Role Delegation for a Distributed, Unified RBAC/MAC[†]

M. Liebrand and H. Ellis
Dept. of Engineering and Science
275 Windsor Street
Rensselaer at Hartford
Hartford, CT 06120-2991
mark.liebrand@snet.net, heidic@rh.edu
860.548.5387, fax: 860.547.0868

C. Phillips, S. Demurjian and T.C. Ting
Dept. of Computer Science & Engineering
191 Auditorium Road
The Univ. of Connecticut,
Storrs, CT, 06269-3155
{charlesp, steve, ting}@engr.uconn.edu,
860.486.3719, fax: 860.486.4817

Abstract

The day-to-day operations of corporations and government agencies rely on inter-operating legacy, COTs, databases, clients, servers, etc., which are brought together into a distributed environment running middleware (e.g., CORBA, JINI, DCOM, etc.). Both access control and security assurance within these distributed applications is paramount. Of particular concern is the *delegation of authority*, where an authorized individual (not the security officer) may delegate all or part of his/her authority to another individual, increasing security risk. The ability of an authorized individual to operate in a manner akin to a security officer, without oversight, must be carefully considered. This paper explores the definition and inclusion of role delegation into an existing distributed, unified role-based/mandatory access-control (RBAC/MAC) security model and enforcement framework. The RBAC/MAC model/framework controls access to software APIs to limit, by role, which users (clients) can access which parts of APIs, constrained by time, classification, and data values. This paper uses the RBAC/MAC model/enforcement framework as a context for a detailed examination of role delegation, including: the general characteristics of role delegation; the incorporation of role-delegation into the RBAC/MAC security model; and, the impact of role delegation on security assurance at design and run times.

1 INTRODUCTION

The assembling of legacy, COTs, databases, clients, servers, etc., into meaningful distributed applications via middleware (e.g., CORBA, JINI, DCOM, etc.) has emerged as a norm as companies and government agencies seek to allow existing software artifacts to inter-operate with new artifacts, as illustrated in the top portion of Figure 1. The security capabilities embodied within these distributed applications must enforce the security policy for all users and protect sensitive information from access and misuse [DoD88, Sand98]. We must be concerned with both controlling the access of individual users and the interactions among users. In the latter case, there has been increased attention on the *delegation of authority*, where an authorized individual (not the security officer) may delegate all or part of his/her authority to another individual, increasing security risk, and raising interesting security assurance implications [Bark00, Lin99, Na00, Zhan01]. Large organizations often require delegation to meet demands on individuals in specific roles for certain periods of time. In fact, this work grew out of the actual needs of the lead author, who also works as an information systems specialist for a large financial management/insurance company, where delegation is critical for a number of in-house applications and processes. The main objective of this paper is to examine role delegation, proposing a means to allow individuals to delegate roles within security policy guidelines, while simultaneously maintaining security assurance at run time.

In support of this objective, we leverage our ongoing research on a unified role-based/mandatory access control (RBAC/MAC) security model and enforcement framework for a distributed environment

[†] Support for the work in this paper was provided in part by the GE Foundation through a grant to the School of Engineering at the University of Connecticut.

comprised of software resources (interacting via middleware) that has been under design and development for the past three years [Demu01, Phil02a, Phil02b], based on our prior research on security for object-oriented systems [Demu97]. Our approach concentrates on the APIs of software resources, providing the means for them to be customizable and restricted by time intervals, data values, and clearance levels to support: RBAC – which portions of APIs can be invoked based on the responsibilities of a user role; and, MAC – only access portions of APIs based on the security level of the user. The associated security administrative tools and enforcement framework for this research is shown in the bottom half of Figure 1. The enforcement framework, the Unified Security Resource (USR), consists of three sets of services: *Security Policy Services* to manage roles and their privileges; *Security Authorization Services* to authorize roles to users; and, *Security Registration Services* to identify clients and track security behavior. The USR is a repository for all static and dynamic security information on roles, clients, resources, authorizations, etc. Figure 1 also depicts the *Security Policy Client (SPC)* to manage URs by granting/revoking privileges and setting classification (CLS) levels, and the *Security Authorization Client (SAC)* to assign clearances (CLRs) and authorize roles to end users.

The major focus of this paper is to extend the RBAC/MAC security model and enforcement framework (Figure 1) with role delegation, allowing a security officer to assign delegation authority at design time, which can then be enforced, at run time. Role delegation will need to adhere to the same rules already in place for RBAC and MAC, but the security model must be expanded to support delegation concepts at design time, and to incorporate delegation and its enforcement into the run-time environment. A secondary focus of this paper is to detail the attainment of security assurance, at design time via SPC, SAC, and a *Security Delegation Client (SDC)* to grant, update, and revoke delegations (see Figure 1), and at run time by extensions to the enforcement framework (USR) to support delegation. To address these two foci, Section 2 presents a unified RBAC/MAC security model, as background material. Section 3 examines role delegation, detailing extensions to the RBAC/MAC model and enforcement framework to support delegation and revocation [Zhan01], and analyzing our approach against delegation characteristics (e.g., monotonicity, permanence, totality, etc.) [Bark00]. Section 4 discusses security assurance of the security model/enforcement framework in the presence of role delegation. Conclusions and future work are reviewed in Section 5.

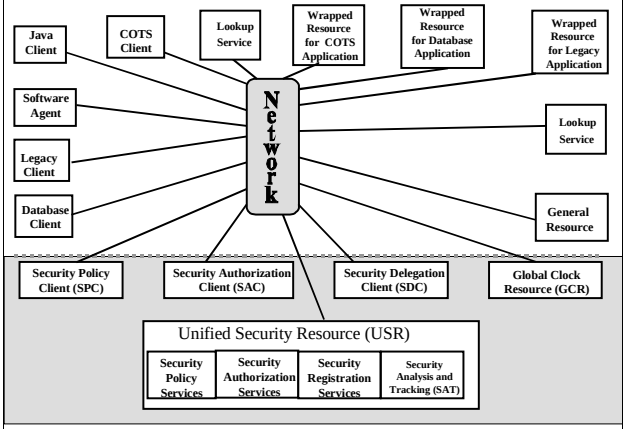

Figure 1: Clients/Resources in a Distributed Environment.

Joint Service with Methods:	**a.k.a**
(S)Weather (Token);	METOC
(S)VideoTeleconference (Token, fromOrg, toOrg);	TLCF
(S)JointOperationsPlannning (Token, CrisisNum);	JOPES
(S)CrisisPicture (Token, CrisisNum, Grid1, Grid2);	COP
(S)TransportationFlow (Token);	JFAST
(S)LogisticsPlanningTool (Token, CrisisNum);	LOGSAFE
(S)DefenseMessageSystem (Token);	DMS
(T)NATOMessageSystem (Token);	CRONOS
Component Service with Methods:	
(S)ArmyBattleCommandSys (Token, CrisisNum);	ABCS
(S)AirForceBattleManagementSys (Token, CrisisNum);	TBMCS
(S)MarineCombatOpnsSys (Token, CrisisNum);	TCO
(S)NavyCommandSystem (Token, CrisisNum);	JMCIS

Note: Access Classification Precedes Each Entry.

Figure 2: A GCCS Resource with Two Services.

2 BACKGROUND: A DISTRIBUTED, UNIFIED RBAC/MAC SECURITY MODEL

This section reviews the definitions for the security model introduced in [Demu01] and formalized in [Phil02a, Phil02b]. Note that all of the definitions have been reorganized and renumbered from [Phil02b]. We begin the discussion with two definitions for the concepts of lifetimes and security levels, as follows:

Definition 1: A *lifetime, LT,* is defined as a discrete time interval with start time (st) and end time (et), denoted [st, et], that an entity is available for use, where et > st, and st or et is of the form (month, day, year, hour, minute, second). Operations on LTs X and Y are as follows:

- X $\triangleright$ Y means that Y's LT is within X's LT ($Y.st \geq X.st$ and $Y.et \leq X.et$)
- X $\triangleleft$ Y is equivalent to Y $\triangleright$ X
- Let $ST = \max\{X.st, Y.st\}$ *and* $ET = \min\{X.et, Y.et\}$. Then

  $$X \cap Y = \begin{cases} \emptyset & \text{if } ET \leq ST \quad (1.1) \\ [ST, ET] & \text{if } ET > ST \quad (1.2) \end{cases}$$

  Note that intersection can be generalized to multiple operands.
- LT = [ct, ∞] means from the current time (ct) onward.

Note that in Definition 1 for intersection, we are defining a null overlap when the start time of Y is equal to or after the end time of X (1.1), with 1.2 representing the case where the LT of Y overlaps the LT of X.

Definition 2: The important concepts that underlie mandatory access control are as follows:

- A *sensitivity level*, *SLEVEL*, a set of values that represent the degree of impact on security by the exposure of information which has a particular level. For our purposes, $SLEVEL = \{U, C, S, T\}$ where: unclassified (U) or no impact; confidential (C), expected to cause some damage; secret (S), expected to cause serious damage; and top secret (T) expected to cause exceptionally grave damage [Exec82].
- SLEVELs form a hierarchy: U < C < S < T [Exec82].
- *Clearance (CLR)* is the SLEVEL given to users for access to information.
- *Classification (CLS)* is the SLEVEL given to entities (roles, objects, methods, etc.).

Note that while we are using four SLEVELs, any number will work with any hierarchy in our approach.

Definitions 3 through 6 are for a distributed application comprised of resources, services, and methods, utilizing both lifetimes (when a resource/service/method is available) and classifications (what is the SLEVEL of a resource/service/method).

Definition 3: A *distributed application*, *DAPPL*, is composed of a set of unique *software/system resources* (e.g., a legacy, COTS, DB, etc.), $R = \{R_i \mid i = 1 \ldots m\}$, each of which is composed of a set of unique *services*, $S_i = \{S_{ij} \mid j = 1 \ldots n_i\}$, each of which is composed of a set of unique *methods*, $M_{ij} = \{M_{ijk} \mid k = 1 \ldots q_{ij}\}$.

Definition 4: Every *method* M_{ijk}, *for some* $i = 1 \ldots m, j = 1 \ldots n_i, k = 1 \ldots q_{ij}$ of service S_{ij} of resource R_i is registered from a security perspective as: $M_{ijk} = [M_{ijk}^{Name}, M_{ijk}^{LT}, M_{ijk}^{CLS}, M_{ijk}^{Params}]$ where M_{ijk}^{Name} is the method name, M_{ijk}^{LT} is the security LT for which the method is available for use with default [ct, ∞], $M_{ijk}^{CLS} \in SLEVEL$ with default U, and M_{ijk}^{Params} is the list of parameter names and types.

Note that the Name, LT, CLS, and Params for each method are all set when the resource registers itself, its services, and their methods with the security middleware (USR in Figure 1).

Definition 5: Every *service* S_{ij}, *for some* $i = 1 \ldots m, j = 1 \ldots n_i$, of resource R_i is registered from a security perspective as: $S_{ij} = [S_{ij}^{Name}, S_{ij}^{LT}, S_{ij}^{CLS}]$ where S_{ij}^{Name} is the service name,

S_{ij}^{LT} is the security LT where $S_{ij}^{LT.st} \le \min\{M_{ijk}^{LT.st} \mid k = 1...q_{ij}\}$ and $S_{ij}^{LT.et} \ge \max\{M_{ijk}^{LT.et} \mid k = 1...q_{ij}\}$, and $S_{ij}^{CLS} = \min\{M_{ijk}^{CLS} \mid k = 1\ldots q_{ij}\}$.

Note that the Name for each service is set when the resource registers itself, its services, and their methods with the USR. The CLS and LT for each service are calculated as given in Definition 5, with CLS set as the minimum CLS of all methods of the service, and LT set with the earliest start time and the latest end time of its methods.

Definition 6: Every *resource* R_i, *for some* $i = 1\ldots m$, is registered from a security perspective as: $R_i = [R_i^{Name}, R_i^{LT}, R_i^{CLS}]$ where R_i^{Name} is the resource name, R_i^{LT} is the security LT where $R_i^{LT.st} \le \min\{S_{ij}^{LT.st} \mid j = 1...n_i\}$ and $R_i^{LT.et} \ge \max\{S_{ij}^{LT.et} \mid j = 1...n_i\}$, and $R_i^{CLS} = \min\{S_{ij}^{CLS} \mid j = 1\ldots n_i\}$.

Note that the Name for each resource is set when the resource registers itself, its services, and their methods with the USR. The CLS and LT for each resource are calculated as given in Definition 6, with CLS set as the minimum CLS of all services of the resource, and LT set with the earliest start time and the latest end time of its services. For illustrative purposes, we use the U.S. Global Command and Control System (GCCS), an automation tool that provides a U.S. commander with operational awareness of the situation (crisis) in near real-time through integrated sets of services. GCCS provides information-processing for planning, mobility, sustainment, and messaging, by bringing together 20 separate automated systems in over 625 locations worldwide [GCCS99] in a private network. Figure 2 contains a GCCS resource with two services, Joint and Component, for our examples.

The remaining set of definitions, 7 through 18, involves the privilege specification process for user roles against the resources, services, and methods. Definitions 7 and 8 involve user roles for a DAPPL.

Definition 7: A *user role, UR,* is a uniquely named entity representing a specific set of responsibilities against an application, and is defined as: $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$ where UR^{Name} is the role name, UR^{LT} is the role LT with default [ct, ∞], and $UR^{CLS} \in SLEVEL$ is the role classification level, with default U.

Note that the name, LT, and CLS of each user role is set by the security officer when designing and defining the security policy for a DAPPL.

Definition 8: A *user-role list,* $URL = \{UR_i \mid i = 1\ldots r\}$, is the set of *r* unique roles that have been defined for DAPPL, where each role is as specified in Definition 7.

Note that user role names cannot be reused with different lifetimes and/or different clearances. Representative user roles for GCCS could be Commander [CDR_CR1, [ct, ∞], T] and Joint Planner [JPlannerCR1, [01dec00, 01jun01, S], where [ct, ∞] for LT is the default. From a privilege perspective, URs will be granted access to resources, services, and methods, which have classification levels that are at or below the role's classification level. For the two GCCS services in Figure 2, CDR_CR1 may be granted access to both services or JPlannerCR1 to all methods from either service that have levels S, C, or U. Next, Definitions 9 and 10 involve users for a DAPPL.

Definition 9: A *user, U,* is a uniquely named entity who will be accessing the DAPPL via a client application, and is defined as: $U = [U^{UserId}, U^{LT}, U^{CLR}]$, where U^{UserId} is the user identifier, U^{LT} is the user LT with default [ct, ∞], and $U^{CLR} \in SLEVEL$ is the user clearance level with default U.

Note that the name, LT, and CLS of each user is set by the security officer when designing and defining the security policy for a DAPPL.

Definition 10: A *user list,* $UL = \{U_i \mid i = 1 \ldots u\}$, is the set of *u* users that have been defined for DAPPL, where each user is as specified in Definition 9.

To support information assurance of clients accessing methods based on role, three types of constraints are utilized to verify allowable values, time limits, and CLR/CLS. First, signature constraints limit user role access to methods based on allowable data values.

Definition 11: A *signature constraint, SC,* is a boolean expression defined on the signature of method M_{ijk} *for some* $i = 1 \ldots m, j = 1 \ldots n_i, k = 1 \ldots q_{ij}$, of a service S_{ij} of a resource R_i, to limit the allowable values on the parameters, M_{ijk}^{Params}. The SC is a boolean expression (with default "true") defined on parameter names and values (type specific) constructed with operators: AND, OR, and NOT.

Note that while it is called the "signature constraint," currently, the constraint only involves the parameters and does not yet involve the return values (subject of future work). SCs limit the conditions under which a method may be invoked. For example, an ArmyLogCR1 UR can use method CrisisPicture from the Joint Service, but needs an SC (Grid1 ≤ NA20 AND Grid2 ≤ NC40) to limit the view. Thus, methods are off/on based on a specialization of the parameter/return values. Second, a time constraint limits the execution based on when a user (playing a role) can execute a method.

Definition 12: A *time constraint, TC,* is a lifetime, and is defined to represent when a method can be assigned to a user role (or invoked by a user) or when a user is allowed to play a role. A TC has the default of [ct, ∞] which equates to: the user role and method LT constraining when the method can be assigned; the user role, method, and user LTs constraining when the method can be invoked; and the user role and user LT constraining when the user can be authorized to the role at design time and at runtime.

JPlannerCR1 has a TC on ArmyBattleCommandSys of [10dec00, 16feb01]. In the ArmyLogCR1 UR, we combine SC and TC to limit access to the LogPlanningTool method to a specified timeframe, for a specific crisis leading to SC: (CrisisNum = CR1), TC: [10dec00, 16feb01].

The third type of constraint is for MAC to support CLR and CLS. As discussed in Definition 4, 5, and 6, the CLS level is assigned to individual methods that comprise each service of a resource (see Figure 2), and on user roles (Definition 7) and users (Definition 9). By enforcing the relationship among CLR and CLS at design and run times, it is possible to realize MAC and the Bell and LaPadula Model [Bell75].

Definition 13: A *mandatory access control constraint, MACC,* is the *domination* of the SLEVEL of one entity (e.g., user, role) over another entity (e.g., role, method), which is the valid relationship between CLR and CLS. In some cases, we must check CLR ≥ CLS (user CLR vs. role CLS), in others, CLS ≥ CLS (role CLS vs. method CLS).

Since all resources, services, methods, and roles have CLSs, MACC can be utilized to properly compare subject (user) CLR to CLS and deny or accept based on MAC rules. Since a UR is assigned a CLS, the authorized user must possess a CLR greater than or equal to the role CLS. At run time, MACC verifies if the client (user with a CLR level) playing a role (with a CLS level) is allowed to invoke a specific method (with a CLS level) at a particular time.

The final set of definitions involves different types of authorizations, of method(s) to a user role, and of a user role to a user, which are captured in the following:

Definition 14: A *user-role authorization, URA,* signifies that a user role can be authorized to invoke a method under optional TC and/or SC, and is defined as: $URA = [UR, M, TC, SC]$,

where UR is as given in Definition 7, M is as given in Definition 4, TC is as given in Definition 12 and represents when (LT) the method is available to UR for invocation with default of [ct, ∞] and SC is empty (true) or as given in Definition 11 and represents the values under which the invocation can occur.

Definition 15a: For each DAPPL, there is a *UR authorization matrix, URAM*, an $r \times q$ matrix, where $q = \sum_{\substack{i=1..m \\ j=1..n_i}} q_{ij}$, indexed by roles and methods, with each entry defined as:

$$URAM(UR_i, M_j) = \begin{cases} 1 & UR_i \text{ is authorized to invoke } M_j \\ 0 & otherwise \end{cases}$$

Note that we assume that initially, *URAM*, contains all 0 entries. Note also when the value is equal to 1 for some $URA = [A, M, TC, SC]$, the authorization is called a *valid URA, VURA*. At design time, a VURA must satisfy the CLS domination of role over method and the overlap of TC and lifetimes to allow the matrix entry to be set to 1.

Definition 15b: A *valid user-role authorization list,* $VURAL = \{VURA_i \ \forall \ i = 1 \ldots v\}$, where $v \le r \times q$, is the set of all VURAs for which URAM(UR,M) = 1.

Definition 16: A *user authorization, UA*, signifies that a user is authorized to play a specific role and is defined as: $UA = [U, UR, TC]$, where U is as given in Definition 9, UR is as given in Definition 7, and TC is as given in Definition 12 and represents when (LT) the role is available for use by U with default of [ct, ∞].

Definition 17a: For each DAPPL, there is a *user authorization matrix, UAM*, an $r \times u$ matrix, indexed by roles and users, with each entry defined as:

$$UAM(UR_i, U_j) = \begin{cases} 1 & U_j \text{ is authorized to } UR_i \\ 0 & otherwise \end{cases}$$

Note that we assume that initially, *UAM*, contains all 0 entries. Note also when the value is equal to 1 for some $UA = [U, UR, TC]$, the authorization is called a *valid UA, VUA*. At design time, a VUA must satisfy the CLR/CLS domination of user over role and the overlap of TC and lifetimes to allow the matrix entry to be set to 1.

Definition 17b: A *valid user authorization list,* $VUAL = \{VUA_i \mid i = 1 \ldots w\}$, where $w \le r \times u$, is the set of all VUAs for which UAM(UR,U) = 1.

Definition 18: A *client, C*, represents an authorized user *U*, and is uniquely identified for each session via a *client token* C = [U, UR, IP-Address, Client-Creation-Time]. The Client-Creation-Time is the clock time at client startup.

Note that we will defer examples related to user and clients to Section 3.1, where they can be more complete and inclusive of role delegation concepts.

3 ROLE DELEGATION

Role delegation is a user-to-user relationship that allows one user to transfer responsibility for a particular role to another authorized individual, and can be classified as: *administratively-directed delegation*, where an administrative infrastructure outside the direct control of a user mediates delegation [Linn99]; and, *user-directed delegation* where an individual (playing a role) determines if and when to delegate responsibilities to another individual to perform the role's permissions [Na00]. User-directed delegation is situation specific. For example, suppose that a delegation is defined to allow a supervisor to delegate a

role to a subordinate. In practice, one supervisor may want to delegate the role to a subordinate while another supervisor may not. While subordinates may have the same official job function and permission, the authority is granted at the discretion of the supervisor; it is user directed. We have concentrated on user-directed delegation due to its interesting characteristics and challenges. User-directed delegation does not eliminate security administrators, who must continue to establish the security policy and maintain delegation authority, including who can do delegation at what times. User-directed delegation is not intended to take over complete control of the administration of the user-role relationship. When a user's function changes, whether it is to add privileges or revoke privileges, this belongs in an administrative infrastructure governed by policy, which will be set by an administrator. Administration of RBAC, MAC, and delegation must be carefully controlled to ensure that policy does not drift away from its original objective [Sand00].

The concept of delegation can cause some confusion. For example, when a user delegates their role, they can delegate: *authority, responsibility, or duty*. The authority, responsibility, and duty to perform a task are often used interchangeably when discussing delegation, but have different connotations. In most organizations, authority can be delegated, but responsibility can never be delegated. *Authority* to do the task, carries the least responsibility necessary to execute the task, but does mean the delegated user can execute the delegated task or role. *Responsibility* to do a task implies accountability and a vested interest that a task or role can be executed properly. The *duty* to perform a task implies that the delegated user is obligated to execute the given task. The focus of this section is the inclusion of delegation authority in our unified RBAC/MAC model and enforcement framework.

The remainder of this section investigates and analyzes the extensions to the unified RBAC/MAC security model and enforcement framework [Demu01, Phil02a, Phil02b], presented in Section 2, to support all aspects of role delegation. In Section 3.1, the focus is on security model extensions to support delegation. In Section 3.2, the emphasis is on the enforcement framework modifications, exploring the inclusion of role delegation and revocation rules [Zhan01]. Finally, in Section 3.3, the model extensions and framework modifications are analyzed against a set of delegation characteristics: monotonicity, permanence, totality, administration, levels of delegation, multiple delegation, agreements, and cascading revocation and grant-dependency revocation [Bark00].

3.1 Security Model Extensions for Role Delegation

In this section, we examine the RBAC/MAC security model extensions that are needed in support of role delegation. The extensions are necessary to support changes to the enforcement framework (see Section 3.2) in order to attain security assurance (see Section 4). First, we define the concept of *delegatable*:

Definition 19: A *delegatable UR, DUR,* is a $UR \in URL$ that is eligible for delegation.

Definition 20: The *delegatable UR vector, DURV,* is defined for all r *URs* $\in URL$ as:

$$DURV(UR_i) = \begin{cases} 1 & UR_i \text{ is a DUR} \\ 0 & UR_i \text{ is not a DUR} \end{cases}$$

Initially, all entries in *DURV* are set to 0. As a security officer is defining the security requirements for an application, whenever a role is designated as delegatable, then the relevant role entry is set to 1. Figures 3a and 3b contain a detailed example using GCCS, including U, UR, and VURAs. URs for Crisis 1 are: Commander [CDR_CR1, [ct, ∞], T]; Joint Planner [JPlannerCR1, [01dec00, 01jun01], S]; and Army Logistics Officer [ArmyLogCR1, [10dec00, 01mar01], S]. In Figures 3a and 3b, the roles CDR_CR1, JPlannerCR1, JPlannerCR2, are delegatable (respective DURV(UR) = 1) and ArmyLogCR1, and ArmyLogCR2 are not delegatable (respective DURV(UR) = 0).

The next three definitions are for the concepts of an *original user*, a *delegated user,* and a matrix that stores whether a user is original, delegated, or unauthorized for each role.

Definition 21: An *original user, OU∈ UL*, of a UR is defined as being authorized to the UR as part of the security policy definition (there exists a VUA for the OU/UR, i.e., UAM(UR,OU) = 1 and not as a result of a delegation).

Definition 22: A *delegated user, DU∈ UL*, is a user U who is eligible to be delegated a UR by an OU or a DU (there is not a VUA for the DU/UR, i.e., UAM(UR,DU) ≠1). Note that a DU of a UR cannot be an OU for that same UR.

Definition 23: The *user delegation/authorization matrix, UDAM*, is an $r \times u$ matrix indexed by roles and users, with each entry of the matrix is defined as:

$$UDAM(UR_i, U_j) = \begin{cases} 2 & U_j \text{ is a DU of } UR_i \quad (23.1) \\ 1 & U_j \text{ is an OU of } UR_i \quad (23.2) \\ 0 & U_j \text{ is not authorized to } UR_i \quad (23.3) \end{cases}$$

Initially, all entries in *UDAM* are set to 0. As users are authorized to roles via VUAs (see Definition 17a), then the relevant (user, role) entry is set to 1.

Users: $U = [U^{UserId}, U^{LT}, U^{CLR}]$

```
(T)General DoBest: [DoBest, [ct, " ],  T]
(T)Colonel DoGood: [DoGood, [01dec00,01jun01], T]
(S)Major DoRight:  [DoRight, [01dec00,01jan01], S]
(T)Major CanDoRight: [CanDoRight,[01jan01,01feb01, T]
```

User-Roles: $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$

```
[CDR_CR1, [01dec00,01dec01],  T]
[JPlannerCR1, [01dec00, 01jun01], S]
[JPlannerCR2, [01jul01, 01sep01], C]
[ArmyLogCR1, [10dec00, 01mar01], S]
[ArmyLogCR2, [01jul01, 01aug01], C]
```

User Role Authorizations: $URA = [UR, M, TC, SC]$

```
[JPlannerCR1, CrisisPicture, [ct, " ],true]
[JPlannerCR1, ArmyBattleCommandSys, [10dec00,16feb01], true]

[ArmyLogCR1, CrisisPicture, [10dec00,16feb01],
             Grid1 ≤ NA20 AND Grid2 ≤ NC40],
[ArmyLogCR1, LogPlanningTool, [10dec00,16feb01],CrisisNum=CR1]
```

Figure 3a: Sample Users, U; User-Roles, URs; and User-Role Authorizations, URA.

User Authorization Matrix (UAM): 1 = authorized, 0 = other

User\ User-Role	ArmyLogCR1	ArmyLogCR2	JPlannerCR1	JPlannerCR2	CDR_CR1
DoBest	0	0	0	0	1
DoGood	0	0	1	1	0
DoRight	1	0	0	0	0
CanDoRight	0	1	0	0	0

Delegation Authority Matrix (DAM): 2 = has DA and PODA, 1 = has DA, 0 = neither

User\ User-Role	ArmyLogCR1	ArmyLogCR2	JPlannerCR1	JPlannerCR2	CDR_CR1
DoBest	0	0	0	0	2
DoGood	0	0	1	1	0
DoRight	0	0	0	0	0
CanDoRight	0	0	0	0	0

User Delegation/Authorization Matrix (UDAM): 2 = U is a DU, 1 = U is a OU, and 0 = not authorized

User\ User-Role	ArmyLogCR1	ArmyLogCR2	JPlannerCR1	JPlannerCR2	CDR_CR1
DoBest	0	0	0	0	1
DoGood	0	0	1	1	0
DoRight	1	0	0	0	0
CanDoRight	0	1	0	0	0

User-Role Authorization Matrix (URAM): 1 = UR authorized to invoke Method, 0 = otherwise

Method\User-Role	ArmyLogCR1	ArmyLogCR2	JPlannerCR1	JPlannerCR2	CDR_CR1
ArmyBattleCommamdSys	1	1	1	1	1
CrisisPicture	1	1	1	1	1
MarineCombatOpnsSys	0	0	1	1	1
LogPlanningTool	1	1	0	0	1

Figure 3b: Sample UAM, URAM, UDAM, URAM.

The remaining three definitions for the model are critical to establish regular and pass-on delegation authority, which are needed to realize role delegation in the enforcement framework (Sections 3.2 and 4). Essentially, we are allowing, at design time, the ability to have delegation authority (able to delegate) or pass-on delegation authority (able to pass on the authority to delegate) for a given user of a role.

Definition 24: *Delegation authority, DA*, is the authority given to the OU to allow delegation of a DUR to another user.

Definition 25: *Pass-on delegation authority, PODA*, is the authority given to an OU or a DU to pass on the delegation authority for a DUR to another user (OU or DU).

Definition 26: The *delegation authority matrix, DAM*, is an $r \times u$ matrix indexed by roles and users, with each entry of the matrix is defined as:

$$DAM(UR_i, U_j) = \begin{cases} 2 & U_j \text{ has DA and PODA for } UR_i \quad (26.1) \\ 1 & U_j \text{ has only DA for } UR_i \quad (26.2) \\ 0 & U_j \text{ has neither DA nor PODA for } UR_i \quad (26.3) \end{cases}$$

Note that we assume that initially, *DAM*, contains all 0 entries. To illustrate the concepts of delegation definitions, in Figures 3a and 3b, there are URs: CDR_CR1, JPlannerCR1, and ArmyLogCR1. Figures 3a

and 3b also contain users, U: General DoBest, Colonel DoGood, Major DoRight, and Major CanDoRight with CLR levels and relevant authorization matrices.

Example 1: To illustrate delegation, suppose that General DoBest wishes to delegate his UR (CDR_CR1) to Colonel DoGood with delegation authority (DA), where DoBest, CDR_CR1, and DoGood are defined in Figures 3a and 3b.

This delegation can take place because General DoBest is an OU (UDAM(CDR_CR1, DoBest) = 1) of CDR_CR1, DoGood is not an OU or DU (UDAM(CDR_CR1, DoGood) = 0), the UR is delegatable (assume DURV(CDR_CR1) = 1), and Colonel DoGood holds the correct clearance ((CDR_CR1^{CLS} = T) ≤ (DoGood^{CLR}= T)). General DoBest can also grant delegation authority, DA, because he has pass-on delegation authority, PODA (DAM(CDR_CR1, DoBest) = 2). Note that Colonel DoGood can execute the UR: CDR_CR1, but is limited to the LT for Colonel DoGood. Also note, that the User Authorization Matrix, UAM(CDR_CR1, DoGood) is set to "1" (authorized), the User Delegation/Authorization Matrix, UDAM (CDR_CR1, DoGood) is set to "1" (delegated user, DU), the Delegation Authority Matrix, DAM(CDR_CR1, DoGood), is set to "1" (has delegation authority) and a Valid User Authorization, VUA = [DoGood, CDR_CR1, [ct, ∞]] is created (TC by default is [ct, ∞], but can be used to further constrain the delegation).

Example 2: Continuing from Example 1, suppose Colonel DoGood wishes to also delegate UR: CDR_CR1 to Major CanDoRight.

This delegation can take place because the role is delegatable (assume DURV(CDR_CR1) = 1), CanDoRight is not an OU or DU (UDAM(CDR_CR1, CanDoRight) = 0), Major CanDoRight does have the prerequisite clearance ((CDR_CR1^{CLS} = T) ≤ (CanDoRight^{CLR}= T)) and the delegation authority (DAM(CDR_CR1, DoGood) = 1) from Example 1. However, this delegation will be limited to the LT of Major CanDoRight (CanDoRight^{LT} = [01jan01, 01feb01]). Note, that the User Authorization Matrix, UAM(CDR_CR1, CanDoRight) is set to "1" (authorized), the User Delegation/Authorization Matrix, UDAM (CDR_CR1, DoGood) is set to "1" (delegated user, DU), the Delegation Authority Matrix, DAM(CDR_CR1, CanDoRight), remains "0" (has no delegation authority) and a Valid User Authorization, VUA = [CanDoRight, CDR_CR1,[ct, ∞]] is created (TC by default is [ct, ∞], but can further constrain the delegation).

3.2 Enforcement Framework Delegation and Revocation Rules

In addition to the RBAC/MAC security model extensions for role delegation in Section 3.1, there are also infrastructure-related extensions that are needed in support of the enforcement framework. In the definitions given in Section 3.1, the DU is only allowed permissions because of the OU, and if the OU delegates a user role, and then has that role revoked, the DU, will also lose the delegated role. This is an example of cascading deletes, which must be handled dynamically via an enforcement framework rather than by security administrative intervention. Thus, Definitions 19 to 26 are required to track the relationships between users and delegated roles, to maintain security assurance and reduce risk or compromise of the security policy.

To augment Definitions 19 to 26 and support role delegation in our enforcement framework (i.e., the Unified Security Resource and associated security tools in Figure 1), we employ a useful set of definitions and rules for delegation which underlie a proposed delegation language [Zhan01].

- *Original User* (OU), *Delegated User* (DU), *Delegation Authority* (DA), *Pass-On Delegation Authority* (PODA), and *Delegatable User Role* (DUR) as given in prior definitions.
- *Original User Delegation Relation* – the relation between an original user and a delegated user.
- *Delegated User Delegation Relation* – the relation between a delegated user and its delegated user.
- *Delegation Path* – a set of ordered user role assignments of OU to DU (to DU, etc.).

- *Prior* – a function that maps one user role relation to another user role relation, or to null if that user role relation is for an original user.
- *Revocation Authority* (RA) – the authority to revoke a delegation path, which can be only taken by the security administrator and the OU or the DU initiating the delegation.

Granting, revoking, and delegating user roles, while a simple process, has the potential to have a wide-ranging impact, particularly during revocation of delegated roles. The delegation and revocation rules for our enforcement framework are a simplified version of those proposed in [Zhan01]. The two main differences are: 1. a simplification of the rules with the PODA and DA matrices used to manage delegation depth rather than an integer; and 2. the prohibition of independent (by another OU) revocation, i.e., revocation only by the security officer. The rules that are being incorporated into our enforcement framework are:

- User-To-User Delegation Authority Rule: A user (OU or DU) who is a current member of a delegatable role (DUR), can delegate that user role to any user that meets the prerequisite conditions of the role: the DU receiving the role is not a member of the role; the OU or DU is identified as having delegation authority for the role; the DU meets the mandatory access control constraints (MACC); and the DU satisfies lifetime constraints.
- Delegation Revocation Authorization Rule: An original user (OU) can revoke any delegated user (DU) from a user role in which the OU executed the delegation. This is a stricter interpretation than [Zhan01], which allows any OU of a role revocation authority over a DU in the delegation path. In addition, a security administrator can revoke any delegation.
- Cascading Revocation Rule: Whenever an original user (OU) or delegated user (DU) in the delegation path is revoked, all DUs in the path are revoked.

These rules and definitions detail the critical run-time considerations of role delegation, that must be supported as part of the Unified Security Resource (see Figure 1), and at design time, incorporated into the security administrative tools to be discussed in Section 4.

3.3 Analyses of Role Delegation Capabilities

This section analyses the role-delegation extensions of the RBAC/MAC security model (Section 3.1) and enforcement framework (Section 3.2) against a number of different criteria, providing the opportunity to assess our work on role delegation versus the context of other research efforts. Specifically, we leverage the work of [Bark00] for a set of critical identifying characteristics of delegation including: monotonicity, permanence, totality, administration, levels of delegation, multiple delegation, agreements, cascading revocation, and grant-dependency revocation. We evaluate each of these delegation characteristics for incorporation into our security model.

- *Monotonicity (monotonic/non-monotonic)* refers to the state of control the original user (OU) possesses after role delegation. *Monotonic delegation* means that the OU maintains control of the delegated role. *Non-monotonic* means that the OU passes the control of the role to a delegated user (DU). In most real-world environments, the original user does not relinquish control of the role, and this is the approach we have taken in our security model and enforcement framework.
- *Permanence* refers to delegation in terms of time duration. Permanent delegation is when a delegated user (DU) permanently replaces the original user (OU). Temporary delegation has an associated time limit with each role. When the time limit passes, the DU no longer has that role. Temporal constraints are an important part of our security model, since limiting access is in concert with the concept of least privilege. We are incorporating temporary delegation into our security model by allowing the OU to set lifetimes for each role delegation and by allowing the OU to revoke the delegation at anytime (monotonicity).

- *Totality* refers to how completely the permissions assigned to the role are delegated. *Partial delegation* refers to the delegation of a subset of the permissions of the role. In *total delegation*, all of the permissions of the role are delegated. Barka and Sandhu [Bark00] note that partial delegation works best in a hierarchical RBAC model. Our position is that partial delegation defeats the purpose of creating roles and since our model is not hierarchical, we are implementing total delegation.
- *Administration refers* to how delegation will be administered. The two obvious alternatives to administration are user-directed and administrator-directed (third party, agent-directed) delegation. Administrator-directed delegation already exists in our security model to the extent that the security administrator currently assigns all privileges. Our enforcement framework is being extended to support user-directed delegation with revocation.
- *Levels of delegation* refers to the ability of a delegated user (DU) to further delegate a role (PODA) and the number of vertical levels the delegated role can be delegated. Barka and Sandhu [Bark00] identify a single step delegation where a DU would not be able to re-delegate and a multi-step delegation where a DU could re-delegate the role. [Zhan01] extended [Bark00] by describing three ways to control the depth or levels of delegation: *no control* – where roles can be re-delegated without limit; *boolean control* – where roles can be re-delegated until a delegating user says no; and *integer control* – where roles can be re-delegated until a certain number of re-delegations have occurred. In order to maintain control of delegation at the policy level, but still allow the original user (OU) some flexibility in delegating roles, we employ a *modified boolean control* and use the *Delegation Authority Matrix*, DAM, to control delegation. This matrix limits the levels of delegation by allowing only the OU to have delegation authority, DA, and pass-on delegation authority, PODA, which is consistent with our monotonicity approach, where the OU maintains control of the role. An OU can grant PODA to a DU, but a DU cannot pass on PODA again, there by limiting the delegation depth. This also reflects evaluation and information flow paradigms of the military and other large organizations, where a senior leader rates a subordinate two levels below and the subordinate is responsible for knowing the mission intent two levels above. The security policy will determine what OUs have DA and PODA and what roles can be delegated. The OU will have the option of allowing DA, DA and PODA or neither to a DU. A DU given DA can delegate the delegated role, but the DU cannot grant DA (PODA) to the next DU. We feel this delegation process will satisfy most organizations. Note that limiting who can have PODA enforces the two-level limit.
- *Multiple Delegations* refers to the number of delegated users (DU) (horizontally) to whom a delegatable user role (DUR) can be delegated to at any given time. We are including unlimited delegations in our security model since we want to maintain the user's flexibility. Cardinality within a role has been found not to be used [Awis97]; cardinality within a delegated role would also probably not be used. A limit on the number of DUs to a role, particularly when greater than one, is subjective. Subjective limits are not often enforced; there are no hard bases for them.
- *Agreements* refer to the delegation protocol of the original user (OU) to the delegated user (DU). There are two types of agreements, bilateral and unilateral [Bark00]. In *bilateral agreements*, the DU needs to accept the delegated role. In *unilateral agreements*, the OU delegates the user role permissions and the DUs are not required to accept or even acknowledge the delegation. Bilateral agreements make sense if the responsibilities of the role placed upon the DU require action (duty or responsibility) vs. just the ability to perform an action (authority). For example, if there is a task that needs to be done by everyone in a role on a monthly basis, then a bilateral agreement would make sense as the user should acknowledge that responsibility. However, that is more of an operational policy than security policy. In our model, all tasks given to a role are added capabilities and the operational actions required for those users are of operational concern, therefore unilateral agreement is being modeled and implemented.

- *Cascading Revocation* refers to the indirect revocation of all delegated users (DUs) when the original user (OU) revokes delegation or administration revokes the OU's delegated role. Non-cascading revocation could be useful in the event a middle manager user is fired without replacement and subordinates need to execute the vacated roles. However, having uncontrolled delegation is an unnecessary assurance risk, so this special case will be handled by security administration, but will not effect a cascading revocation policy. Our existing enforcement framework is being modified to support cascading revocation.
- *Grant-Dependency Revocation* refers to who has authority to revoke a delegated user (DU). In grant-dependent revocation, only the original user (OU) can revoke the delegated role. In grant-independent revocation, any original member of the DUR can revoke a delegated role. We are utilizing a limited form of grant-independent revocation where only the delegating user and the security administrator can revoke a DUR. The goal is to let the OU have some delegating authority, but still allow the security administrators to have final control. Allowing a second party OU to revoke a delegation of a fellow OU is not necessary as long as the security administrator maintains the revocation capability.

4 SECURITY ASSURANCE AND DELEGATION ENFORCEMENT

For the RBAC/MAC model of Section 2, extended for role delegation (Section 3), the ability of the resulting administrative tools and enforcement framework to support delegation with a degree of security assurance must be demonstrated. These checks must occur at both design and run times, so that the consistency of user roles, CLR/CLS levels, lifetimes, role delegations, and end-user authorizations, is meticulously verified whenever the security policy is changed, modified, or augmented. The main issue is to provide a detailed set of security assurance checks for role delegation that augment the ones described in [Phil02b], which did not include role delegation. There must be automatic alerts to the security officer when potential conflicts occur during the delegation process, thereby heading off possible inconsistencies. The intent of design and run-time assurance checks is to have a distributed application that provides strong confidence to both system administrators and users in the attainment of the RBAC/MAC security policy with role delegation.

4.1 Security Assurance at Design and Run Times

The administrative tools and enforcement framework for the RBAC/MAC security model and its role delegation extensions must support security assurance at design time (for security officers creating the security policy and for users establishing delegation) and at run time (for enforcing the delegation and revocation rules in Section 3.2 and the critical delegation concepts in Section 3.3). Design-time checks prevent illegal actions at definition, while run-time checks insure that the model and its concepts are enforced operationally. Design-time assurance checks are enumerated and discussed below:

1. **MACC Domination**: There is a MACC check when adding a role to a user. Recall that the user must dominate the role with respect to clearance vs. classification. This check has a direct impact on role delegation because MAC constraints cannot be violated.
2. **Role Delegation**: The security system checks to make sure the delegated user (DU) is not already a member of the delegated role before allowing a delegation to be defined. For example, if user X is assigned role B, and is attempting to delegate B to user Y, then for that delegation to be successful, user Y cannot be an OU or a DU of role B (UDAM(B,Y) = 0).
3. **User-To-User Delegation Authority:** Recall that a user (OU or DU) who is a current member of a delegatable role (DUR), can delegate that role to any user that meets the prerequisite conditions of the role: the DU receiving the role is not a member of the role; the OU or DU is identified as having DA for the role (DAM(UR,U) > 0); and, the DU meets the mandatory access control constraints (MACC). This rule is partially checked during definition.

4. **Lifetime Consistency:** In regard to the permanence criteria (Section 3.3), the lifetime of the DU be within the lifetime of the delegating user (OU or DU), verified at design time, as given in Definition 21.
5. **Modified Boolean Delegation:** In regard to the levels of delegation criteria (Section 3.3), the combination of pass-on delegation authority, PODA, and delegation authority, DA, used in the Delegation Authority Matrix, DAM, controls the levels of delegation. An OU can delegate – with optional DA and PODA, and the delegated user can only DA. This must be enforced at design time to insure more than two levels are prohibited. In order for a user to have PODA, UDAM(UR,U) = 1 (U is an OU) and DAM(UR,U) = 2 (U has DA and PODA) for the UR.

These five different assurance checks can be formally supported in assurance rules. A security assurance rule is a logical statement that must be satisfied in order for a status to be set (design-time check) or an action to be performed (run-time check). Rules I and II are for the assignment of DA and PODA to a user X at design time. We assume that a user must have DA in order to have PODA (e.g., a user cannot have PODA without DA).

Rule I: Let $X \in UL$ be a user (Definition 9) and $A \in URL$ be a user role (Definition 7).

X can have DA for A ($DAM(A, X) = 1$) iff
$UDAM(A, X) = 1 (X \ an \ OU), DURV(A) = 1 (A \ a \ DUR), and \ \exists \ a \ VUA = [X, A, TC].$

Note that UDAM(A, X) =1 implies that UAM(A, X) = 1 (see Definition 17a).

Rule II: Let $X \in UL$ be a user (Definition 9) and $A \in URL$ be a user role (Definition 7).

X can have DA and PODA for A ($DAM(X, A) = 2$) iff
$UDAM(A, X) = 1 (X \ an \ OU), DURV(A) = 1 (A \ a \ DUR), and \ \exists \ a \ VUA = [X, A, TC].$

Note that UDAM(A, X) =1 implies that UAM(A, X) = 1 (see Definition 17a). Note also that Rules I and II, establish, respectively, DA or DA/PODA for user X to role A in the case where X is an OU. Rules I and II lead to Rule III, for the delegation by a user of a role to another user, which is relevant at design time. The delegation sets UAM and UDAM for the delegated user and delegated role.

Rule III: Let $X \in UL$ be a user (Definition 9) and $A \in URL$ be a user role (Definition 7), such that $DAM(A, X) \geq 1$ (Rules I or II).

X can delegate A to user Y limited by TC ($UAM(A,Y) = 1$, $UDAM(A,Y) = 2$, and VUA = [Y, A, TC]) iff

- Case 1: $UDAM(A,Y) \neq 1$, $Y^{CLR} \geq A^{CLS}$, $TC \lhd (Y^{LT} \cap A^{LT}) \neq \emptyset$, and $TC.et > ct$ when TC $\neq \emptyset$.
- Case 2: $UDAM(A,Y) \neq 1$, $Y^{CLR} \geq A^{CLS}$, $Q = (Y^{LT} \cap A^{LT}) \neq \emptyset$, and $Q.et > ct$ when TC = $\emptyset$. Set TC = Q.

For either case, VUAL = VUAL $\cup$ UA, where $UA = [Y, A, TC]$ is the *valid UA, VUA*, for the delegation of A to Y.

There are two cases for Rule III, based on whether there is a TC given or if the TC is null. In either case, the condition checks that A is not an OU of Y, and Y's CLR dominates A's CLS. For Case 1, the given TC must be within the non-null overlap of the LTs of Y and A, and the specified TC end time is after the current time. For Case 2, the intersection of the LTs of Y and A must also be non-null, and the end time of the intersection must be after the current time. In Case 2, when a TC is not given, TC is set to Q in the

VUA. Note that Rule III is utilized to establish that Y is a DU of A, assuming that X has at least DA for A (X satisfies Rule I or II). Delegation requires Y to be authorized to A, hence UAM(A, Y) = 1.

Run-time assurance checks are similar to design time checks, and are enumerated and discussed below:

1. **MACC Domination:** The user CLR must dominate the role CLS. This dynamic check will revalidate this relationship between role and the user at run time. This run-time check is needed since the security privileges may have changed at the current time vs. the time of definition.
2. **Role Delegation**: At run-time, the security system must check to make sure the delegated user (DU) is not already a member of the delegated role before allowing a delegation to occur. This run-time check is needed since the security privileges may have changed at the current time vs. the time of definition
3. **User-To-User Delegation Authority Rule**: A user (OU or DU) who is a current member of a delegatable role (DUR), can delegate that user role at run time to any user that meets the prerequisite conditions of the role. This rule must also be checked during run time.
4. **Lifetime Consistency:** In regard to the permanence criteria (Section 3.3), the lifetime of the DU must be within the lifetime of the oOU as given in Definition 21. This run-time check is needed since the lifetimes must also be verified with respect to the current time.
5. **Modified Boolean Delegation:** In regard to the levels of delegation criteria (Section 3.3), the two levels (first, OU to DU – with optional DA and PODA, and second, DU to DU with only optional DA) must be enforced at run time using the Delegation Authority Matrix, DAM and User Authorization Matrix, UAM. Both levels of delegation must be checked at run time, with the further check to prohibit the second DU from delegating, enforcing the two levels.
6. **Delegation Revocation Authorization Rule:** An original user (OU) can revoke any delegated user (DU) from a role in which the OU executed the delegation. This is a stricter interpretation than [Zhan01], which allows any OU of a role revocation authority over a DU in the delegation path. In addition, a security administrator can revoke any delegation. This rule is checked during run time.
7. **Cascading Revocation Rule:** Whenever an original user (OU) or delegated user (DU) in the delegation path is revoked, all DUs in the path are revoked. This rule is checked during run time.

In addition to Rules I, II, and III, at runtime, there are additional rules that must be checked. First, Rule IV for the passing on of DA or DA/PODA from a user (either OU or DU) to another DU.

Rule IV: Let $X \in UL$ be an OU or a DU, $A \in URL$ be a role, and $Y \in UL$ be a DU of A (Rule III is satisfied).

- Case 1: Y can have DA for A ($DAM(A,Y) = 1$) if X has at least DA for A($DAM(A,X) \geq 1$).
- Case 2: Y can have DA and PODA for A ($DAM(A,Y) = 2$) if X has both DA and PODA for A ($DAM(A,X) = 2$).

Note that Rule IV establishes, respectively, DA or DA/PODA for user Y a DU of role A, and assumes that Rule III is satisfied. While the rule is general to n levels, our enforcement framework is limited to two levels.

Finally, Rule V, is for runtime delegation.

Rule V: Let $X \in UL$ be an OU or a DU. Then, at run time, X can delegate role A to DU Y limited by TC iff (Rule I or II) and Rule III.

Remember that the runtime check is needed since privileges are dynamic.

4.2 Administrative Tools for Role Delegation

For consistency in authorization, there must be the ability to define, examine, and control role consistency and delegation authority. Recall that Figure 1 depicted the *Security Policy Client (SPC)*, to manage URs by granting/revoking privileges (TCs, methods, SCs) and setting CLS levels, the *Security Authorization Client (SAC)* to assign CLRs and authorize roles to end users, the *Security Delegation Client (SDC)* to grant, update, and revoke delegations, and the *Security Analysis Tool (SAT)* to dynamically track all client activity, including logons and method invocations. SPC, SAC, and SDC all have different responsibilities in defining and assuring role delegation. In Figure 4, SPC is used to establish whether a role is delegatable (DUR). In Figure 5, SAC is used to authorize role delegation. From the user's point of view, once given authority by the security administrator to delegate roles, there must be tools for the OU to manage delegation of his/her roles (which are also used by the security administrator), which is accomplished via SDC, as we will shortly discuss. In addition to these definitional capabilities, Figure 6 is an example of one output of SAT. SAT will not prevent an unwanted delegation, but can track when delegations occur and how the delegated role is being used, which allows a security officer to hold a DU accountable, or assist in recovery when problems are detected.

The Security Delegation Client (SDC) is depicted in Figures 7, 8, and 9, and has three primary delegation functions: Grant, Update, and Revoke. In order to invoke the SDC, one must be an authorized user of a delegatable role. If this is the case, an OU will be able to invoke SDC. The Grant tab (Figure 7) allows the user to choose a DU, the delegatable roles, set the lifetime, and authorize pass on delegation, PODA. The Update tab allows a user to modify any existing delegation (Figure 8), while the Revoke tab enables the user to cancel any delegation (Figure 11). The delegation of a role by a user is similar to the security officer assigning roles, works in conjunction with the other elements of the Unified Security Resource (see Figure 1 and [Phil02a]), and utilizes the same underlying databases. With SDC, an organization has the flexibility to give certain users DA, while still maintaining administrative control and security assurance.

Finally, the design-time assurance checks, Rules I, II, and III, as given in Section 4.1, are summarized below with respect to the tool (SAC or SDC) that is utilized to perform the check.

1. **MACC Domination**: This check is performed by SAC.
2. **Role Delegation**: This check is performed by SAC.
3. **User-To-User Delegation Authority:** This check is performed by SDC.
4. **Lifetime Consistency:** This check is performed by SDC.
5. **Delegation Authority Matrix:** This check is performed by SDC.

Remember, all of the information for delegation is accessible via the Unified Security Resource (see Figure 1) which contains a database for storing and managing all security-related meta data. Thus, the design-time assurance checks can be in separate tools since the database is shared.

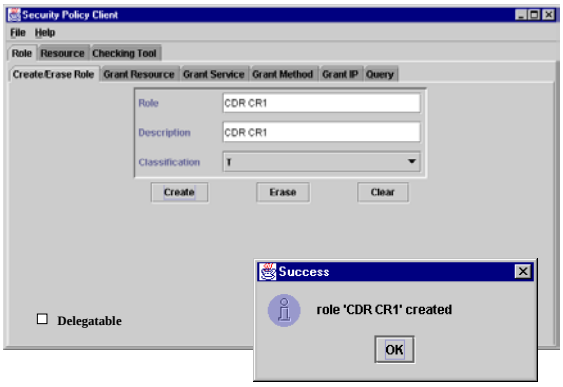

Figure 4: Security Policy Client, Create Role.

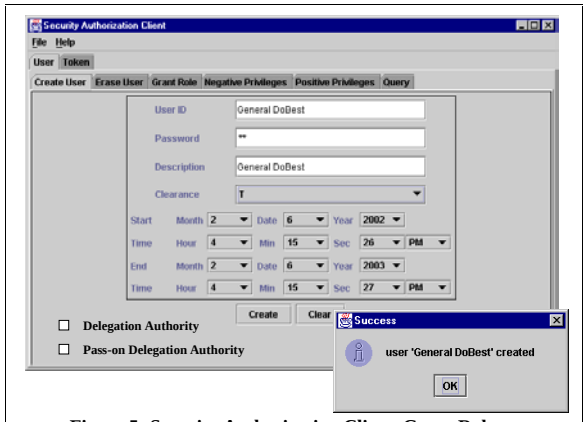

Figure 5: Security Authorization Client, Grant Role.

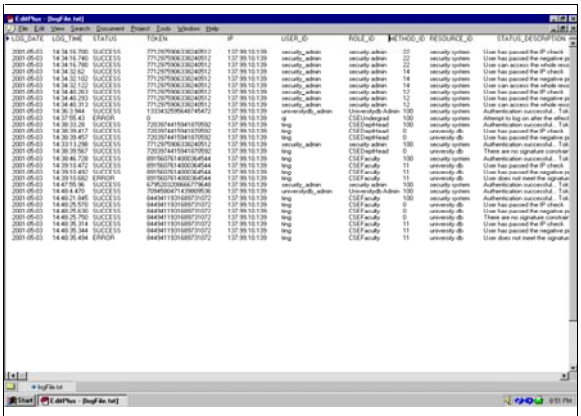

Figure 6: Security Access History from SAT.

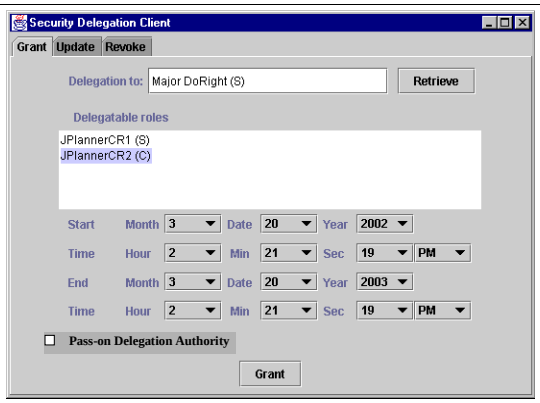

Figure 7: SDC Granting Delegation.

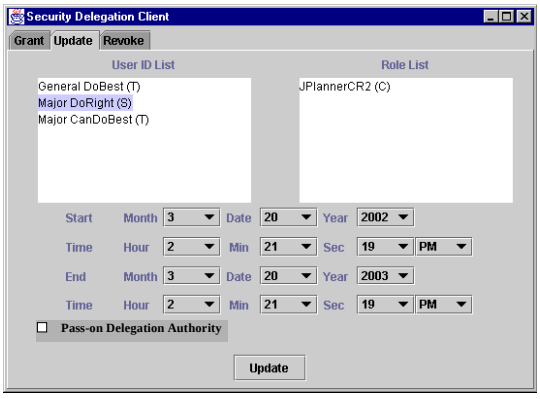

Figure 8: SDC Update Delegation.

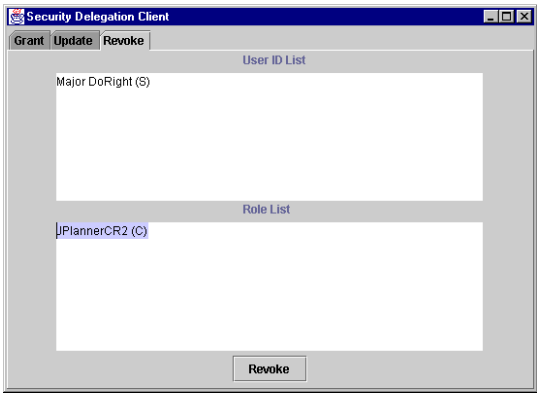

Figure 9: SDC Revoke Delegation.

5 CONCLUSIONS AND FUTURE WORK

This paper has examined the inclusion of role-delegation into a distributed, unified RBAC/MAC security model and associated enforcement framework with an added focus on attaining security assurance. To facilitate the discussion, in Section 2, we provided an updated version of our RBAC/MAC security model [Demu01, Phil02a], with minor revisions to support lifetimes of roles and users, which are important for

supporting role delegation. In Section 3, we presented the role-delegation modifications and extensions in three parts: the detailed extensions to the RBAC/MAC security model; the critical delegation rules to support enforcement [Zhan01]; and, an analysis of our approach against a set of delegation criteria [Bark00]. In Section 4, the revised and extended RBAC/MAC model/framework has security administrative tools (design time) and accompanying infrastructure (run time) for security assurance of user roles, classifications, end-user authorizations, and delegation. Overall, we believe this work provides a strong foundation towards role delegation for a distributed setting.

Ongoing research and prototyping efforts are in a number of different areas. First, there is ongoing research in the ability to define and establish user constraints, which in turn leads to a second area, role deconfliction, which involves both consistency constraints and mutual exclusion. Second, the prototyping in support of role delegation, namely, changes to SPC and SAC, development of SDC, and modifications to the Unified Security Resource, is ongoing (see http://www.engr.uconn.edu/~steve/DSEC/dsec.html). Finally, we are also investigating security issues for the dynamic coalition (DCP)[Phil02c] with an emphasis on the ability of our model (see Sections 2 & 3.1) and enforcement framework to support DCP.

REFERENCES

[Awis97] R. Awischus, "Role Based Access Control with Security Administration Manager (SAM)," *Proc. of 2nd ACM Wksp. on Role-Based Access Control*, November 1997.

[Bark00] E. Barka and R. Sandhu, "Framework for Role-Based Delegation Models," *Proc. of 23rd National Information Systems Security Conf.*, October 2000.

[Bell75] D. Bell and L. LaPadula, "Secure Computer Systems: Mathematical Foundations Model." M74-244, Mitre Corporation, Bedford, Massachusetts, 1975.

[Demu97] S. Demurjian and T.C. Ting, "Towards a Definitive Paradigm for Security in Object-Oriented Systems and Applications, " *J. of Computer Security*, Vol. 5, No. 4, 1997.

[Demu01] S. Demurjian, et. al., "A User Role-Based Security Model for a Distributed Environment," *Research Advances in DB. and Info. Systems Security*, Therrien (ed.), Kluwer, 2001.

[DoD88] DoD Directive 5200.28, "Security Requirements for Automated Information Systems (AIS)," March 1988.

[GCCS99] Joint Operational Support Center. "GCCS," DISA, 1999. http://gccs.disa.mil/gccs/

[Linn99] J. Linn and M. Nystrom, "Attribute Certification: An Enabling Technology for Delegation and Role-Based Controls in Distributed Environments," *Proc. of 4th ACM Wksp. on Role-Based Access Control*, October 1999.

[Na00] S. Na and S. Cheon. "Role Delegation in Role-Based Access Control," *Proc. of 5th ACM Wksp. on Role-Based Access Control*, July 2000.

[Phil02a] C. Phillips, et al., "Security Engineering for Roles and Resources in a Distributed Environment, ," *Proc. of the 3rd Annual ISSEA Conf*, March 2002.

[Phil02b] C. Phillips, S. Demurjian, and T.C. Ting, "Towards Information Assurance in Dynamic Coalitions," *Proc. of the 2002 IEEE Information Assurance Wksp*., June 2002.

[Phil02c] C. Phillips, T.C. Ting, and S. Demurjian, "Information Sharing and Security in Dynamic Colaitions," *Proc. of the 7th ACM Symp.on Access Control Models and Technologies*, June 2002.

[Sand98] R. Sandhu, "Role-Based Access Control", *Adv. in Computer Science*, Vol. 48. Zerkowitz (ed.), Academic Press, 1998.

[Sand00] R. Sandhu and Q. Munawer, "The ARBAC99 Model for Administration of Roles," *Proc. of the 15th Annual Computer Security Application Conf.*, October 2000.

[Zhan01] L. Zhang, G. Ahn, B. Chu. "A Rule Based Framework for Role-Based Delegation," *Proc. of 6th ACM Symp. on Access Control Models and Technologies*, Fairfax, Virginia, May 2001.