Title
Anchor Toolkit - a secure mobile agent system

Permalink
https://escholarship.org/uc/item/2594j56c

Authors
Mudumbai, Srilekha S.
Johnston, William
Essiari, Abdelilah

Publication Date
2008-06-16

Anchor Toolkit

(A Secure Mobile Agent System)

Srilekha Mudumbai, Abdeliah Essiari, William Johnston
Imaging and Computing Sciences Division
Ernest Orlando Lawrence Berkeley Laboratory
University of California

Abstract: Mobile agent technology facilitates *intelligent operation* in software systems with less human interaction. Major challenge to deployment of mobile agents include secure transmission of agents and preventing unauthorized access to resources between interacting systems, as either hosts, or agents, or both can act maliciously. The Anchor toolkit, designed by LBL, handles the transmission and secure management of mobile agents in a heterogeneous distributed computing environment. It provides users with the option of incorporating their security managers. This paper concentrates on the architecture, features, access control and deployment of Anchor toolkit. Application of this toolkit in a secure distributed CVS environment is discussed as a case study.

1.0 Introduction

A software agent is "*a software entity that functions continuously and autonomously in a particular environment, often inhabited by other agents and processes*" (Shoham 1997)[1]. An agent can be imagined as an autonomous piece of code. This code can initiate actions, construct plans of action, form its own goals, respond to appropriate events, plan its itinerary, communicate with other agents, collect information etc. It can push (send) itself to a remote host and can be pulled (withdrawn) from the remote host by the local host where it originated. An agent can be deactivated to avoid consuming system resources when idle, and can be reactivated later to resume its actions. An agent can be dispatched to any other system by itself or by an agent system and can be retracted to its origin whenever necessary. It can be cloned to execute an action in parallel in several systems. Finally, it can be disposed off after terminating all of its tasks. An agent can be categorized based on the tasks it is assigned. It can be mobile (able to migrate from one system to another), collaborative (able to work with other agents in a system towards a common goal), communicative (communicate with other agents to gather information), or may exhibit several other behaviors.

2.0 Purpose and System Goals

Current computing environments often consist of distributed software running on heterogeneous platforms. Some of the problems that emerge in such an environment include maintaining consistent software, monitoring remote execution, load sharing, asynchronous interaction of remote events, and using heterogeneous systems. Mobile agents can be used to help solve these problems. They can provide a variety of services including fault tolerance and recovery (when a resource server dies, an agent can continue the service provided by that server or restart the server); distributed version control (if the same application is distributed on different hosts, an agent can keep the version of these applications consistent); distributed repository sharing (agents can share databases/repositories distributed at different hosts through concurrent version control); monitoring (e.g. of resources that are accessed in a system, in real time); and auditing (agents can report a history of accesses made on a resource server for a particular time period). An agent system should be capable of providing the essential features required for mobile agents.

Security is one of the most challenging issues that arises when attempting to send executable code to a remote system. The mobile agent infrastructure should identify any entity, whether an agent or an agent system that acted maliciously. In order to accomplish this, the Anchor toolkit has the following goals:

- The agent systems have to identify and trust each other and be able to communicate in a secure fashion.
- The code must be transmitted in a manner such that the receiving host can verify its integrity.
- The execution host must be able to restrict the actions of the mobile code (e.g., limiting access to local files or the amount of resources it may use).
- In case of the agent's failure, the execution host must securely notify the sending host.

An agent system must be authenticated before sending its agents and agent tasks should be subject to access control. An agent's integrity, and possibly confidentiality, must be ensured as a prerequisite to access control and subsequent operation on the remote system. The Security Sockets Layer (SSL)[2] protocol provides basic authentication, confidentiality, and integrity. Access control is exercised by setting policies for access to the resources. LBL has implemented a distributed access control system, called Akenti[3,4], that enables distributed management of access rights for resources that have multiple, independent, and geographically dispersed stakeholders (resource owners).

Akenti is used by an agent system to provide access control. It is designed to use currently available distributed security technologies. It is a module that can be hooked into any application to provide access control. Akenti uses public/private key signed certificates to express user identity, resource use conditions, and user attributes; Public Key Infrastructure (PKI[5]) certificate authorities (CAs[6]) and Lightweight Directory Access Protocol (LDAP[7]) servers manage the certificates. Once mutual authentication between communicating systems is established over SSL, Akenti is called to exercise access control on the resource accessed, to give out capabilities (actions allowed on that resource) for the remote system [sec 4.2].

3.0 Anchor

3.1 *Architecture*

The architecture aims to provide:

- A user-friendly toolkit that supports minimal but essential features required for a mobile agent framework.

- A monitoring capability that displays real-time access of resources by local and remote agents.
- An agent transfer protocol that hides actual transport mechanisms.
- Effective security to support integrity and/or confidentiality between communicating systems and access control to prevent unauthorized access to resources.
- A 100% pure Java implementation.
- Customizable security whereby users can incorporate security managers of their choice.

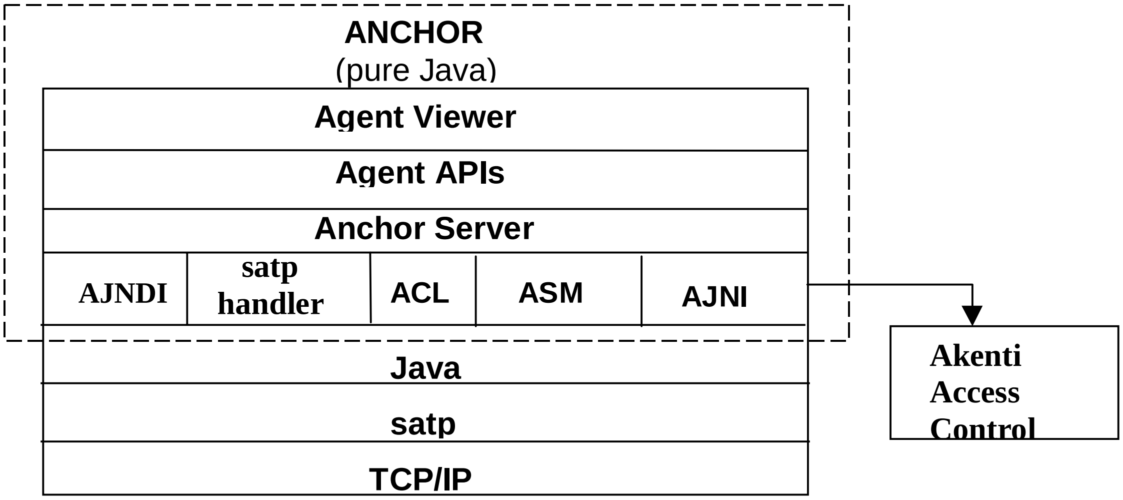


Figure 1: ***Anchor Architecture***

The architecture of the Anchor toolkit is displayed in Figure 1. It consists of an Agent Viewer (GUI), Agent APIs, Anchor Server, Anchor Security Manager (ASM), Anchor Class Loader (ACL), ***s***ecure ***a***gent ***t***ransfer ***p***rotocol (satp) handler, Anchor Java Naming and Directory Interface (AJNDI) and Anchor Java Native Interface (AJNI). ASM and ACL are discussed in later sections.

3.2 *Agent Model*

This model as displayed in figure 2 follows IBM Aglets[8] model. In our model, *agents* are serializable Java objects that are capable of migrating from one machine to another by carrying code as well as their state. They resume their operation after reaching the other machine. They are autonomous as they run as individual threads. They are created within a *context*. A context is the namespace under which agents are grouped together. It is a stationary object that is responsible for maintaining and managing active agents in a uniform execution environment. Context and namespace are used interchangeably. The namespace consists of host name, port and the name of the context itself. There can be several such contexts in a server and there can be several agents created within the same context.

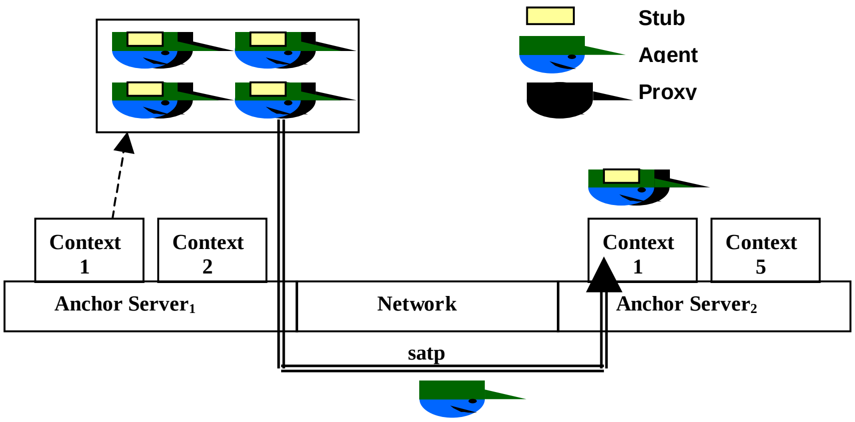


Figure 2: ***Agent Model***

Agents can be accessed only through their *proxies*. A *proxy* represents an agent and prevents other agents from directly accessing its public methods. In other words, any message or action to be conveyed to an agent is forwarded through its proxy and the agent may accept or deny such messages or actions. A proxy provides location transparency for agents, meaning that it represents an agent in a machine even if the agent has migrated to other machines, thereby hiding the agent's genuine location. Anchor Toolkit supports open agent architecture. An agent remembers other agents that it comes across irrespective of their location as long as the other agents are alive. A proxy created for an agent in a location other than its origin does not become invalid even if the agent is dispatched to other locations. Instead it virtually represents the agent so that its reference held by the other agents remain valid. Of course, proxy becomes invalid when the agent expires. At any time, all the proxies that exist in a context can be acquired. Agents can communicate with other agents only if they are in the same namespace. This holds even if they are dispatched to a remote host. In that case, they can communicate with the agents in their remote namespace and the agents within the namespace from where they were dispatched. Each agent has a unique identifier assigned to it during its creation and is immutable throughout their lifetime.

3.3 *Anchor Server and Agents*

The Anchor Server is a run-time environment that serves as the backbone for the toolkit. It runs on a host by listening to a specific port. It sets the security manager for the current environment. It performs all system-related functions. It keeps track of all the agents that are currently running on its machine and provides information on their status. It creates an agent within a particular context. Once the agent gets initialized, it starts executing. An agent can be cloned in which case an identical copy of the agent is created except for the identifier. The clone then restarts execution in its own thread.

Agents can be dispatched by the server to a remote place. As a result the agent is removed from the current context and is inserted into the destination context where it restarts execution. This is a push model as the agent is pushed from one context to another. Anchor Server handles the transport mechanism to migrate agents from one machine to another through satp. Similarly agents can be retracted to their original context from the destination context. This is a pull model as the agents are pulled back to their original context. Agents can be activated and deactivated. Deactivating an agent removes it from the current context and puts it in secondary storage. Re-activating it inserts it into the same context and resumes its action. Check-pointing an agent stores its current state (assumed to be stable) into secondary storage, from which it can be used later in case of any failures. Disposal of an agent halts its current execution and removes it from the current context. Agents should be able to communicate effectively with the other agents. This is achieved through message passing and is currently under implementation. The server accepts each incoming agent, authenticates the identity of its source [sec 4], and passes the authenticated agent to the appropriate context. When an Anchor Server is shutting down, it notifies all the agents currently running in the server. The agents can then decide to move to other Anchor Servers. Otherwise they are de-activated, and re-activated when the server restarts.

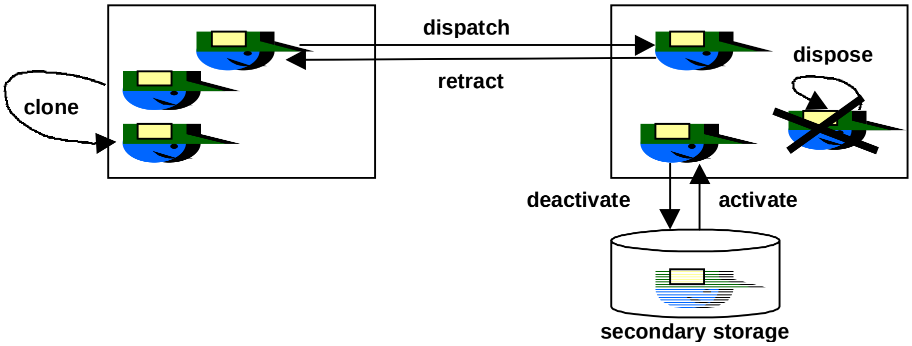


Figure 3: ***Agent features***

Agents implement a checkAccess() method through which they can restrict other entities from executing sensitive operations pertaining to their mobility and existence. These operations are not controlled by the agent context. Instead the agent implementor is responsible for building its own policy which promotes its autonomous nature. Currently each agent assumes a unique identity from where it was indited [ref sec 5.0]. Authentication occurs only between Anchor Servers and not agents. The agent's identity is considered only at the resource access level.

3.4 *Anchor Java Native Interface (AJNI)*

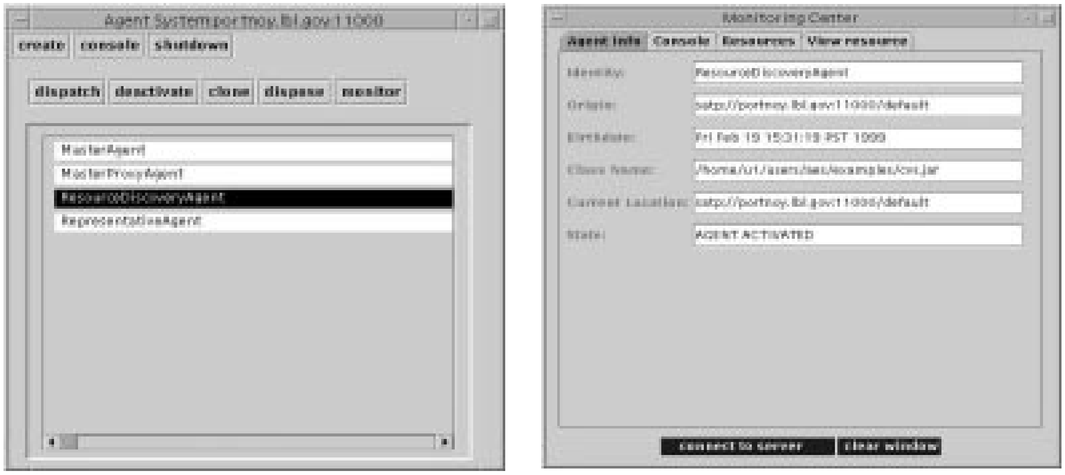
A Java native interface is implemented to invoke Akenti access control module whenever authorization is required for an identity. Akenti requires a client’s identity and the identity of the resource it is trying to access in order to make access control decisions. When an agent tries to perform any sensitive operation on a local resource, Akenti's policy engine determines the agent's authorization for the current operation by marshaling and verifying all the policies that are applicable for that resource and the appropriate credentials. Akenti is a C/C++ implementation of a distributed access control mechanism. In the future we are planning to implement Akenti entirely in Java.

3.5 *Agent and Server APIs*

Agent APIs currently support cloning, activation, deactivation, checkpointing, dispatching, retracting, and disposal as discussed in section 3.3. Server APIs support monitoring, information, console and shutdown.

3.6 *Agent Viewer*

The Agent Viewer reflects the features supported by the Anchor toolkit. One can create, dispatch, retract, dispose, activate, deactivate and clone agents via the Agent Viewer. In addition, it provides information about an agent (either local or remote) and a monitoring capability to display in real time resources used/accessed by an agent in a server. It also keeps track of the current status of the agent so that it carefully activates only those features that are applicable to the current state of the agent.



MasterAgent
MasterProxyAgent
ResourceDiscoveryAgent
RepresentativeAgent

Identity: ResourceDiscoveryAgent
Origin: satp://portnoy.lbl.gov:11000/default
BirthDate: Fri Feb 19 15:31:19 PST 1999
Class Name: /home/u1/asen/sec/examples/cm.jar
Current Location: satp://portnoy.lbl.gov:11000/default
State: AGENT ACTIVATED

Figure 4: ***Agent Viewer and Monitor***

The Monitor is a separate entity by itself that can be plugged into different applications and is shown in the right side of Figure 4. When an agent is chosen for monitoring, the monitor keeps track of all the resources the agent attempts to access. It exhibits all these resources and provides detailed information on the access control decisions made on them. Agent Info provides details about an active agent. It consists of an agent's identity, creation time, context name, origin, current location and its state. The console is used to monitor the current actions performed by the agents within that server. Access to the console is allowed for only one agent at a time in order to avoid confusion and aid debugging. It follows the concept of Netscape's Java console. The server has a separate console that monitors its actions.

3.7 *Anchor Java Naming and Directory Interface (AJNDI)*

Mobile Agents are autonomous. They can visit any host in the network. This leads to a problem of trailing them. A registry mechanism needs to be in place to facilitate locating agents. This mechanism consists of a naming service through which every agent can register and publish its current information and a directory service to enable effective searching of agents. The Anchor Server implements a directory service by itself through its agent context. But if there were several agent systems within a local area network, a unified directory service would be more appropriate. LDAP can be used to store agent information. AJNDI uses Java's naming and directory service to connect to LDAP for fetching and storing objects and other related attributes. The term 'registry' would refer to LDAP in future discussions. An entry for the agent in the registry appears as

- **Common Name** = "*Resource Discovery Agent*"
- **Agent ID** = <*Java serialized Object*>
- **Agent Service** = <*Service available through this agent*>
- **Public Key** = <*Java Serialized Object*>
- **Codebase** = *satp://server.lbl.gov:1000/*
- **Java Factory** = <*Factory Name*>
- **Java Reference Address** = <*Reference address for object reconstruction*>

The **Common Name** attribute refers to the unique identity of an agent in X.400 format. An agent updates its entry in the registry through its creator whenever it sojourns a different location. The service provided by an agent is available through its **Agent Service** attribute. It is not essential for an agent to carry its public key to diverse locations as it can be procured from the registry. The **Codebase** specifies the Anchor server from where the agent originated. Optionally a reference to the **Agent ID** could be stored along with its factory in lieu of the object itself. Registry supports the following operations:

- **Lookup -** *querying the database for locating agents.*
- **Register -** *procedure through which an agent binds itself to the registry.*
- **UnRegister** – *operation for unbinding the agent from its registry.*

4.0 Akenti

4.1 *Access Control Model*

Akenti is an access control system designed to address the issues raised in allowing restricted access to distributed resources which are controlled by multiple stakeholders. The stakeholders are the people with authority to grant access to resources and may be both physically and organizationally remote from the resource. Akenti enables these stakeholders to remotely and securely create and distribute instructions authorizing access to their resources.

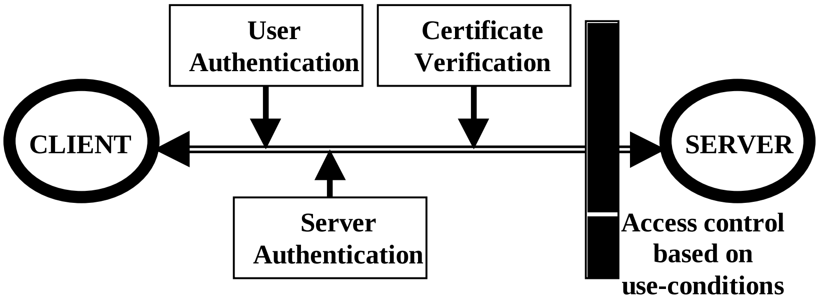


Figure 5: ***Access control model***

Access control is a means for enforcing an authorization policy. In a client-server architecture, the clients (on behalf of users) attempt to access resources that are controlled by servers. Akenti makes the access control decisions based on a set of digitally signed documents that represent the authorization conditions. Existing public-key infrastructure and security systems provide confidentiality, message integrity, and user identity, during and after the access decision process (Figure 5).

4.2 *Components of the Model*

- An *Identity* (X.509[9]) C*ertificate* mainly consists of user's distinguished name, user's public key and the signature of a trusted CA.
- A *Use-Condition Certificate* consists of: a conditional expression of attributes and values, name of the resource, scope of the resource in a hierarchical tree, access/action information, issuers of the use condition and the attribute certificates, trusted CA of the user and other issuers, and the signature of the use condition certificate issuer.
- An *Attribute Certificate* consists of: an attribute, value, subject (User) and it's trusted CA, issuer and its trusted CA, and the signature of the attribute certificate issuer.
- A *Policy* at the resource level consists of information on: where to obtain a use condition certificate, where to look for attribute certificates, list of allowable use condition issuers, and where to look for identity certificates for user's identity and issuer's identity. There is also a root policy that consists of all the above in addition to trusted CAs and their public keys. It refers to the root authority of all the system resources.

4.3 *Authorization*

The resources that Akenti controls may be information, processing or communication capabilities, and physical systems, by accessing a network-based front-end. Access can be the ability to obtain information from the resource (as in "read" access), to modify the resource (as in "write" access), or cause that resource to perform certain functions (as in changing instrument control set points). Authorization is required to check whether the user is allowed to access a resource within the server that is owned by a stakeholder (policy maker). The policy makers are responsible for designing policies at the resource level. The use condition issuer, to whom the policy maker delegates authority, is responsible for issuing use conditions on that resource. Attribute certificates are issued by the attribute issuers as mentioned in the use condition certificate. During the authorization phase, attribute certificates are searched for a list of attributes and values the user has to satisfy according to the use-conditions set on that resource. If there are attribute certificates available for the user for the above attribute, value list, then the access information from the use condition is obtained which either allows access to the user or forbids the user from accessing any information on the server side.

5.0 Anchor Security

5.1 *Integration of Akenti and Anchor*

We use the IAIK-SSL[10] toolkit along with Java Cryptography Extensions (JCE[10,11]) for encrypted SSL communication between Anchor servers. A mutual authentication between Anchor servers is involved. The identity of the server is presented as an X.509

certificate, which gives information about the Certificate Authority who signed the certificate, and the identity for whom it is signed. If the Anchor server trusts the CA, it may accept the other server's identity with or without further restrictions.

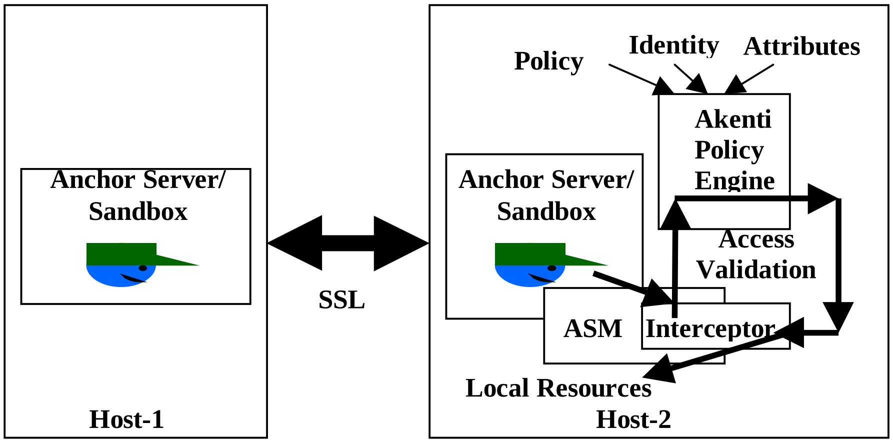


Figure 6: ***Akenti and Mobile Agents***

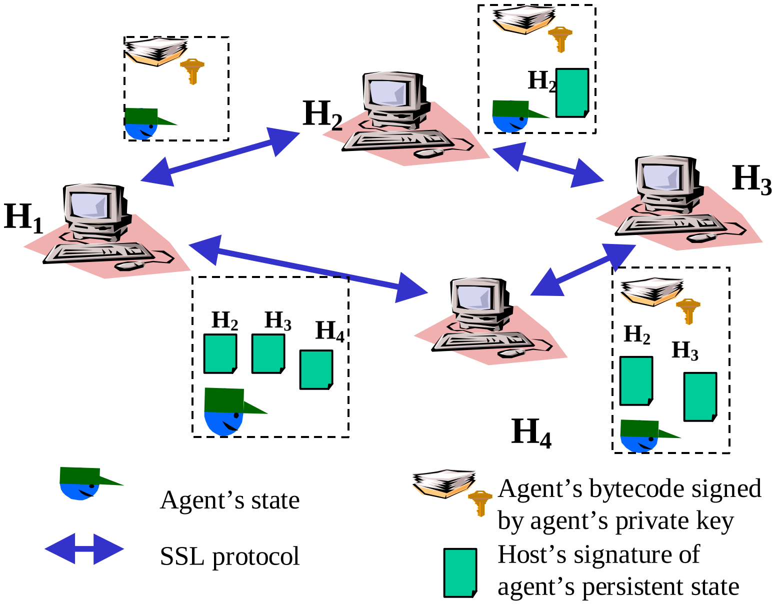
Java architecture facilitates the implementation of Anchor Toolkit's own security manager, by allowing Anchor to extend its security manager which controls access to resources at runtime at a very low level. Java's use of the "sandbox" security model provides the capability of separating agent servers from the agents that migrate into the server by the use of Class Loaders. The architecture permits the toolkit to implement its own class loader (ACL). A bytecode archive essential for agent's creation and execution is supplied after the agent is dispatched to reduce the network congestion. This loader is dedicated for loading the bytecodes of the agent.

Theoretically, it is feasible for agents to carry their private key. Practically it is infeasible and a tremendous exploratory work is being carried out in this area[12]. We realize the difficulties involved in this area and are awaiting a good solution. Anchor Toolkit provides a simplified and a secure access control model that defines five precepts for accomplishing the same. These are

- Agents are similar to applets with code and behavior. They must be trusted to communicate to their origin or go back to their origin without any issues.
- Mutual authentication among anchor servers,
- Agent authentication through signing of the bytecodes, and
- Authentication of agent hops or itineraries. Agents can carry the host's signature or communicate it back home for future reference. The host signs both the agent's code and behavior.
- Agents always come back to their origin for analyzing critical information.

Figure 7 explains the security model.

When trusted agent code has been allowed to enter the server, it is subjected to runtime restrictions from accessing resources, as it is not authorized for any access yet. The Anchor security manager enforces a particular security policy for each resource that can be accessed. The security manager can distinguish if sensitive actions are performed by the server or a foreign entity. If it is a foreign entity, it executes access control if the mobile agent is not authorized. The Anchor security manager overrides the methods of the Java security manager to invoke Akenti during runtime for \access control decisions. An access control request to Akenti is of the form <agent's DN, resource, operation> where agent's DN is its distinguished name and the operation is the action it tries to perform. For example, <"/C=US/O=ABC/CN=XXX", "file:/tmp/x.txt", "write">.



H₁
H₂
H₂
H₃
H₂
H₃
H₄
H₄
H₂
H₃
Agent's state
SSL protocol
Agent's bytecode signed by agent's private key
Host's signature of agent's persistent state

Figure 7: ***Anchor Security Model***

5.2 Eschewing Security Threats

The best case scenario provides with each party becoming liable for their acts.

Agent Protection –The encrypted channel between systems dispatching agents prevents any untrusted third party intimidation to agents. As a part of the protocol, hosts are required to sign the agent's persistent state before they are dispatched. This aids during analysis phase, to detect the location that lead to the abnormality of an agent's state.
Host Protection - Private resources of the hosts are protected from corruption and mishandling by Akenti. Akenti executes proper access control decisions for each trusted agent in order to determine their capabilities. A host allows agent execution only after it authenticates, the agent, the server that transmitted the agent and a list of hosts through which the agent traveled to attain its current state.

5.3 *Anchor Security Services*

Authentication - Mutual authentication between agent systems is established through SSL. The parties agree upon their identities before sharing any information between them. Agents are authenticated by signing their bytecodes with their private keys.
Integrity and Confidentiality - These features are supported as part of SSL.

Authorization - Access control is accomplished by the Akenti distributed access control system. This provides each agent with a set of capabilities on varied resources owned by the Anchor Server.
Non-repudiation - Anchor Servers and Agents are liable for their self-actions.
Auditing - Auditing service is available through Akenti. Akenti logs the resources being accessed by an agent and the monitor displays the same.

6.0 Related Work

Agent systems akin to Anchor exists, such as IBM's Aglets, ObjectSpace's Voyager[13], Mitsubishi's Concordia[14] etc. All agent systems provide similar features (some advanced) pertaining to mobility, messaging and transportation. Because of the diverse security policies, security models differ across systems. Aglets provides its own security manager and a policy tool for setting resource permissions. This restricts users from implementing their own security model. Agent protection is not assured because of the unencrypted communication channel. Anchor provides users with an option of implementing their security managers and also provides plug-ins (currently IAIK-SSL) for adding encrypted channel between anchor servers. Voyager and Concordia adopt a uniform username/password based access control for accessing system resources. This approach limits an agent system to a finite set of users. Anchor provides a fine-grained access control based on PKI through Akenti and hence offers an open system based on trust that is not limited to a finite set of users. Anchor supports a graphical tool in addition to standalone applications. Voyager does not support graphical tools for generating agents. Aglets invalidates an agent's proxy when it is dispatched from a location other than its origin. This baffles other agents in the system holding a reference to it, in the middle of their execution, which is not the case in the Anchor Toolkit. Anchor will support the naming and directory service using JNDI similar to Voyager. Anchor like Concordia, allows an agent to checkpoint itself to maintain the integrity of the system and provides a means to implement fault tolerance.

7.0 Future Work

A future version of this toolkit is likely to support an advanced message passing mechanism. Agent messaging using XML is favorable which eliminates the need of message objects to be available with every agent system and points the other agents to a DTD for this message in order to communicate with a particular agent. Currently, the toolkit is compatible with JDK 1.1.x. Future version is expected to be compatible with Java 1.2. Satp is the only protocol supported by the toolkit. This protocol has to be made transparent so that other distributed communication techniques available such as RMI and ORB services can also be incorporated. Agents have to be programmed meticulously. They can incapacitate the Anchor Server when an event queue is involved. A typical scenario would be an agent put into an indeterminate mode by an event arising out of user interaction. This can be avoided if agents are given independent event queues, which is an ongoing effort. Currently no caching mechanism is involved for bytecodes. This leads to the increased network traffic on transferring bytecodes every time an agent has to visit another host. Versioning needs to be implemented, if caching has to take place. In future, an efficient caching mechanism for caching bytecodes and a proficient versioning system detecting only bytecodes that differ from their previous version are expected.

8.0 Applications

8.1 *Secure Distributed Concurrent Version Control*

Concurrent version control system (CVS[15]) supports software development by a group of people working in a local or shared file system. It maintains a central repository shared by all the users and each user is isolated from other users and their work area cannot be interfered by others. It merges the work of each developer when it is done. It has built-in features that allow users to manipulate project files stored in a repository. The only constraint is that the user must be a part of the local network.

In a distributed environment, remote users may not be registered users of the local network, where the repository is located. Further, the software development by such users, if registered, has to be carried out locally. This imposes overhead on the system administration in terms of maintaining a remote user database, in addition to reduced performance due to sharing of computing resources.

In a collaborative computing environment, the need for a truly distributed CVS becomes significant, to improve productivity. In order to extend the conventional CVS to a distributed environment, the following issues need to be addressed:

- Remote system administration
- Synchronized operations between end systems
- Database (user/project) and application transparency
- Remote user authentication and access control for shared database
- Fault tolerance
- Load sharing to improve performance
- Efficient use of available resources

The ability of agents to co-exist and communicate seamlessly, collaborate and migrate, in addition to their autonomous nature makes them an ideal tool for distributed heterogeneous systems.

The agent system for CVS consists of a Master Agent (MA), Master Proxy Agent (MPA), Resource Discovery Agent (RDA), several User (UA), User Proxy Agents (UPA), and Representative Agents (RA). A brief description of the agents and their tasks is summarized in Table 1.

Figure 8 describes a distributed CVS. Two situations are discussed one in which the user requested resource is available locally and the other in which it is available remotely. Here, u1 and u3 refer to the CVS users, and r1 and r3 refer to resources in the main CVS repositories. When u1 requests a local resource, the request is interpreted by MPA. It checks with the MA if the resource is available and if so, whether the user has access permissions for that resource. If the MA authenticates the user and if the user has access permissions for that resource, it registers the user in its registry and creates a UA for him. The UA then becomes responsible for carrying out user requested tasks locally. Otherwise user's request is rejected. If the UA already exists for a user, the MPA simply forwards the user commands to that agent. This situation reflects a normal CVS with access permissions.

Table 1: ***Agents and their tasks***

Agents	Tasks	Mobility	Cloning	Description
Master Agent (MA)	Maintains common repository access permissions and performs access control, creates checkpoints, creates MPA.	No	No	Safeguards CVS Main/Central repository
Master Proxy Agent (MPA)	Interprets agent requests, monitors agents, creates UA, and RDA, maintains user access permissions and performs access control, Maintain both local and remote user agents registry.	No	No	Agents command Request interpreter and agent brokering with local and remote agents.
Viewer Agent (VA)	Presents users with unique views of the repository based on their capabilities.	No	No	Repository viewer creating a unique view for the user, based on the access permissions the user contains to different resources.
Resource Discovery Agent (RDA)	Search for requested resource in a list of hosts specified	Yes	Yes	Resource discoverer for shared CVS repositories in distributed systems.
User Agent (UA)	Creates user proxy agent, monitors user's activities, synchronize actions with its proxy, creates checkpoints, memorize recent task executed, activates on user events, responsible for local task execution.	No	No	Represents users in a distributed system.
User Proxy Agent (UPA)	Responsible for migrating across hosts to carry out tasks, communicates results back to user agent about events occurring in a remote host, creates checkpoint.	Yes	Yes	Proxy acts on behalf of user agents representing it remotely.
Representative Agent (RA)	Re-broadcast messages from MA to individual remote user agents.	Yes	No	Messenger for broadcast messages from MA.

In the second situation, u3 requests resource r2. The MPA interprets the request and checks with the MA for the resource availability. If the resource is unavailable, the MA passes referrals of those resources to the MPA. The MPA creates a RDA and provides the itinerary of hosts referred. The RDA creates clones to be dispatched simultaneously to locate the resource in different hosts. The RDA clones interact with the MA's of other hosts to find out the resource availability. The clones report to the main RDA, which in turn reports to the MPA about the host where the resource was found. The MA keeps track of where the resource was found for future reference. The UA's create proxies in order to execute remote CVS commands. If the user request involves remote CVS execution, the UA dispatches its proxy to the remote host to carry out the commands. The UPA and UA synchronize their operations. As a result, data transmission is handled by both at either host in order to make necessary updates to the user's work area. A secure transport layer can be used to authenticate remote users and maintain confidentiality and integrity among hosts.

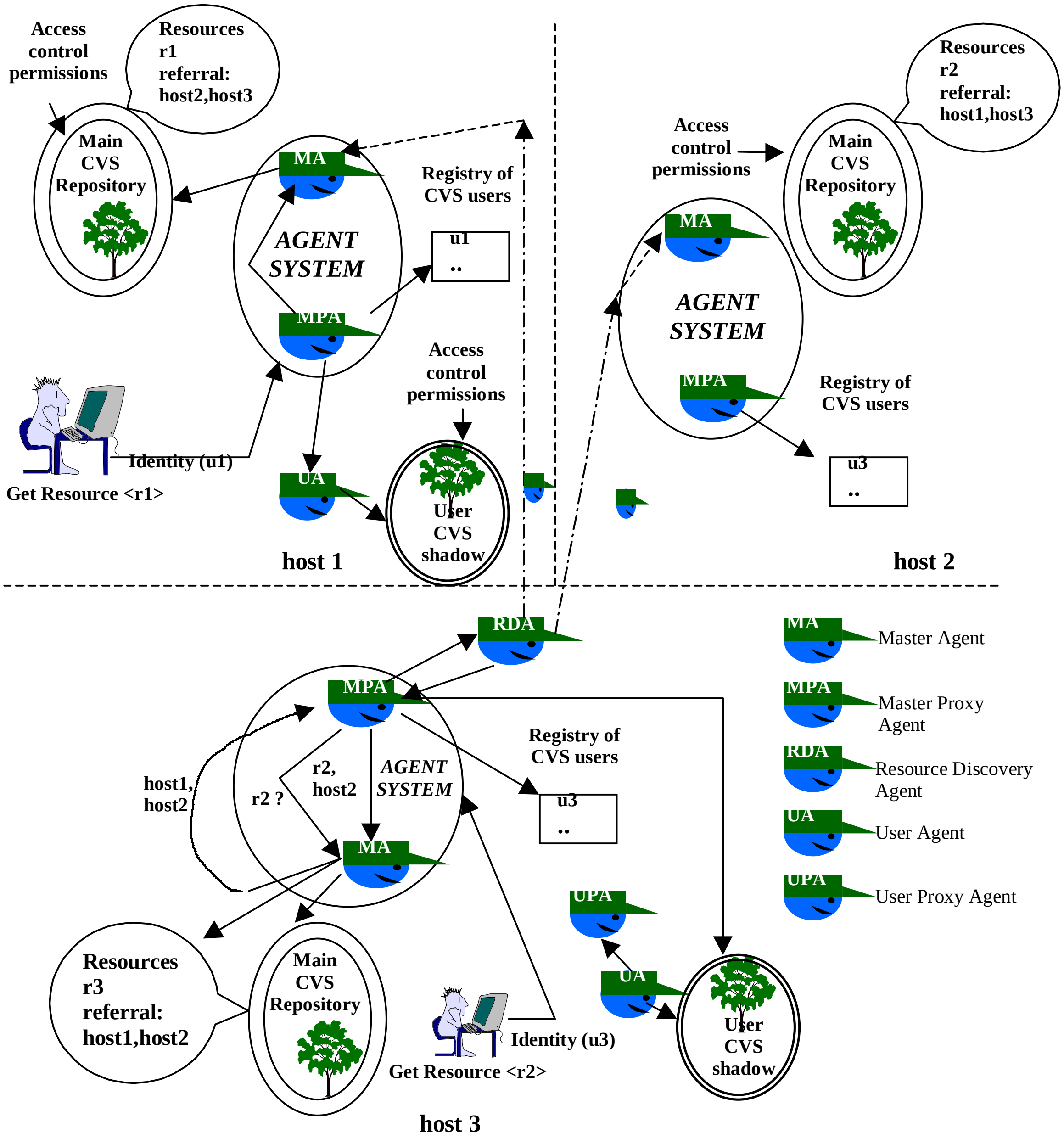


Figure 8: ***Distributed CVS***

All user agents are required to report to the MA on the completion of their tasks. There are possibilities for data losses during agent transactions, incomplete or unattended agent tasks. Fault tolerant activities have to be carried out in order to recover from such losses. These activities include periodic check-pointing to recover agents, continuous monitoring of agents to keep them alive, generating clones if task- accomplishing agents are at remote places. The system is made robust by keeping the MA and MPA as lightweight agents. These agents are primarily used for supervision and not for compute intensive processing. The MA's can have mirror agents as a backup for continuous availability of their services in case of failure. These MA's monitor each other. The MA is always event triggered by other agents. It can be deactivated by the agent system and reactivated whenever required. Since agent systems hide the complexities involved in managing underlying remotely distributed systems, the repository and the application itself are location transparent. The user's host contributes towards load sharing by handling local user operations and by storing the user's work area.

As for additional features to the CVS itself, the UA can monitor the user's activities and remind the user of those activities when necessary. For example, the user may add or delete files in the work area. Users may try to commit before informing the main repository of those additions and deletions. The UA can remind the user of the changes made in the work area. Another feature is to maintain and perform access permissions on the user's work area. If two users work on different modules and want to update each other's work area, it can be made possible with their access permissions. For example, if one user works on a GUI front-end and another user works on implementing the backend features and if both of their works are mutually exclusive, they can allow each other to update their work area. MPA's can create a viewer agent (VA) that when requested for a view from a user, consults with the MA to obtain access rights for that user on the entire repository and creates a view based on these resources. For geographically and physically isolated networks, a repository dump can be duplicated in order to collaborate with isolated networks. Finally, the MPA's can create representative agents for each registered user. Because the MA is aware of the initiation and completion of agent tasks, it can broadcast messages to all the agents about any updates in the central repository. The representative agents can carry the broadcast message to their respective users. Agents can also handle sequencing of CVS operations for intelligent database manipulations to restore dependencies.

9.0 Conclusion

This paper discussed using mobile agents with the Anchor toolkit. The architecture and APIs that are currently supported by this toolkit were elaborated. The access control model used for authorized access was explained. The secure transmission of agents and access control using Java SSL and Akenti based on Public Key Infrastructure (PKI) credentials was analyzed. The ability to retrofit security managers in Anchor was compared to other existing agent systems. Currently the toolkit provides barebones for a mobile agent framework. The applicability of this framework was envisioned in the context of a secure distributed CVS, which can be extended to analogous applications.

Our future work aims at providing more features to this toolkit and make it user-friendly and deployable by others.

References

[1] Software Agents
Jeffrey M. Bradshaw
AAAI Press/The MIT Press, Copyright © 1997, American Association for Artificial Intelligence.

[2] Introduction to SSL
http://developer.netscape.com/docs/manuals/security/sslin/

[3] Design and Implementation Issues of a Distributed Access Control System [to appear]
Srilekha Mudumbai, William Johnston, Mary Thompson, Gary Hoo, Keith Jackson and Abdeliah Essiari

[4] "Authorization and Attribute Certificates for Widely Distributed Access Control"
IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises - WETICE '98
William Johnston, Srilekha Mudumbai and Mary Thompson.

[5] Applied Cryptography, *Bruce Schneier, Second Ed. John Wiley & Sons, Inc.1996*

[6] Netscape Certificate Server
http://home.netscape.com/cms/v4.0/index.html

[7] LDAP - Programming Directory-Enabled Application with Lightweight Directory Access Protocol, *Timothy A. Howes, Mark C. Smith, McMillian Technical Publishing, Indianapolis, In. 1997*

[8] Programming and Deploying JAVA MOBILE AGENTS with Aglets
Danny B. Lange, Mitsuru Oshima
ADDISON-WESLEY, Copyright © 1998.

[9] "Internet X.509 Public Key Infrastructure Certificate and CRL Profile".
R. Housley, W. Ford, W. Polk,D. Solo
http://www.ietf.org/internet-drafts/draft-ietf-pkix-ipki-part1-11.txt

[10] IAIK-SSL, IAIK-JCE
http://jcewww.iaik.tu-graz.ac.at/JavaSecurity/index.htm

[11] Java Network Security
Robert Macgregor, Dave Durbin, John Owlett, Andrew Yeomans
Prentice Hall Publications, Copyright © 1998, International Business Machines Corp.

[12] Towards Mobile Cryptography
Tomas Sander and Christian F. Tschudin, ICSI

[13] ObjectSpace Voyager
http://www.objectspace.com

[14] Mitsubishi's Concordia
http://www.metica.com/HSL/Projects/Concordia

[15] Version Management with CVS, *Per Cederqvist et al*