CSE300-2 Distributed Object Computing
Final Project Report
7 December 2000

Topic: Security for Distributed Resource Environments

Group Members:
Charles E. Phillips, Jr. (Chip)
Qi Jin (Qi)
Jae-guon Nam (Jae-goun)
Zhenlin Qian (Jason)

COVER SHEET
Security for Distributed Resource Environments

Prof. S. A. Demurjian
Z. Qian, J. Nam, Q. Jin, and C. Phillips
Computer Science & Engineering Dept.
The University of Connecticut
Storrs, CT 06269-3155
steve@engr.uconn.edu
Tel: 860.486.4818
Fax: 860.486.4817

1. Introduction and Motivation.

There are several distributed resource technologies that can assist in the integration of distributed assets to improve on distributed processing needs. Familiar names like: CORBA, DCE, DCOM, Enterprise Java Beans, and more are very promising because they can provide needed interoperability with almost unlimited resources, hardware and software. Where they all have been lacking, is in consideration of security. Who can use what, when, where, and why? Three pillars of security mechanism and in particular role-based are: authorization, authentication, and enforcement. Authorization encompasses the privileges that are granted or revoked based on the client's role. Authentication is simply verifying the identity of the client. Enforcement is the mechanism that allows client access to only the services authorized [Demu 97]. There have been a number of related efforts to control access, authentication and authorization to resources based on role. Client authentication efforts using Kerberos [Neum94] have become popular and Cheron [Fox96] takes client authentication one-step further by considering agent authentication. CORBA offers some security capabilities, but not a total solution for role-based security [Yang96, Vino97]. We are interested in using these concepts to improve the security capabilities of a distributed resource environment by allowing resources to selectively and dynamically control who can access its services based on the role of the client.
We feel distributed applications and resources (hardware and software) should be as plug-and-play over a distributed environment as your local computer. Of course, like your computer, access needs to be controlled in a way that satisfies the user, but the user is not one person and user needs can only be satisfied in a distributed environment.

Distributed resource environments have lagged in support of security, providing minimal functionality to control the availability of a resource’s services to clients. In a distributed resource environment, different resources (hardware or software) are treated in a fashion that allows all clients and resources to be seamlessly integrated. Clients can consult a Lookup Service to locate and execute “services” on “found” resources to carry out their tasks. However, these environments lack security support. When a resource registers its services with the Lookup Service, there is currently no way for the resource to dictate which service can be utilized by which client [Demu 00]. The current solution for a resource to control access to its services is by changing the program. This, of course, is not an optimum solution.

Figure 1.1: General Architecture of Clients and Resources.
To address our concerns, we developed a general software architecture (Figure 1.1), a security client model (Figure 1.2) and two prototypes for integrating a role-based security into a distributed resource environment [Demu00]. This is the general architecture and security client model we will use to test our role-based security concepts. We will test this by integrating the Security Client with a general resource and then improve the architecture as necessary.

The initial prototypes were implemented on NT Platforms using Sun’s JINI technology (lookup service of choice), which promotes the construction and deployment of distributed applications and Java 1.2.1. In the beginning, the prototypes ran separately, one as a University Database System with simplified role-based security and the second was a stand alone, reusable, security client. The security client is designed for use as a security resource for any General Resource needing to enforce a dynamic role-based security policy.
This research effort seeks to improve on the previous prototypes by modifying code and implementing new security features. The result of our effort, of course, is to provide an improved security model and prototype that will support role-based security in a distributed resource environment. These are our objectives:

I. Prototype our security model by merging the University Database functionality with the reusable Security Client to provide a fully functional role-based security to the University Database System as a proof of concept.

II. Implement the use of different database management systems within our distributed resource environment. We are looking to incorporate Oracle into our environment with the immediate task of providing increased database functionality for the Security Client. We will initially use Access for the University database, but will eventually transition everything to Oracle to test all different configurations possible.

III. Implement the prototype security environment using different computer platforms and operating systems. Our intent is to use both UNIX and PC platforms, with the PC platforms configured with LINUX and NT operating systems. We feel this should provide adequate variety for our distributed environment prototype.

IV. Establish support of dual security clients running in the same distributed environment. We cannot accomplish this task using Microsoft Access, so incorporating the use of Oracle is critical. Oracle does have the capability to manage concurrent updates and dual-home databases.

V. Explore JINI Leasing capabilities and implement a leasing enforcement mechanism that will enhance the security environment. We will consider using both JINI and an
improvement to our Security Registration Resource. The modification to the Security Registration Resource that will allow for a processing window based on "to" and "from" dates.

VI. Create and implement a negative-privilege mechanism as part of the security client that will be used by a security officer to prevent the use of a specific privilege normally associated with the user's role. This mechanism is desired so a user-role can be customized with respect to the specific user.

VII. Each group member will survey noteworthy works in a distributed security related area. This, along with the efforts to modify the current prototype, should prove valuable for the development of an improved security model.

The remainder of this paper is organized as follows. Section 2 is a brief description of JINI and its capabilities as a lookup server. Section 3 is a description of the architecture and modifications necessary to meet our objectives. Section 4 reviews our current prototyping effort. Section 5 compares our efforts to other noteworthy role-based security efforts. Section 6 is our conclusions and future work. I have included Appendix A which contains the student work breakdown, Appendix B with the security system database scheme, and Appendix C and D for UML Class Diagrams.

2. JINI

JINI allows for the construction of distributed applications by combining groups of users with groups of required resources [Arno99, Edwa99, JINI, JINIARCH, Wald99]. In JINI, the resources register services that represent the methods needed for use by clients. A Lookup
Service is provided by JINI, and operates as a clearinghouse for resources to register services and clients to find services. The registration of services occurs via a leasing mechanism. With leasing, the services of a resource can be registered with the Lookup Service for a fixed time period or forever (no expiration). The lease must be renewed by the resource prior to its expiration, or the service will become unavailable. From a security perspective, the lease that is given by a resource to its services is not client specific. Once leased, a service is available to all, even if the service was intended for a targeted client or group of clients. Our work in this paper seeks to address and overcome this limitation.

A distributed application constructed under the JINI framework is designed around the utilization of one or more Lookup Services, as shown in Figure 1.1. The Lookup Service is key, through which all interactions by resources and by clients must occur. Whenever resources leave the environment, the Lookup Service must adjust its registry and clients must be able to adjust. Resources must register all their services with the Lookup Service, hence, they need to be lookup service compatible. One limitation of this process is that once registered, a resource’s services are available to all clients; JINI does not provide the ability to have services registered that are only available to certain clients. Once the resources have registered their services, clients can discover and request services. Using Figure 2.1 [Demu 00], we will step through a service request (AddCourse()). First, AddCourse() must be registered or any request for that service will be terminated. If the client makes the request for the registered service (AddCourse), lookup Service will return a service proxy. The service proxy allows the client to invoke any or all of the methods defined within the service. Using the proxy, the client invokes the needed method(s) with the result returned to the client.
3. **Software Architecture and Improvements**

The priority for this project is to integrate the University Database System with the full Security Client. Completing the integration is key because the other planned enhancements rely on knowledge gained from the integration. The next priority is the implementation of a different database management system. We will use Oracle because it is a robust database management system that will also allow us to create and operate dual security clients and at the same time use a different computing platform, which assists in accomplishing three of our objectives. Oracle has capabilities to run dual security clients (concurrent updates) and is currently resident on an available Linux platform, which meets one of our objective requirements. After the prototype merge, the progression is to establish new databases using Oracle, then move our integrated system to the Linux platform (with Oracle), and finally to duplicate the security client and run simultaneously. As part of Objective 3, the databases (University and Security Client) and the integrated system should be tested for platform independence by running the integrated system.
using different platform combinations. The only exception is for the Security Client Database, which needs to stay in Oracle on the Linux platform due to MS Access limitations. Finally, the remaining objectives (leasing capabilities and negative privileges) will be implemented.

This section will continue with a logical flow of events that support the implementation of our objectives. We start with the prototype merge (Section 3.1), continue with Security Client database implementation (Section 3.2), move to dual security clients (Section 3.3), then discuss platform independence (Section 3.4), leasing enforcement (Section 3.5) and negative privileges (Section 3.6). This section will culminate in a new software security model for role-based security (Section 3.7).

3.1. Merge Prototypes.

The software architecture presented in Figure 1.1, contains a set of clients that utilize a set of security resources and Lookup Services that allow clients to find resources (and their services. There are also, three security-specific resources: Role-Based Privileges, Authorization List, and Security Registration. Figure 1.2 contains a depiction of a Security Client that was not yet fully implemented with the University Database prototype. The objective is to implement the University Database System as the general resource in Figure 1.2. This will provide a complete security mechanism to the University Database System. Figure 3.1 depicts the proposed interactions between the Security Client and the Database Resource (University Database System) once the merge is completed.
In order to make this merge happen, we decided to create an interface that both the Security Client and the General Resource would use to communicate. This interface is depicted in Figure 3.2. The interface allows the merge between the existing security client code and the University Database with only minor modification to both and no loss of functionality. The University Database was modified to eliminate its limited security client. The University Database Security mechanism was logically similar to the Security Client, but did not have the ability to dynamically allocate roles or change roles.

We incorporate the use of a security token to build our resource interface. The Token will represent the client (user_id and user_role) when interfacing with the security mechanism. This token is generated each time a valid user enters the system. The Token for the security clients is good for only a limited duration and will end the current secession at that time limit. This is an added security feature to ensure limited access to the security control mechanisms.
A security token was devised to establish valid communication between the Security Client and the resource because it is a simple implementation. As long as the Security System passes that valid token to the client, the interactions required by the client will be accepted. As one can see in Figure 3.2, the protocol between security, client, and resource require that token. The token itself is a randomly generated token of type long integer (eight bytes).

To facilitate the merge of the original Security Client and the General Resource, the three Security Client Resources were combined into one Security System. There is no logical difference between the original Security Client and the new Security System, but it was determined easier to work with one resource rather than manipulating separate resources. We still feel the security resources should be deployed separately. An organization may have different security requirements as systems grow and it could be of great benefit or necessity to run the resources on separate machines.

Figure 3.2 Model for Security System Integration
With the new model for system integration in mind (Figure 3.2), we have created two new security clients that can be invoked depending on user role. They are the Enforce Policy Client and the Security Policy Client. This is the same concept used with the University Database System (Figure 3.3). Separate clients were constructed for the Student Role (Client 1) and the Faculty Role (Client 2) and each client controls access to the available methods from the DBServer Service. Depending on the User Identification, Password, and Role, a GUI will be provided for that client's particular needs. With respect to the security system, there is a policy and enforcement GUI. The basic premise is that the security officer may have jurisdiction over the authorization of an individual or client for a role, but have no authority over the policy regarding roles and what services a role requires. An organization may want one individual to serve in both roles; in that case, that individual would be allowed to hold both roles and could sign in using either role. The separation of privileges for policy making and policy enforcement are primarily along the lines of the Role-based Privileges Resource and Authorization List Resource (Figure 3.4). Referring back to our Model for Security System Integration (Figure
3.2), the new security enforcement and policy clients would act as the Resource Client and the Security System becomes the Resource Server.

![The Services and Methods for Security Resources.](image)

**SECURITY POLICY**

- **Role-Based Privileges Services**
  - Register Service
    - Register_Resource(R_Id)
    - Register_Service(R_Id, S_Id)
    - Register_Method(R_Id, S_Id, M_Id)
    - UnRegister_Resource(R_Id)
    - UnRegister_Service(R_Id, S_Id)
    - UnRegister_Method(R_Id, S_Id, M_Id)
  - Query Privileges Service
    - Check_Privileges(UR_Id, R_Id, S_Id, M_Id)
  - Grant-Revoke Service
    - Grant_Resource(UR_Id, R_Id)
    - Grant_Service(UR_Id, R_Id, S_Id)
    - Grant_Method(UR_Id, R_Id, S_Id, M_Id)
    - Revoke_Resource(UR_Id, R_Id)
    - Revoke_Service(UR_Id, R_Id, S_Id)
    - Revoke_Method(UR_Id, R_Id, S_Id, M_Id)
  - User Role Service
    - Create_New_Role(UR_Name, UR_Disc, UR_Id)
    - Delete_Role(UR_Id)
    - Find_Role(UR_Id)
    - Find_Roles(UR_Id)

**POLICY ENFORCEMENT**

- **Authorization-List Services**
  - Client Profile Service
    - Create_New_Client(C_Id)
    - Delete_Client(C_Id)
    - Find_Client(C_Id)
    - Find_All_Clients()
  - Authorize Role Service
    - Grant_SR_Client(UR_Id, C_Id)
    - Revoke_SR_Client(UR_Id, C_Id)
    - Find_SR_Client(UR_Id, C_Id)
    - Find_All_SR_Clients()

- **Security Registration Services**
  - Register Client Service
    - Register_Client(C_Id)
    - UnRegister_Client(C_Id)
    - Find_Client(C_Id)
    - Find_All_Clients()
  - Verify_Client(UR_Id, C_Id)
  - Find_All_Clients(UR_Id)
  - Find_All_Clients(UR_Id)

- **Query Privileges Service**
  - Check_Privileges(UR_Id, R_Id, S_Id, M_Id)
  - Grant-Revoke Service
    - Grant_Resource(UR_Id, R_Id)
    - Grant_Service(UR_Id, R_Id, S_Id)
    - Grant_Method(UR_Id, R_Id, S_Id, M_Id)
    - Revoke_Service(UR_Id, R_Id, S_Id)
    - Revoke_Method(UR_Id, R_Id, S_Id, M_Id)
  - Find_AllUR_Resource(R_Id)
  - Find_AllUR_Service(R_Id, S_Id)
  - Find_AllUR_Method(R_Id, S_Id, M_Id)

- **User Role Service**
  - Create_New_Role(UR_Name, UR_Disc, UR_Id)
  - Delete_Role(UR_Id)
  - Find_Role(UR_Id)
  - Find_Roles(UR_Id)

**3.2. Security Client Database**

Implementing our system using a different database management system serves more than one purpose. Our primary consideration was to make our security system flexible with respect to user needs and requirements by providing multiple platform functionality. Our solution should be adaptable to the distributed environment available and not be tied to certain platforms or operating systems. Another consideration for the database change is increased functionality. In the initial configuration, it is not possible to operate concurrent security clients. Redundancy in case of failure, or for a redistribution of processing power during peak access times is an important consideration when planning reliable and, in this case, secure systems. Microsoft Access cannot handle dual security clients and has less capability than other database products, such as Oracle. Operating more than one Security Client adds complexity to our system, but we
feel this is necessary for providing a survivable security client. Other considerations for changing database management systems are transaction management, database updates, and distributing access during peak processing times. These are potential problems that cannot be ignored, so we will consider them as part of our software system development process. Changing database management systems does not change the concept of using a lookup service to facilitate role-based security, but it will enhance the capabilities of our system both operationally and with respect to security. The goal for this objective is to incorporate Oracle as the database management system for both the Security Client and the General Resource (University Database System). A secondary goal will be to implement this system by using Oracle for the Security Client and Microsoft Access as the University Database repository. The remainder of this section summarizes the changes made to the databases to incorporate the system changes (3.2.1) and the database architecture that supports security (3.2.2).

3.2.1. Database changes.

There were no major changes to our database caused by moving from MS Access to Oracle. As one would expect, most changes and additions came about from programmatic changes due to additional requirements. A database scheme is found in Figure B.1 of Appendix B. The following tables were added to support negative privileges: USER_NP_METHOD, USER_NP_SERVICE, USER_NP_RES. In additions, a new table was required to support the use of the token (TOKEN) and IP addresses (IP) as part of our security checking. There were data elements added to USERS and RES (resources) tables to accept begin and end dates and to
the Token to handle creation time and duration. These changes were all required facilitating the added functionality of the Security System.

3.2.2. Security Client Database Architecture

In the initial Security Client prototype, MS Access is used as the database server. To fully deploy this system, we needed to switch database management systems because MS Access cannot satisfy our needs. MS Access is designed mainly as a single-user application, so it often gets corrupted when multiple users update the system at the same time in a distributed environment because MS Access does not manage transactions. MS Access also cannot support the replication of the database, so it also cannot support for the survivability of the system. This is not to say MS Access cannot be used because we did use it for the initial prototype, it simply has limited capabilities. For these reasons, we decide to use Oracle as our database server. Oracle has proven stability, especially in distributed environment and Oracle has two efficient ways to support the system’s survivability, Advanced Replication and Parallel Server.

For the single-security-client environment or in the multiple-security-client environment where all security clients share the same database, survivability of the security enforcement mechanism is very important. An Oracle Server provides two different technologies for accomplishing survivability: the Oracle Parallel Server and the Advanced Replication Facility. The Oracle Parallel Server supports fail-over to surviving systems when a system supporting an instance of the Oracle Server fails. This solution, however, can only provide the survivability support for instance fail-over. If the Oracle database server is down or disconnected, the
information will not be available. To add survivability support for this failure, we decide to use the second technology, Advanced Replication and Parallel Server (Figure 3.5). We can use the advanced replication facility to maintain a replica of a database at another database server. Oracle provides two advanced replication configurations: one is multi-master configuration and the other is updatable-snapshot configuration. In multi-master replications, all the sites act as equal peers and can operate on any object on any servers. In the updatable-snapshot configuration, one master site can consolidate the information that applications update at snapshot sites and propagate to all the other snapshot sites. Fast refresh can be configured and scheduled to reduce the network traffic. We selected the second configuration. One master database can be used and several snapshot servers can be backup for the master database. It is not necessary for the database servers to be equal peers, as long as their domain can be serviced. The multi-master replication is probably the best way to ensure backup coverage in our environment, but this can be very resource intensive and our resources are limited.

![Figure 3.5. Dual Security Client Database Architecture](image)

Figure 3.5. Dual Security Client Database Architecture
3.3. Dual Security Clients

There are really two issues when discussing dual Security Clients. The first issue is ensuring there is a backup Security Client (3.3.1) available, which is similar to the database redundancy issue. The second issue is more of a Multiple Security Clients policy issue (3.3.2).

3.3.1. Backup Security Client.

The successful operation of dual security clients as depicted in Figure 3.5 is very important to the security environment. Dual clients give the system built in redundancy in the case of partial system failure, provide for database back up in the event of destruction or communications failure, and can provide the system with a capability to distribute work load during peak processing times. This, of course, implies dual homing of resources. The goal for this objective is to successfully implement identical security clients on different platforms which update each other and will automatically assume Security Client responsibility if one or the other is disabled.

In a multiple-security-client environment where each security client has its own database (in large distributed environment, it is unavoidable), the survivability of each security client might not be enough. Keeping the consistency of data in different security clients becomes very crucial. Since security clients must be equal peers, we configure a multi-master replicated environment to solve this problem. In this environment, you can have any number of databases replicating each other. Users can perform any operations on any replicated site and the change
will automatically be updated to all other database. Oracle provides several system-defined
conflict resolution methods to keep data consistency. This solution can satisfy our needs for
keeping the data consistency among the security clients.

3.3.2. Multiple Security Client Policy.

Security Client Policy can get complicated if there is more then one Security Client in the
same environment and different security officers or policy manages them. This, of course is not
the case in our system, but could be a problem if one Security Client has down time and the
backup Security Client is managed by a different standard. Certainly, a Security Client is
necessary if the system requires this kind of protection, but what is the authority of the backup
Security Client? In future work, we will work on a policy to manage multiple Security Clients
and their interactions. In our prototype, we manage identical Security Clients and assume the
same security manager and policy.

3.4. Platform Independence

With flexibility in mind, we implement our system using different computing platforms.
Depending on the scale of the distributed environment, a personal computer may not have the
processing power to manage all of the resources for the General Resource or efficiently run the
Security System configured to enforce the user’s security policy. Our intent is show that
different computer platforms and operating systems can be used to effectively adapt the system
to different needs. Our goal for this objective is to configure a system using a personal computer
running Linux, a personal computer running NT, and a Sun Ultra5 running Unix to run the University Database system. We will limit the Security Client Database to Oracle on a Linux platform (for our immediate objective) because of limited resources, but the General Resource will be implemented in different configurations using Unix and NT platforms.

3.5. Leasing Enforcement

Leasing is a capability of JINI that enables a system to police itself without administrator interference. System resources are kept up to date or the lease will expire making a resource inaccessible. This is a concept that can also be leveraged for security as needless information can be deleted or made inaccessible without administrator assistance. We wanted our security system to have some control over time restrictions, so we implemented a feature based on time windows. When a resource is registered, the Security Client, will require a date time window (to/from date) in which this resource can be used. For example, the University Database System allows students to register for courses. The policy for registration limits enrollment after a certain day of the semester. The security resource will take the to/from date into consideration before making this function available. We accomplished this feature by adding functionality to the Security Registration Resource and by running the hasClientRight( ) method to be covered in the next section.
3.6. Negative Privileges

The concept of negative privileges is introduced into this prototype. In the previous version of our prototype only denotes positive privileges. This means that if the privilege is present in the security policy, the said privilege is granted. The denial of one permission is expressed by either the absence of said permission or the presence of a negative permission. Restricting oneself to only positive permissions has benefits for simplicity and good performance, but does not allow one to specify policy conveniently.

Negative privileges are the privileges that would normally be associated with a role, but because of a client specific issue, policy requires the specific client to be denied access to a Resource, service or method. This is not a trivial problem and we have only implemented a partial solution. In our prototype, we have added a separate negative privilege database to track these privileges. We handle negative privileges similar to granting privileges to a role, except the negative privileges are awarded to the specific client and not the role. We have decided to create an extra table to hold the privileges because eliminating the privilege from the role could possibly effect another client unnecessarily. Having an additional table also makes it easier for the security officer to manage negative privileges. In the event a privilege needs to be re-established, it simply needs to be eliminated from the list, instead of trying to find the privileges that is missing. The disadvantage of using an additional database table is the increased processing required to check authorization against yet another table.
Figure 3.6 is a summary of the steps taken to ensure authorized use of a resource. Before checking a client’s rights, a client must commit to authentication. After authentication, the user is assigned a token that represents the User ID and User Role. With every request, the client will go through the following sequence of events. First, invoke method, hasClientRight(long token, String resourceID, String Method ID). The second, check is on the token and user. Each token is initiated with a time restriction, which must be valid in order to continue. Next, there is a check on the resource. Each resource also has a valid time restriction (to from date) that could limit its availability. The negative privileges for that client are then checked. If the resource, service, or method is assigned a negative privilege, activity will be denied. Finally, the positive privileges will be checked. Again, positive privileges are those privileges assigned to a client base on role. The method hasClientRight( ) will return the appropriate Boolean.

![Diagram]

**Figure 3.6 hasClientRight( ) Method**
3.7 Security Model Improvements.

We have made a few changes to the original security model described in Section 1, Figure 1.1. First we combined the three resources (Role-based Privileges, Security Registration, and Authorization List). This was for convenience as it required fewer interfaces with the lookup service, but was still adequate to test our concepts. We created two separate security clients to handle policy and enforcement issues separately. We feel the security officer’s primary responsibility is to enforce policy and can possibly have no say in the policy itself. We separated the functionality for that reason. In some cases it is reasonable to think the security officer can create policy, so in that case both clients would be made available. The incorporation of dual security clients does not change the functionality of the security system, but does allow for redundancy, which will ensure a more reliable security system.

We added four other security features that will give added enforcement and management control. One security feature is the time limit imposed on secessions. This is accomplished through the issuance of the token with a time stamp. This limit will prevent clients from maintaining unlimited, continuous access to resources. Two, the token itself, being a randomly generated number, is also helpful in preventing the success of snoop programs by ensuring the same token is not used again within a reasonable amount of time. The third feature is the time restrictions imposed on both clients and resources. When a client and a resource register, a to-from time window is assigned. This window will be adjusted based on the need (need to know) of the client and the availability of the resource. Policy, of course, can also come into play. The
fourth and final security feature is negative privileges. A more detailed discussion is found in Section 3.6. It is difficult to capture all these features in a single graphic, but Figure 3.7 is an attempt to capture some of these improvements.

![Figure 3.7: New Architecture of Clients and Resources.](image)

4. Experimental Prototype

4.1. Authentication.

When logging on to the University Database system or Security System, this is the first interface presented. The user, of course, needs to enter their user identification, password, and the role for that secession. Keep in mind an individual can hold more than one role. The user
cannot change roles in mid secession. Figure 4.1. Depicts the GUI presented at login. This is called the Authentication GUI.

![Authentication GUI](image)

**Figure 4.1. Authentication GUI.**

### 4.2. Security Client.

As stated earlier, we have chosen to divide the security responsibilities into policy and enforcement categories. The two clients available are appropriately named Security Policy and Security Enforcement. The only way to access either of the security clients is to be authorized access by role. The following sections will describe the two new security clients.

#### 4.2.1. Security Policy.

The Security Policy Client is designed with the functionality to realize the organization’s security policy. The functionality that is reflected on the Role-based Policy Client GUI, Figure 4.2 is the same functionality described earlier in Figure 3.4. The format of this interface is different than the interface used with the initial prototype. The initial prototype used radio
buttons to indicate user preference; this version uses a tab-folder method because it was easier to adapt to change.

The policy client is divided into two parts, for role issues and resource issues. Under the Role division, a security manager can Create Role, Grant Resource, Grant Service, Grant Method, Grant IP, and query. Each specific task has a unique GUI that describes the task and the necessary information for executing that task. Figures 4.2 – 6 are representative of the policy enforcement tasks.

![Policy Client, Role, Create Role](image)

Figure 4.2. Policy Client, Role, Create Role
Figures 4.2 and 4.3 refer to the Role division of the Policy Client, Figures 4.4 to 4.6 refer to the Resource division of the Policy Client. In the Resource division, the following interfaces are available. Resource, Service, Method, Add Method to Service, IP, and Query.

Figure 4.4. Policy Client, Resource, Method
4.2.2. Security Enforcement.

The Security Enforcement Client is designed with the functionality to enforce the organization’s security policy. The functionality that is reflected on the Role-based Enforcement Client GUI, Figure 4.7 is the same functionality described earlier in Figure 3.4. The format of this interface is the same as used in the previous section for the Security Policy.
The Role-based Enforcement Client is divided into two parts, for User issues and Token issues. Under the User division, a security manager can Create User, Erase User, Grant Role, and query. Each specific task has a unique GUI that describes the task and the necessary information for executing that task. Figures 4.7 – 4.10 are representative of the policy enforcement tasks.

Figure 4.7. Enforcement Client, User, Create User

Figure 4.8. Enforcement Client, User, Grant Role
The Role-based Enforcement Client is divided into two parts, for User issues and Token issues. Under the Token division, a security manager can Unregister Token and query.0 Each specific task has a unique GUI that describes the task and the necessary information for executing that task. Figure 4.8 is representative of the policy enforcement tasks for Token.
4.3. **University Database.**

The University Database System required some modification in order to make it compatible with the Security System and complete the merge. The first step was to write a routine so the University System could self-register with both the lookup service and the security system. It is or should be the responsibility of the service requesting access to the lookup service and the security system to ensure compatibility to those services. The next step in our merge process was to incorporate the model for integration discussed in Section 3.1 and depicted in Figure 3.2. Finally, the University Database GUIs were modified for user friendliness. Figures 4.11 to 4.13 step through the functions of the University Database. There are only seven functions (methods) available with the University Database System and they are the same functions depicted in Figure 3.3, The University Database Architecture. The Security Policy, enforced by the Security System determines what functions are available to individual users.

![Figure 4.11. University Database, Query Database](image)
The University Database System is broken down into three major divisions, Query Database, Update Course, and Register Courses. The Query Database division allows the functions depicted in Figure 4.11. Upon selection of a function and appropriate window will open inside the GUI to assist the user. The Update Course division is displayed in Figure 4.12 and allows a valid user to add, remove, and change courses. The third division of the University Database System is Register Courses, depicted in Figure 4.13. This division allows the user to register and drop courses.

The University Database System can run using any relational database system to hold its database, but only ORACLE and MS Access have been tested.
5. Related Work

In this section, we review related efforts and contrast their work with our effort. In the area of security policy and enforcement, there has been work on security filters and screens, header encryption, user-level authentication, IP encapsulation, key management protocols, secure sockets, and secure shell [Oppl97]. This work is important to our effort, since this technology must underlie a distributed resource environment to provide OS-level security. Likewise, there have been studies showing the vulnerability of browser and network programming software to domain name server, spoofing, and denial of service attacks [Kemm97]. Browsers and network programming software are cornerstones of any distributed application, and their vulnerability will increase the difficulty in attaining security within a distributed resource environment.

Encryption is important to network security, and has been used for role identification, delegation of access control, securing a communication channel, and establishing a trusted computer base [Lamp92]. These problems are directly relevant to our work, since clients use service proxies for remotely located resources, and the clients must employ communications
channels and have their request processed by a computer whose trust is unknown. In the authentication area, Kerberos, is an authentication service for computer networks that uses encryption to verify both the user and that the message received is the message sent [Nuem94]. Another network protocol, Charon, which uses Kerberos techniques, provides a secure proxy service between client and server services [Fox96]. Using security capabilities from Charon and Kerberos in conjunction with a JINI Lookup Service, could be a major step in insuring the utilization of registered services and assisting in providing secure transmission paths for those services.

If clients in our approach are mobile agents, then it is important that we attempt to leverage work in the security of mobile agents and mobile objects to support our work. There have been several attempts to authenticate objects as they move through the network, including: Saga, a security architecture that controls access to and monitors the security of mobile agents via access tokens, access control vectors, and a security monitor [Sosh97]; Legion, a scalable security policy based on security related functions, where an object may or may not need security protection based its class [Wulf96]; and, Concordia, a framework for developing and executing mobile agent security that offers agent storage protection, transmission protection, and server resource protection [Wall98]. In addition, there are a growing list of efforts that insure that mobile agents are protected, namely: Trust Appraisal [Swar97], Metric Analysis [Reit99], SDSI (Simple Distributed Security) [Rive96], DCM (Distributed Compartment Model) [Gree96] and Short-lived Certificates [Hsu98]. Finally, there is the Seamless Object Authentication (SOA) approach, developed in CORBA and JAVA, which manages the signature of a CORBA client
and ACL with a security server [Tezu00]. This effort mirrors our own by examining the realization of security for a distributed architecture.

6. Conclusions and Future Work.

6.1 Conclusions.

In this paper, we attacked objectives designed to realize a role-based solution to security in a distributed resource environment, concentrating on authorization, authentication, and enforcement. We were successful in integrating a security client system with a University Database System using Java 1.3 and the JINI (1.3) lookup service. This system now has a dynamic way to enforce security policy on the University Database System users and change policy as necessary. In addition, we were able to improve our security system flexibility and increase its usefulness by moving code to different computing platforms with different operating systems and incorporating a more robust database management system, ORACLE. Taking advantage of ORACLE capabilities also enabled us to improve the survivability of our security system by creating concurrently running backup security system resources in case of failure. The use of ORACLE, along with faster machines made marked improvements to system response. Another successful implementation is negative privileges. The security enforcement mechanism can now recognize a user with a particular role as having additional limitations (negative privileges) placed on the particular user, without adjusting the role requirements set by the security policy. We found using the JINI lookup service leasing capabilities to control distributed resources was not adequate. Consequently, we developed our own mechanisms as
part of the security system to manage resources. Our solution requires a to-from time stamp on each resource as part of the registration process. We also used this same concept with user registration to ensure a temporary user does not maintain indefinite privileges. Finally, we captured the work completed in a new security model that can be the basis of future work.

The UML Class Diagrams for the University Database System and Security System are captured in Appendix C and Appendix D.

6.2 Future Work.

The work presented in this paper represents another plateau as we climb towards the goal of a true role-based solution to security in a distributed resource environment. Currently, there is one doctoral and three masters students working on this effort, which will continue, as student interest is significant. The following are a number of issues that need to be investigated as this project grows.

- The distributed application has clients and resources executing on different nodes in a potentially complex network. Does the location of a client affect the services that may be invoked? Does the location of a resource determine which services are made available to whom? Perhaps intranet and Internet activity should be viewed differently [Demu00].

- We have implemented the capability to operate multiple security clients. Unless all security clients are identical and all security folks manage the exact same way, there is a potential
problem if one security client is needed as a back up to a failed system. Exploring a hierarchical approach to managing multiple security clients could prove valuable.

- Resources in JINI are stateless in their ability to track a computation over time. When a client invokes a service and obtains a result, that invocation is independent from one that the client will invoke at a later point in time. Thus, each client invocation must include some identifier, which the resource can identify dynamically and determine if the invocation should occur. We will attempt to develop an analysis tool to follow invocations as they occur and use this tool not only to test our security system, but determine a means to solve this chaining invocation problem [Demu00].

- Mandatory Access Control is an issue with the military. Classification of documents and data can limit access of clients based role (need to know) and the individual’s clearance to view information, where the over riding control is classification. This problem is critical to the military as they work to solve a difficult problem called the Dynamic Coalition Problem. This problem, in short, is how to handle security in a dynamic, real-world, environment where one mission requires working with forces (countries) that, in another mission scenario are forbidden.

- Another interesting concept that warrants attention is ‘parameter restrictions’. This is a concept where dynamic restrictions can be put on parameters. Example: An individual can work in an organization and is authorized to review salaries of other individuals in the department. However, policy says the individual making the query can do this only if they have a higher rank. We could use parameters, reflecting rules set in policy, to restrict access or the availability of this type of information. (check_salary(my_rank > their_rank, return_value).
Appendix A. Topic Breakdown by Student

A.1. Charles E. Phillips, Jr. (Chip)

Chip was the coordinator of the group; responsible for making sure course requirements are met and progress was continuous. He also conduct research with respect to tracking chaining resource invocations explore the use of negative permissions, and was responsible for changes to the security model. He directed improvements to the initial security model to meet the objectives of this semester’s work and reflect goals discussed earlier, along with setting the path for future work.

A.2. Jae-guon Nam (Jae-guon)

Jae-guon implemented the current security model. He was the primary architect in the University Database/Security Client merge. The merge required Jae-guon critically analyze the existing code and determine how the University Database, without its version of security, could be modified with no loss of functionality. Our concept for the integration is depicted in Figure 3.3. Notice that we will try to incorporate a security token that will represent the client when interfacing with the security mechanism. He also put a secession time restriction on the token to prevent unlimited secession length. These changes were attempted using the existing Security Client with as little modification as possible to promote its reusability for other applications. In doing this, he also changed to a different GUI interface.

Jae-guon developed and implemented two clients to support the security system (Role-based Security Policy Client and Role-based Security Enforcement Client. This was done because we felt a clear distinction of policy and enforcement was necessary.

Jae-guon also did the leg work involved in transitioning our development platforms. Unfortunately this did not go real well at times, but with help we figured it out.

A.3. Qi Jin (Qi)

Qi re-implemented the security system using Oracle on a Linux platform, and she established dual security clients running in the same distributed environment, but on different platforms. This was not trivial objective because they required the modification of the source code to reference a different database management system and required mastery of Oracle capabilities in order to properly configure the concurrent databases.

Qi also re-wrote the University Database system with specifications dictated by Jae-guon’s security system. This includes changing the GUI interface.

Qi wrote a routine that will automatically register the University Database with the lookup service and make the services available to the security system.

A.4. Zhenlin Qian (Jason)
Jason was responsible for the exploration of Jini’s Leasing capabilities and implementing a leasing enforcement mechanism that will enhance the security environment. His initial push was to learn JINI and take advantage of JINI capabilities with respect to leasing. This did not work out as well as we had liked. JINI does not offer a capability to set a lease based on a ‘from’ and ‘to’ date. We felt this capability was necessary for limiting access to resources that should not be available except for specific time frames, so Jason worked on an implementation of this concept. Jason had also worked on establishing a Proxy-List of approved resources for a client to use during an active secession. This is a security and system performance related feature that would eliminate redundant Security Client checks and eliminate the Proxy-List at the end of the secession. We decided to abandon this concept and concentrated on “negative privileges” as an objective.
Figure B.1. Security System Database Scheme.
Appendix C. UML Class Diagram for the University Database System

Figure C.1. Database Client UML
Figure C.2. Database Server UML
Appendix D. UML Class Diagram for the New Security System.

Figure D.1. UML Class Diagram for the University Database System
References