities throughout the network.
- Support for different types of communication among the distributed agents and
- Measures to ensure safe operations by the agents and the agent environment.

Meeting security requirements is fundamental to successful deployment of mobile agent systems [1]. Fully mobile agents compound security issues [2]. A major concern in the deployment of these systems is the ability to prevent illicit operations by any of the distributed entities. The concerns here may be better understood by listing some potential sources of illicit or unwanted operations:

1. Agent corruption during agent storage. A secure repository for the agents is required to ensure that agents are not corrupted while in storage.
2. Corruption during agent transfer. The transfer of an agent must be secured and sources verified to ensure that the agent is not corrupted in transit. It is vital to ensure the integrity and privacy of the agent and the information it may carry.
3. Communication interception and corruption. In a system of distributed agents, communication between agents located at different nodes must be secured to ensure that:
   - sender and receiver are allowed to communicate,
   - message transfers may not be easily deciphered by a third party and
   - the message transfers between the parties are not modified.
4. Corruption or damage to host resources. Agents must be restricted from access to resources on the host. The concern here is unauthorized, potentially harmful access or changes to host resources.
5. Corruption or damage to network resources. Depending on the mandate of an agent, it may be restricted to accessing only certain network resources.

This paper briefly reviews some of the better known developments to date for mobile agent deployment in the Java programming language with an emphasis on communication and security aspects. It also describes an agent communication and distribution environment (ACDE) which uses service negotiation and authentication among other techniques for secure deployment of one-hop agents. The system is part of a core development for network management applications in the area of security.

[1] National Research Council, Room 286B, Building M-50, Montreal Road, Ottawa, Ontario K1A 0R6. (613)998-3967, NRC Paper # 41600

2. Mobile Agent Environments

This section provides a brief comparison of the communication and security features of several popular mobile agent systems. Kiniry and Zimmerman provide a more general overview of these systems [3] [4]. There are currently many mobile agent environments developed by universities or research institutions and by companies [5]. The environments considered for this paper include: Aglet's Workbench [6], Concordia [7], Voyager [8], and Odyssey [9]. Table 1 summarizes current communication and security features of these environments.

	Security	Communication	Comments
Aglets Workbench	Java.lang.SecurityManager extensions	message-based	Primitive security and communication means.
Concordia	java.lang.SecurityManager Secure Sockets Layer Encrypted Storage User Identity	Distributed multicast Events Collaboration communication based on Java's Remote Method Invocation (RMI) API	Excellent security provisions but not available with evaluation license. Authentication not available yet
Objectspace Voyager	java.lang.SecurityManager extension	Multicast Event Object Sharing, Remote Methods	Mobile Agents as an extension to CORBA
Odyssey	java.lang.SecurityManager extension	RMI (DCOM, IIOP optional)	General Magic holds US Patent Number 5,603,031, System and method for distributed computation based upon the execution and interactions of processes in a network

Table 1. Comparison of some of the better known Java-based mobile agent packages.

2.1. Mobile Agent Communication Environments

CORBA and Java's RMI are commonly used for inter-agent communication in a number of environments. These techniques are useful in that they abstract away the communication interface to object exchange and remote method invocation. A local method call may have the arguments serialized and transferred to a remote target class where the call is executed. Any objects returned from the remote method are serialized and returned to the entity which remotely invoked the method. CORBA is designed to work across many computer-programming languages. This makes CORBA especially useful in large enterprise applications where there is a need to interconnect applications written in different languages. It has become an effective way of integrating new client-server developments with legacy applications. In a multi-agent system, this feature may prove useful to interface with packages or modules written in languages other than Java. Essentially CORBA can serve as a bridge for these cases.

To provide this common communication substrate across many programming environments, CORBA uses Object Management Group's Interface Definition Language to provide a common interface for communication. Although IDL provides "the best standard notation language available for defining component boundaries" [23], the disadvantage of this approach is the significant level of overhead required to map IDL constructs into programming.

RMI is a Java-only solution for inter-object communication. It extends Java's object model to support remote method invocation. For building homogenous Java solutions, RMI is a reasonable choice, since it is a bit easier to learn than CORBA and does not require the purchase of an ORB. Both CORBA and RMI require extra overhead, however, due to the requirement of extra servers (for CORBA, an Object Request Broker (ORB), for RMI, RMI Registry). Although some performance tests have indicated that CORBA performs very well for remote method invocation compared to RMI [10], object initialization can take substantial amount of time. Another issue is the Java class library support required for both RMI and CORBA is on the order of 200K. This means that in order to run a CORBA-based applet, a web browser without the class file support for the communication means must transfer this large class file unless the client has specific support for the communication means. Both CORBA and RMI have a considerable learning curve due to the extent of the support library.

None of the Java-based mobile agent packages offer extensive tools for monitoring or controlling inter-agent communication. In contrast, ACDE uses a communication infrastructure in which inter-agent communication may be monitored and restricted as required. The communication system is object based, offering the advantages of RMI by providing remote method invocation for peer-to-peer communication and multicast object messaging for one-to-many communication. More detail on the communication infrastructure may be found in the communication section (5.) below.

2.2. Mobile Agents and Security

With the advent of Java and the extensive use of Intranets and the Internet, the possibilities of widespread use of distributed applications have come into focus. Mobile agent technologies offer developers the advantage of building applications that may be distributed across the network using high levels of abstraction. Although many advantages of mobile agents have been offered, the overriding advantage of their deployment is the ability to balance network and processing load among nodes in a network. Essentially, the mobile agent may be sent to the node where the resources with which it needs to deal are located. By so doing, local communication rates between the agent and its required resources increase, while overall network traffic decreases. Mobile agents may also migrate to nodes that offer more processing resources for performing a task.

In terms of Security requirements, mobile agent environments are problematic. The push to use agent mobility entails having autonomous agents that can roam a network, from computer-to-computer based upon an itinerary generated by the agent, modulated by what the agent senses and its prescribed goals. To allow free agent movement requires a high level of trust as to the legitimacy of the host and mobile agent as well as the integrity and confidentiality of the network connections for agent communication and transport. These are all challenging research issues.

Developers and researchers have taken a variety of approaches to security of mobile agent environments. In recognition of the need to bolster security facilities for the Aglet environment for instance, Karjoth et. al. have suggested a security model which defines an authorization language specifying the principals within an Aglet system and the responsibilities of each of them within the system [11]. The model also defines how the agents might migrate and access local resources, providing a means for handling agent privileges and local preferences. This work does not define the infrastructure for distribution of the security information nor does it address the problem of protection of the internal state of Aglets. In the management of Intranet resources, an approach has been proposed in the form of a role-based access control system for Intranet security [12]. In this case, network objects, humans users as well as all network resources, are given unique identities, permissions, properties and access control lists. Local role manager agents handle validating an object's identity and deriving permissions for accessing resources. Although this work is focussed on access to resources by Intranet users rather than a solution for the Mobile agents environment, it appears that the relation of an agent to other agents and network resources can form a fundamental framework for provisioning access to only those resources required for operation. The approach taken for security implementation within ACDE follows along the lines of role-based access to resources.

Authentication verifies that messages really come from their stated source. There are a number of authentication protocols or frameworks [13]. Chess describes an itinerant agent framework to support mobile computing [14]. Digital signing of itinerant agents as the sole security means has been suggested to have serious limitations [15]. In cases where mobile agents roam widely, the identity of the author or the owner of the signed agent may be unknown. For widely roaming agents, a high level of trust would be required by the operator of a network node when a foreign mobile agent would like to use resources. In this case, the author or the owner of the agent may be unknown. The operator needs to have more information about the source of the agent, the benefits that will accrue to the operator from the agent's operation, knowledge of the agent's role, and the full extent of resources the agent will require. Agent authentication using the details of the role of an agent could help provide some level of trust in the agent. Authentication and cryptography in general, however, is problematic. There are questions of scalability, especially when considering the level of verification required (valid digital certificate, public key validity, certificate not expired, certificate not revoked), the lack of interoperability and the difficulty of managing many certificates and keys. [16]. One suggestion for a scalable and open security solution, presented by Hsu and Seymour, uses short-lived certificates [17]. Their approach would provide benefits to Intranets where there is a great deal of control over both client and server. It is not apparent how this technique may be directly applied to mobile agents.

Security policies for mobile agents, i.e. information regarding movement of an agent, resource access and communication history are important areas of research [18]. Problems of security with mobile agents are outlined in various papers contained in [19]. Some key issues are protection from damage or overuse of the network or host resources, protection of private information contained in either the host or mobile agent and protection from damage of either the agent or host code [20].

3. System Overview

The system presented in this paper was designed to target the development of distributed, agent-based, security applications in the area of Network Management. Secured access to wireless local area networks is the first application of this system [21]. For many network management applications, mobile, multi-hop agents are not necessary. Indeed, it has been argued by many [22] that the key benefit of agent mobility is dynamic resource usage or load balancing for distributed applications. For network management applications, agent mobility provides a means for populating a network with required agents for the distributed system. Appropriate design and location selection of agent systems and agents reduces the load on network and host resources within a target network. Single-hop agents, that is agents that can only move from a secure agent repository to a target execution environment, increase the level of trust in the agent and agent system since the agent has not followed a complicated itinerary and it is coming from a trusted source. The goals in the development of the agent distribution and communication environment described in this paper are:

1. Single-hop agent dispatched from secure agent storage,
2. Agent authentication to provide agent authorization based upon the agent’s role,
3. Negotiation for and security management of network and host resources,
4. Inter-Agent communication via remote object sharing and multicast events.
5. Secure communication,
6. Monitoring and logging of resource usage and,
7. A distributed naming service to support multiple authorized managers for the agent system monitoring.

The system takes the following approach. All agents are digitally signed. To simplify security infrastructure, the agents are single-hop, meaning that they can only move from a "safe" agent storage place to an agent meeting place called the Agent Daemon. The safe agent storage place is a restricted access directory of a web server. Agents may also operate in a web browser. All agents have predefined roles for operation within an agent system. The predefined roles of an agent include information concerning its resource requirements and levels of interaction and authority in dealing with network and host objects (databases, computer resources, network resources, Agent Daemons, or other agents). Information regarding the agent systems and authority level details for agents are stored in a secure trusted server. Agents are authenticated via the trusted server when they arrive at an agent daemon. Agents operating within Agent Daemons and within browsers must negotiate with Agent Daemons for network resources. Finally, in order to operate in the Agent Daemon environment, agents must implement an interface which permits Agent Daemon monitoring of the agent communications.

4. Block Diagram

The ACDE deploys Agent Daemons throughout the network to provide an environment at network nodes to which agents may be dispatched. Agent Daemons provide resource allocation based on agent authentication, the agent requirements and authority. Figure 1 depicts the functions supported by the agent daemon.

A Security Manager manages security of the operating environment for the agents. The Security Manager extends the java.lang.SecurityManager Class to monitor activities within the threads of the agent. It also handles authentication negotiation with the agents that wish to operate at the Agent Daemon network node. The communication manager handles requests from agents to set up communication services and monitors agent communication activity, allowing only authorized intercommunication for the agent. The network services controlled by the Agent Daemon include:

1. Publishing of objects,
2. Requesting of objects from other agents,
3. Request of network-based Agent Daemon services,
4. Request for multicast services, and
5. Socket usage.

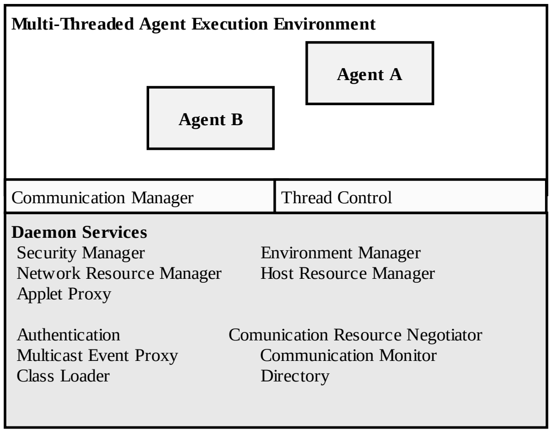


Figure 1. Block Diagram of the Agent Communication and Distribution Environment.

When an agent wishes to publish an object, it must negotiate with the communication manager for available resources. The communication manager determines whether or not the agent is authorized to use communication resources from a trusted server. The trusted server contains reference information about the agent and the agent system(s) with which it is associated. Among that information is a list of the types and targets of interactions the agent is allowed to make. This technique provides more fine grain control over communication permissions that those offered by Security Manager extensions.

5. Communication

Distributed Agent systems have entities operating at different network nodes. To operate in an ensemble they must be able to communicate with each other. One goal with this agent distribution system is for an easy-to-use, versatile, yet lightweight communication infrastructure.

Java RMI (supported within JavaSoft's Development Kit (JDK) Ver. 1.1 and above) takes advantage of Java's Object model. It is a much simpler architecture than CORBA and easier to use. It does have the requirement of a number of steps or rules to make a Java class remote-enabled [24]. The number of classes and interfaces representing the API for the framework are 26 (JDK Version 1.1.3). An RMI Registry program must also be running for each node with remotely enabled objects. Although RMI does an admirable job for providing communication, it still is more complex and requires more resources than the approach taken for communication within ACDE.

ACDE builds upon the communication techniques developed by Ennio Grasso at CSELT (Centre Studi e Laboratori Telecomunicazioni S.p.A., Italy) called the Java Reflection Broker [25]. The package uses several of Java's APIs: Sockets, Object Serialization and Reflection. It deploys a dynamic invocation model similar to the dynamic invocation interface in CORBA. In contrast with CORBA or RMI, remotely enabling a Java class requires no modifications to the class whatsoever. No stubs and skeletons (as used in RMI) need be developed for remote enabling. The communication package works with JDK 1.1x compliant browsers. To publish an object, the agent or applet creates for itself something akin to a mini ORB. Remote method invocations provide point-to-point communication. The package also provides an event service for an asynchronous, multicast, connectionless communication. In this one-to-many communication model, a sender sends a message, and a number of recipients, registered for the message type and implementing an interface to receive these multicast messages, receive it. Not only is the package very easy to use, it is also small, requiring a class file size of about 14K bytes for both client and server sides in contrast to class file sizes an order of magnitude larger for RMI or CORBA approaches.

The communication services provided within the ACDE for the agents are extended to include a "back door" type of communication monitoring system. To be allowed to operate within an Agent Daemon, agents must maintain objects reflecting the identity and nature of communications with its published class. An Agent Daemon may access information within the Agent regarding recent data exchanges. The information includes: contents of communication streams, identities of entities connecting for remote invocation, multicast events requested and received. This provides much more detailed monitoring possibilities for communication monitoring than would be possible with a simple extension of Java's Security Manager class.

To secure communication interchanges between agents within any agent system operating with the ACDE, the system deploys Secure Sockets Layer version 3 (SSL v 3)). This protocol authenticates and encrypts Transmission Control Protocol connection. The protocol is popular and appears to have become a standard for secure network communication. There are a number of third party implementations available in Java, for instance [26].

6. Security Management

Single-hop agents provide a straightforward way of distributing agents to nodes within the network. This eases some of the security requirements from the distribution point of view. Agents are only served from one secure host, rather than moving between many host computers as with other mobile agent systems. In such systems there is greater potential for agent alteration by hostile hosts or by other entities during agent transmission. With ACDE, agents may pass state information to other agents and authorized agents may destroy or deploy other agents. This process is intended for populating a network with an agent system or for dynamic load balancing, i.e. load balancing for an agent system while in operation. Although this technique may be used to provide the effect of node-to-node mobility, it is intentionally less convenient than with truly mobile agent systems. Essentially it offers a reasonable compromise in terms of lowering security risks while at the same time providing agent mobility functions.

Within web browsers, Java's sandbox approach to browser security management places severe limits on access to host and network resources for untrusted applets. The sandbox restrictions may be relaxed through the use of digitally signed code [27]. When classes are created, they are signed with a digital certificate based upon the author of the class and the contents of the class. Successful authentication of delivered classes assures first of all that a class has not been modified during transfer. Second, systems can be configured to allow different levels of privileges depending on the author of the class.

ACDE uses code signing for all agents. This assures that the classes received are the same as the classes at the agent archive server. Rather than allowing the signed agent immediate access to resources, the Agent Daemon authenticates the identity and role of the Agent through interchanges with a trusted server. This approach authenticates the agent as a bona fide member of an agent system that has permission to operate within the network. The interchanges with the trusted server also provide an added level of assurance against agent corruption and provide a means for restricting or relaxing agent operations on the basis of agent role. When agents and agent systems are created, their names, roles, required resources, relationships between agents and agent systems, and lifetimes of agent systems are stored in a secure database within the trusted server. A developer uses a Security Management program to interact with the Security Server in the creation of these entries within this database. The database also relates agent names to the communication relationships with other agents, Agent Daemon services, and network services. A full description of the role-based security provisions of ACDE is the subject of a future paper. Figure 2 schematically illustrates the Security Server while Figure 3 illustrates the use of one-hop agents in ACDE.

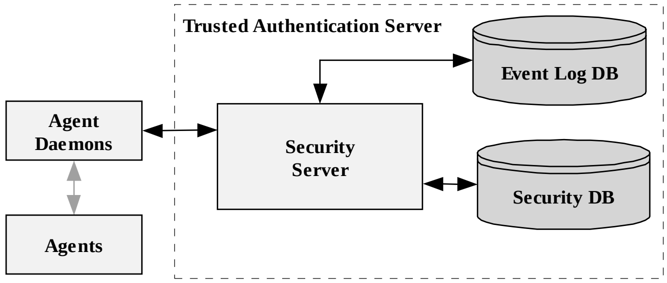


Figure 2. Trusted Authentication server for ACDE.

The Security Server provides two functions: 1) generation of ANSI X9.17 keys for authenticated agents and users and 2) management of a log of security events. Data Encryption Standard (DES) is the basis for encryption used by the Security Server. Although 56 bit encryption keys are considered weak by today's standards [13], DES provides reasonable software implementation speeds and adequate protection against attacks from individuals or organizations other than those with massive resources. In cases where a higher level of security is required (e.g. financial transactions) triple DES (112 bit effective key length) may be used within the present framework.

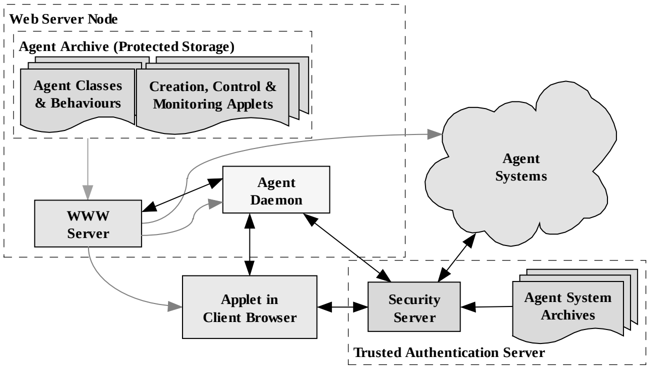
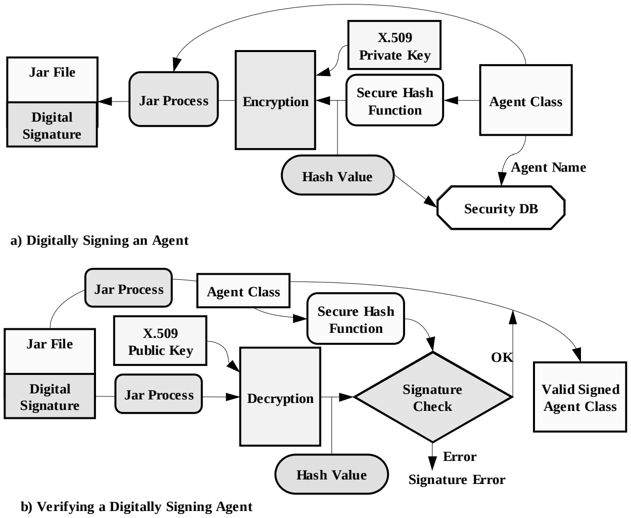

Web Server Node
Agent Archive (Protected Storage)
Agent Classes & Behaviours
Creation, Control & Monitoring Applets
Agent Systems
Agent Daemon
WWW Server
Applet in Client Browser
Security Server
Agent System Archives
Trusted Authentication Server

Figure 3. This diagram illustrates how agents are distributed to Agent Daemons and Browsers.

The process of Agent Daemon and agent authentication, and agent resource negotiation proceeds as follows:

1. Whenever a user starts an Agent Daemon, its digital signature is checked through interchanges with the trusted server. A user name, password authentication exchange permits an Agent Daemon to access agents stored in a protected area of a web server and to act as a trusted entity

for authentication exchanges with the Trusted Server on behalf of deployed agents. This process not only provides some assurance that an authenticated user has started the Agent Daemon, but also that its version is correct.



a) Digitally Signing an Agent

b) Verifying a Digitally Signing Agent

Figure 4 a) Creating digitally signing agents and b) Verifying digital signed agents.

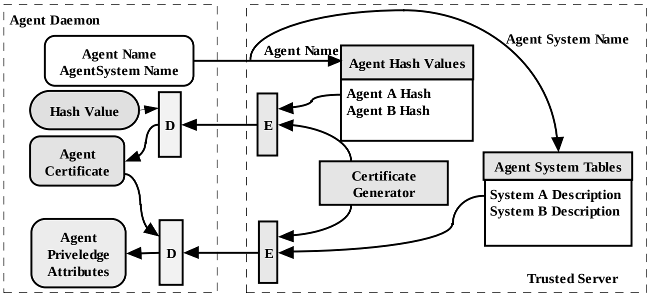


Figure 5. This diagram illustrates some of the interchanges between the Agent Daemon and Trusted Server for certificate and agent security information interchange for individual agents. Blocks E and D are Encryption and Decryption blocks.

2. Through a remote method invocation, the Agent Daemon loads agents from the agent storage area. The Agent Daemon's class loader loads the agent as a signed class. Even though the agent' digital signature has been verified, it is still not yet considered by the Agent Daemon to be trusted. Using Java's Security Manager mechanism, the agent is not allowed access to resources, other than communication with the Agent Daemon security and communication managers until step 3 is completed. Figure 4 depicts the process of digitally signing the agent and receiving a signed agent within the Agent Daemon.

3. The Agent Daemon interacts with the Trusted Server to authenticate the agent as a member of an agent system scheduled for network deployment. The agent name is passed in plain text, the hash value derived from the Agent certificate provides a simple means for securing a channel for exchanging key information. This technique protects the shared DES key during start up of an agent or agent system. Agent security information and agent system information is exchanged using a key generated for the session by the Trusted server. Figure 5 depicts typical agent authentication interactions.

4. The Agent Daemon and the trusted server perform a nonce exchange periodically to ensure the validity of certificates for agents running at the Agent Daemon node [13].

5. If the agent has authority to publish classes for communication, it must negotiate with the Agent Daemon for allocation of communication resources for this purpose immediately. If the Agent does not do so, the Agent Daemon kills the Agent and adds a security violation to the event log.

The Trusted Server monitors, time stamps and logs a number of different events including:

1. Successful and failed attempts of access to the security database.

2. Breaches within the agent daemon security manager. For instance, the Agent Daemon notifies the Trusted Server if an agent attempts to overstep its bounds in terms of host or network resources, or contact with other entities.

3. Creation and destruction agents.

The system administrator is notified whenever any of several settable trigger events occurs. For instance, if there is a repeated authentication failure for an Agent trying to operate at an Agent Daemon, the Trusted Server sends a message to the system administrator by email. As well, subsequent requests from the source address for further authentication are rejected until the system administrator investigates. These events typically would be evidence of a brute force attack on the agent system.

7. Distributed Directory

A distributed agent directory service provides a means for locating agents without requiring a centralized naming or directory server. X.500 Directory services can provide a versatile, distributed directory service for this purpose, providing services for security, search, requesting, administration, access controls and replication. ACDE is intended for use within subnetworks consisting of several hundred nodes whereas X.500 directory services are intended for larger enterprise applications. The complex APIs for implementing X.500 directories are out of keeping with the lightweight philosophy behind ACDE. The service deployed in ACDE uses multicast events to update authorized naming service users of changes in agents resident at any Agent Daemon. Using this information, an agent may determine the physical configuration of agents within a sub-network and inform other agents of their counterparts within agent systems for communication purposes. Figure 6 illustrates one operation of the distributed directory service when deployed from an agent operating as an applet within a web browser. Within some web browsers, an applet is restricted from opening a multicast socket. A remote method invocation between the client agent applet and a multicast reflector service of the Agent Daemon, residing at the node of the web server that served the applet, effectively provides the multicast service. The reflector service generates a multicast message and uses a callback mechanism, making the Agent Daemons deliver each of their respective portions of the distributed directory to the client agent. The operation is similar for agents residing at agent daemons, except that the multicast reflector service is not required.

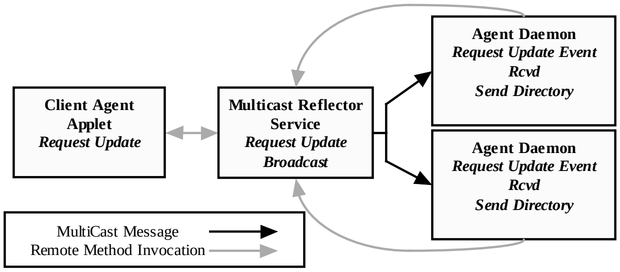


Figure 6. Block diagram illustrating the operation of the naming service for the agent distribution and communication environment.

Authorized agents access the distributed directory and other Agent Daemon services to facilitate the deployment of agent systems. These Agent System Deployment Agents are also authorized to use the information within the security database concerning the agent system to deploy agents and make the necessary agent introductions for communication purposes in the process of building a target agent system. At the same time, agent-monitoring tools may use the directory to monitor agent locations and communication activities.

The simple scheme used in the distributed directory service for ACDE is not meant to replace the potential benefits of an X.500 directory service. The intention is to have an easy-to-use, lightweight directory service. Although IP Multicast provides an efficient means for transmitting information to a group, it does not guarantee packets will be reliably and sequentially delivered to all group members. Several approaches have been suggested for increasing IP Multicast reliability [28]. Although in the current deployment of ACDE IP Multicast has not been an issue due perhaps to small size of Multicast messages (single packet) and light network load, this issue is one that will require careful testing.

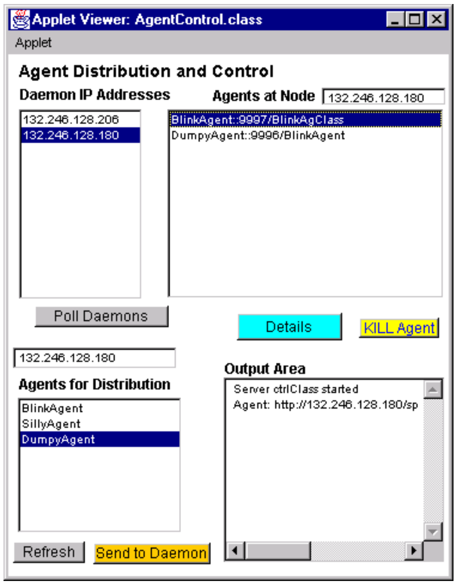
8. Agent Monitoring and Control

The communication issue is a prominent one in the Mobile Agent computing environments. For multiple agents to negotiate, share or allocate tasks, make arrangements, etc. they must communicate. While vital for agents to do their tasks, agent systems with many intercommunicating agents can be very difficult to build without appropriate tools to monitor communications. The Agent Daemon of the ACDE monitors agent communications through a monitoring class within the class library used by ACDE agents. An authorized agent or applet may remotely set environment settings within agents causing them to queue information exchanges they have with other entities. An authorized agent may also inspect the internal methods of the published agent class to determine communication details. Figure 7 shows a screen shot of an agent distribution and monitoring client. The applet lists:

1. The agents available for distribution from the agent repository (bottom left box of Figure 7), in this case, located at the 132.246.128.180 address.
2. The Internet Protocol addresses of the Agent Daemons (top right box of Figure 7). These addresses are updated automatically as Agent Daemons are started and stopped using the distributed directory infrastructure.
3. The selected published classes and their assigned communication ports resident at any Agent Daemon (large box on the upper right side of Figure 7). The items in this list are also updated automatically using the distributed directory.

The applet also provides a text area to display messages resulting from ACDE operations. Whenever a user selects an agent daemon IP address the applet lists the agents resident at the chosen node and the communication ports and names assigned to published classes of the agent. By selecting an agent daemon node and an agent, then clicking on the "Send to Daemon" button, the agent will be sent to the target Agent Daemon from the agent repository.

When an Agent Daemon and an agent are selected and the user presses the "Details" button, a window similar to the one shown in Figure 8 appears. When the user selects a "Published Class" within the agent, the text area at the lower left of the window displays the list of methods and the attributes of those methods and the associated variables. A list at the right of the window displays methods that may be remotely invoked within the agent from the monitoring applet. This feature, implemented using Java’s Reflection class is useful for testing or debugging purposes.



Agents at Node: 132.246.128.180

Daemon IP Addresses:
132.246.128.206
132.246.128.180

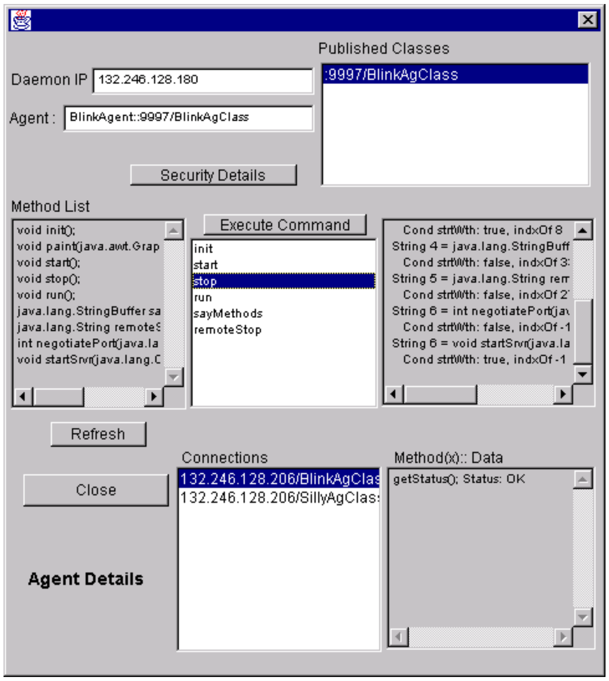
BlinkAgent::9997/BlinkAgClass
DumpyAgent::9996/BlinkAgent

132.246.128.180

Agents for Distribution:
BlinkAgent
SillyAgent
DumpyAgent

```
Server ctrlClass started
Agent: http://132.246.128.180/sp
```

Figure 7. A window snapshot of an agent distribution and monitoring applet.



Daemon IP: 132.246.128.180

Agent : BlinkAgent::9997/BlinkAgClass

Published Classes: :9997/BlinkAgClass

Method List:

```
void init();
void paint(java.awt.Grap
void start();
void stop();
void run();
java.lang.StringBuffer sa
java.lang.String remoteS
int negotiatePort(java.la
void startSrvr(java.lang.C
```

init
start
stop
run
sayMethods
remoteStop

```
  Cond strtWth: true, indxOf 8
String 4 = java.lang.StringBuff
  Cond strtWth: false, indxOf 3:
String 5 = java.lang.String rem
  Cond strtWth: false, indxOf 2'
String 6 = int negotiatePort(jav
  Cond strtWth: false, indxOf -1
String 6 = void startSrvr(java.la
  Cond strtWth: true, indxOf -1
```

Connections:
132.246.128.206/BlinkAgClas
132.246.128.206/SillyAgClas:

Method(x):: Data: getStatus(); Status: OK

Figure 8. A screen shot of the detail screen for an agent.

The lower two message boxes in the dialog window provide communication information. The "Connections" box provides a list of agents that have made connections to this agent. The other list itemizes the exchanges made with the selected agent class.

9. Conclusions

This paper presents a system for securely distributing intercommunicating agents. Although there are several prototype and release versions of mobile agent packages written in the Java programming language available, these packages offer far more than required in terms of agent mobility to provide the agent distribution function, low security provisions and poor provisions for monitoring of agent communications. The approach taken with ACDE is Designed for security applications in Network Management, this agent distribution and communication environment uses single hop agents. Although this technique reduces an agent’s ability to roam, it simplifies security implementation and for most applications while still providing the ability to move an agent to a target node and control its execution. The system provides a mechanism for agent authentication and for authorized agent access to various network and computing resources, on the basis of the roles of agents within an agent system.

To summarize agent instantiation, an agent may be instantiated by an Agent Daemon only if:

1. The agent exists in a secured access agent repository.
2. The agent is properly signed by the agent creator.
3. The agent and its function and resource requirements are detailed in an agent system database stored in a trusted server.
4. The agent system is currently scheduled for operation.

An agent must negotiate with its Agent Daemon for communication resources. If it does not negotiate correctly for the resource, the agent will be terminated. Agents use a communication library which provides a communication back door for the Agent Daemon to monitor agent communications. This feature has been very useful during the development of agent applications. The Agent Daemons and a diagnostic Agent Control and Monitoring program provide an excellent set of tools for monitoring and time stamping inter-agent communication streams. The system may be used to simply distribute Java applications throughout a network. To do this, few changes are required to the application code.

The system is currently undergoing further development and refinement while new security applications of intelligent agents in the network management area are being developed. Until now the agent systems developed for the system include only a low number of agents (ten at most). Although initial results are promising in terms of the level of performance for these small, network management agent systems we are developing, further testing is needed to assess the scalability of the communication and security measures.

10. References

[1] C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile agents: Are they a good idea? Technical report, IBM Research Report, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, March 1995. http://www.research.ibm.com/massive.

[2] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for mobile agents: Authentication and state appraisal. In Proceedings of the Fourth European Symposium on Research in Computer Security, pages 118-130, Rome, Italy, September 1996. Springer-Verlag Lecture Notes in Computer Science No. 1146.

[3] Joseph Kiniry and Daniel Zimmerman. A Hands-on Look at Java Mobile Agents. IEEE Internet Computing, July-August, 1997.

[4] Joseph Kiniry and Daniel Zimmerman. Addendum to "A Hands-On Look at Java Mobile Agents, A Look at Mitsubishi's Concordia", http://computer.org/internet /v1n4/kiniry.htm

[5] Links to other mobile agent tools collected by AgentBuilder. http://www.agentbuilder.com/ AgentTools/

[6] IBM Aglets Workbench. http://www.trl.ibm.co.jp/aglets/

[7] Mitsubishi Concordia Agents Information Page. http://www.meitca.com/HSL/Projects/Concordia/Welcome.html

[8] Objectspace Voyager Information Page. http://www.objectspace.com

[9] General Magic Odyssey Information Page. http://www.genmagic.com/technology/ mobile_agent.html

[10] Robert Orfali, Dan Harkey. Client/Server Programming with Java and CORBA, John Wiley and Sons, 1997.

[11] Gunter Karjoth, Danny B. Lange, and Mitsuru Oshima. A Security Model for Aglets, IEEE Internet Computing, July-August, 1997, http://computer.org/internet

[12] Zahir Tari and Shun-Wu Chan. A Role-Based Access Control for Intranet Security, IEEE Internet Computing, September-October, 1997, http://computer.org/internet

[13] B. Schneier, "Applied Cryptography." John Wiley & Sons, 1996.

[14] David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris and Gene Tsudik. Itinerant Agents for Mobile Computing. IEEE Personal Communications, October, 1995, pp. 34-49.

[15] Tom Walsh, Noemi Paciorek, David Wong. Security and Reliability in Concordia, Hawaii International Conference on Software Systems, HICSS 31, January 6-9, 1998, Hawaii.

[16] Steve Steinke. Authentication and Cryptography. Network - Strategies and Solutions for the Network Professional., Vol. 13, No. 1, pp. 51-57.

[17] Yung-Kao Hsu, Stephen P. Seymour. An Intranet security framework based on short-lived certificates, IEEE Internet Computing, March-April, 1998, http:computer.org/internet/

[18] Fred B. Schneider. Towards Fault-Tolerant and Secure Agentry. 11th Int. Workshop on Distributed Algorithms, Saarbrücken, Germany, Sept. 1997.

[19] http://www.cnri.reston.va.us/home/koe/bib/mobile-abs.bib.html

[20] Tomas Sander, Christian Tschudin. Towards Mobile Cryptography, Technical Report 97-049, International Computer Science Institute, Berkeley.

[21] Larry Korba. Securing Wireless LAN Access: A Network Management Approach. IEEE Int. Symp. on Wireless Communication, Montreal, P.Q., May 22-23, 1998.

[22] UMBC Agent email list. http://www.cs.umbc.edu/agentslist/

[23] Tom Mowbray. The Essential CORBA, John Wiley & Sons, 1995.

[24] Sunsoft Java Development Kit Information Page. http://java.sun.com:80/products/jdk/rmi/ index.html

[25] Java Reflection Broker Information Page. http://andromeda.cselt.it/users/g/grasso/free.htm

[26] Phaos Inc. http://www.phaos.com /main.htm

[27] Sunsoft Java Security Information Pages. http://java.sun.com/marketing/collateral/security.html

[28] Christopher Metz. Reliable Multicast: When many must absolutely positively receive it. IEEE Internet Computing, Vol. 2, No. 4, July-August, 1998, pp. 9-13.