SECURITY ASSURANCE FOR A RESOURCE-BASED RBAC/DAC/MAC SECURITY MODEL

Charles Edward Phillips, Jr., Ph.D.
University of Connecticut, 2004

The day-to-day operations of corporations and government agencies rely on inter-operating software artifacts (e.g., legacy, commercial-off-the shelf (COTS), government-off-the-shelf (GOTS), databases, servers, etc.) and client applications, which are brought together into a distributed environment running middleware (e.g., CORBA, JINI, DCOM, etc.). In such a distributed environment, the interactions occur via the application programmer interfaces, APIs, of the software artifacts, which are available for use by any and all client applications, without restriction. However, security administrators are interested in controlling access by client applications to the methods of these artifact APIs as defined within a security policy. Specifically, they are interested in controlling: who (which client) can invoke which methods of artifact APIs at what times and how (data values). The "who" refers to whether the role and/or the security clearance allows the method to be invoked. The "which" refers to an exact definition, for each API, of the methods that can be invoked by each client (based on role or clearance level). The "what" refers to the time period that the method can be invoked, allowing invocation constrained based on time. Finally, the "how" refers to the actual parameters of the invocation, constraining based on value(s). This dissertation will present the findings of our research effort on the unification of role-based access control (RBAC) and mandatory access control (MAC) into a security model and associated security enforcement framework that provides a level of security assurance. Specifically, we provide the means for security officers to concretely and precisely specify a security policy for a distributed application using a resource-based unified RBAC/MAC security model which will allow fine grained control to the API's of software artifacts (operating in a environment running middleware, e.g., CORBA, JINI, etc.). The RBAC/MAC security model features and accompanying security assurance assertions can be utilized to control access to artifact APIs (methods) based on role, clearance/classification, time limits, and data value constraints. In this dissertation, we report on the research results of this work, focusing on: a detailed discussion of our current unified RBAC/MAC security model - core definitions and role delegation; a review of our accompanying security enforcement framework that utilizes our custom security resource that supports the RBAC/MAC model; an in-depth examination and proof of current security assurance guarantees, checked at design time and run time, which provides for both safety (nothing bad can happen) and liveness (all good things can happen)in attainment of the security policy; and a review of

our prototyping efforts. In addition, we report on related research and highlight the contributions of the research.

SECURITY ASSURANCE FOR A RESOURCE-BASED RBAC/DAC/MAC SECURITY MODEL

Charles Edward Phillips, Jr.

Master of Computer Science, Naval Postgraduate School, Monterey, CA 1989
Bachelor of Science in Engineering, United States Military Academy, West Point, NY 1981

A Dissertation
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy
at the
University of Connecticut
2004

APPROVAL PAGE

Doctor of Philosophy Dissertation

SECURITY ASSURANCE FOR A RESOURCE-BASED RBAC/DAC/MAC SECURITY MODEL

Presented by

Charles Edward Phillips, Jr.,

Major Advisor _____
Steven A. Demurjian

Associate Advisor _____
T.C.Ting

Associate Advisor _____
Ian Greenshields

University of Connecticut
2004

ACKNOWLEDGEMENTS

First, I give many special thanks to my advisor Steven A. Demurjian, Sr. for his constant support, encouragement, and patience throughout this entire Ph.D. process. I simply could not have reached this point without him and I will always carry an unwavering appreciation and debt of gratitude for his efforts on my behalf. Thank you for accepting me into this program, for pushing me, and for helping me make it through every gate.

I would like to thank Prof. Ting and Prof. Greenshields for participating in my committee and taking the time to read the dissertation and provide useful comments. Also, thanks to Lester Lipsky, Alex Russell, Reda Ammar, Eugene Santos and Alex Shvartsman for their candid comments on my work. You all gave valuable insights and constructive criticisms which made for a better product in the end.

Thanks to all my fellow graduate students who helped me in my research and prototyping efforts, I know I would not have made any progress with out you. There will always be a place in my heart for each of you and I will always consider you my friends.

I acknowledge the input, advice, and support of my colleagues in the USMA, Department of Electrical Engineering and Computer Science. I thank Professor Charles Reynolds, for the various formalisms, notations, and definitions in Chapters 3, 4, and 5. He really helped bring my work to culmination; Professor Jean Blair for protecting my time so I could finish this work; and CPT Jim Jackson for helping me through the growing-pains of Latex. I appreciate your support more than you probably realize.

Finally, I thank my family for putting up with my mid-life crisis. You have all had to sacrifice in different ways because of me and the time that has passed can regrettably, never be recovered. Please know I tried my best to lessen the impact of my work on you and that you were always in my thoughts and prayers. My dear wife Beth, I appreciate you more than I can say and I know I can never make up for everything you have done to support me over the years, but I will try with all my everything.

TABLE OF CONTENTS

Chapter 1: Introduction **1**
1.1 Security Assurance . . . 3
1.2 Security Solutions . . . 6
1.3 A Unified RBAC/DAC/MAC Model/Enforcement Framework for Assurance . . . 11
1.3.1 Problem Approach . . . 11
1.3.2 Relevance and Impact of Research . . . 16
1.3.3 Expected Contributions . . . 18
1.4 Credits . . . 21
1.5 Organization of the Dissertation . . . 24

Chapter 2: Background Concepts **27**
2.1 The Dynamic Coalition Problem . . . 28
2.2 Military: Global Command and Control System (GCCS) . . . 31
2.3 Access Control: RBAC, DAC, and MAC . . . 38
2.4 Middlware Security Solutions . . . 42
2.5 Related Work . . . 50

Chapter 3: The RBAC/DAC/MAC Security Model **52**
3.1 An Abstract Middleware Model . . . 54
3.2 RBAC and MAC Modeling Capabilities . . . 57
3.2.1 Lifetimes and MAC Security Levels . . . 58
3.2.2 Resources, Services, and Methods . . . 62
3.2.3 Authorization: Roles to Methods and Users to Roles . . . 66
3.3 Security Model Assumptions . . . 79
3.4 Related Work . . . 84

Chapter 4: Delegation in the RBAC/DAC/MAC Model **88**
4.1 Delegation Model Extensions . . . 90
4.2 Delegation Model - Delegation and Revocation Rules . . . 97
4.3 Delegation Model - Analysis of Delegation Capabilities . . . 99
4.4 Delegation Related Work . . . 105

Chapter 5: Security Assurance **107**
5.1 Time-Based Guarantees . . . 110
5.2 MAC Guarantees . . . 118
5.3 Safety and Liveness Guarantees . . . 125
5.3.1 Security Assurance Rules . . . 126
5.3.2 Proof of Safety and Liveness . . . 128
5.3.3 Summary of Rules and Theorems . . . 142

Chapter 6: Security Enforcement: Framework and Prototype 143
6.1 Security Enforcement Framework 144
6.1.1 The Unified Security Resource (USR) 147
6.1.2 Security Enforcement: Client Application 149
6.1.3 Participation of Resources in Security Paradigm 151
6.1.4 Security Administrative/Management Tools 152
6.1.5 Security Assurance Checks in the Prototype 154
6.1.6 Related Prototyping Research 158
6.2 Security Enforcement Prototype 159
6.2.1 Security Prototype Initialization 162
6.2.2 The Security Policy Client 166
6.2.3 The Security Authorization Client 179
6.2.4 The Security Delegation Client 184
6.2.5 Client Applications and Database Resources 192

Chapter 7: Research Contributions and Future Research 199
7.1 Research Contributions 200
7.2 Future Research 205

Bibliography 207

Appendix A: GUI for GCCS Example 214

Appendix B: Security Model Acronyms 223

LIST OF TABLES

1 Assumptions I - X. 80
2 Assumptions XI - XX. 81
3 Assumptions XXI - XXX. 82
4 Assumptions XXXI - XXXV. 83

LIST OF FIGURES

1 Security Assurance Problem 14
2 Near Simultaneous Crises. 31
3 General Coalition Architecture (5 countries). 32
4 Combined Operations Information Sharing. 33
5 Coalition Artifacts and Information Flow. 33
6 GCCS Services. 35
7 Common Operational Picture. 37
8 The CORBA Security Model. 44
9 The .NET Security Structural Model from MSDN 2003. 46
10 The Java 2 Platform - Compile and Execute. 49
11 Join, Lookup, and Invocation of Service. 56
12 LT of X compared to LT of Y. 60
13 Resource with Services/Methods. 64
14 Lifetimes for Resource/Services/Methods. 65
15 Resource/Services/Methods with SLEVELs. 66
16 Sample Users, User-Roles, and User-Role Authorizations. 68
17 Signature and Time Constraint for GCCS. 71
18 UR Authorization (URAM) and User Authorization (UAM) Matrices. 76
19 Example Users and Client Token. 79
20 Example: UAM, URAM, UDAM, URAM. 96
21 The Compare Function 111
22 Lemma 1, User-Role Authorization, Available Time 112
23 Lemma 2, User-Role, Available Time 114
24 Lemma 3 , User Authorization, Available Time 116
25 Security Enforcement Framework Software Architecture. 146
26 The Services of USR. 148
27 Client Interactions and Service Invocations. 151
28 Prototype Security and Resource Architecture. 161
29 Resource Database Scheme. 163
30 Start Global Clock and Security Servers. 164
31 Start Patient and University Database Servers. 165
32 Lookup Service Architecture. 166
33 Login: Security Policy Client 167
34 SPC: Register Resources. 168
35 SPC: Add Services to Resource. 169
36 SPC: Add and Confirm Methods to Resource. 170
37 SPC: Add Method to Service. 171
38 SPC: Resource Query. 172
39 SPC: Create a User Role. 173

40 SPC: Grant Resource to User Role. 174
41 SPC: Grant Service to a User Role. 175
42 SPC: Grant Methods to a User Role. 175
43 SPC: Create Signature Constraint on Method to User Role. 176
44 USR: Tracking Service. 177
45 USR: Security Analysis Service. 178
46 SPC: Query a Role. 179
47 SAC: Login Authentication. 181
48 SAC: Create a User. 182
49 SAC: Grant Role to User. 182
50 SAC: User Query. 183
51 SPC: Create Delegatable Role. 185
52 SAC: Grant Role with Delegation Authority. 186
53 Login: Security Delegation Client. 188
54 SDC: Delegation Errors. 189
55 SDC: Successful Delegation. 190
56 SDC: Update GUI. 191
57 SDC: Revoke Delegation GUI. 191
58 PDB: Client Authentication. 193
59 PDB: Add Patient. 194
60 PDB: Update Diagnosis. 194
61 PDB: Update Perscription. 195
62 PDB: Update Payment Mode. 195
63 PDB: Query Patient History. 196
64 PDB: Query Patient Diagnosis. 196
65 PDB: Query Patient Prescription. 197
66 PDB: Query Patient Payment. 197
67 PDB: Query Patient List. 198
68 Start Global Clock Server. 214
69 Start Security Server 215
70 Login: Policy Client Authentication 215
71 SPC: Register Resource 216
72 SPC: Register Service 216
73 SPC: Register Methods 217
74 SPC: Added Method to Service 217
75 SPC: Resource Query 218
76 SPC: Create Role 218
77 SPC: Add Resource to Role 219
78 SPC: Add Service to Role 219
79 SPC: Add Method to a Role 220
80 SPC: Add a Signature Constraint 220
81 SPC: Query a Role 221
82 SAC: Create a User 221

83 SAC: Grant Role to a User 222
84 Acronyms - Alphbetical Order 224
85 Acronyms - Order of Appearance 225

Chapter 1

Introduction

Research and development for access control of data and databases has evolved into three approaches: *mandatory access control (MAC)*, *discretionary access control (DAC)*, and *role-based access control (RBAC)*. The MAC approach is based on security *classification* of *objects* and *clearance* of *subjects* [30, 65, 70, 107, 122] using the Bell and La Padula security model [13]. A classification (CLS) or clearance (CLR) refers to a security level (e.g., typically unclassified (U), confidential (C), secret (S), and top secret (T) forming a linear order U $<$ C $<$ S $<$ T) that are assigned to the data (each object has a CLS) and users (each subject has a CLR) which represents a degree of sensitivity. For subjects, the sensitivity represents the level of access; for objects, it represents the level of protection. A subject (user) can access an object (data item) if the subject's CLR *dominates* an object's CLS, which denotes that CLR $\geq$ CLS. The multi-level secure MAC support is at the core of the requirements for trusted computing [32].

The DAC approach allow security emphasis to place the security privilege definition process in the hands of both security administrators and authorized users, adding discretion to the process. The RBAC approach [69, 111, 118, 130, 131], a refinement of DAC, aligns the security definition and management process to the end-user, providing alternative roles to capture responsibilities. DAC and RBAC together support the concept of *delegation*, which allows one user to transfer the responsibility of a role to another authorized user. In *administratively-directed delegation*, an infrastructure outside the direct control of a user mediates delegation [68]. In *user-directed delegation*, a user (playing a role) determines if and when to delegate responsibilities to another user to perform the role's permissions [79], with administrators continuing to establish the security policy and maintain delegation authority. Administration of DAC, RBAC, and MAC with delegation must be controlled to ensure that policy does not drift away from its original objective [114].

As security approaches, MAC, DAC, and RBAC must provide capabilities for the definition of security privileges (design time) and the enforcement of those

security privileges in a dynamic environment (runtime), and these capabilities are embodied in a number of key concepts. First, a *security policy* represents for an application its set of security requirements, including: access control approach, identification of what needs to be protected, anticipated users, user privileges, and so on. Second, *authorization*, a design time action, is critical to a security policy since it is concerned with the actions taken by a security officer/engineer (or other security administrative/management personnel) to establish (grant/revoke) privileges for each user (or user role) based on various criteria. These security privileges must adequately support a user's activities, i.e., *least-privilege*, access to only that information necessary to accomplish one's tasks and no more [42]. Third, *authentication*, a runtime action, is the enforcement of the security policy, focuses on who the users are, and when authenticated, limiting their actions to their precise authorized privileges. Finally, and most importantly, *security assurance* is required to insure that the security-policy-definition process yields a consistent result that is constantly attainable when the policy is realized within the enforcement framework for an active and working application.

1.1 Security Assurance

MAC is governed by strict rules for subjects (users) to access stored information or objects. These rules lack flexibility because, for assurance reasons, there can be no comprise of classified information. Government classified information rules are detailed in the JOS Orange Book Security Requirements [35]. Specifically, in MAC, assurance-related properties represent the association between subjects and objects in terms of their read and write capabilities with respect to security levels (T, S, C, U). In the *Simple Security Property*, a subject can read information that has a classification that is at the same or lower clearance level of the subject [13], which is referred to as *read down - no read up*, i.e., can read at the same or lower level. By using Simple Security, the chance of comprise is eliminated. There is a basic assumption that subjects will only share information at the appropriate levels. In the *Strict *-Property*, a subject can only write information at exactly the level for which it is cleared [91], which is referred to as *write-equal*. For example, if one is sending an email on a SECRET machine, the only machine that can receive that email is another SECRET machine, regardless of the information contained in the message. There is no flexibility; this is a closed, single-level system.

In the *Liberal *-Property*, a subject with a low clearance level can write to an object with the same or higher clearance level [13, 91], which is referred to as *write up - no write down*. Such a capability is known as a *blind write* since the subject can modify an object without seeing it. This protects the flow of classified information since low-level information can be sent to a higher level, but higher classifications can never be sent to a lower. For example, it would not be a security violation to update unclassified information from an unclassified

machine to a database on a SECRET machine (write up). However, it would be a security violation to update unclassified information from a SECRET machine because the information flow is from high to low (no write down), regardless of the information itself. Finally, in the *Simple Integrity Property*, a subject can write information at its own level and lower [18], which is referred to as *write down - no write up*. Such a capability can lead to *information leakage* which is the transfer of information from a higher to a lower security level. From a security administrative perspective, a subset of the MAC properties are chosen to define the security behavior for each application. Specifically, an application could choose to support the Simple Security Property for read and the Liberal-* Property for writes. Alternatively, an application might decide on a read-equal policy (only see objects that have CLSs that exactly matches the subject's CLR) in conjunction with Strict-* to have a write-equal policy. The choice that is made depends on the security requirements for an application.

In addition, assurance must focus on application behavior with respect to its defined and realized security policy, to attain *safety* (nothing bad will happen to a subject or object during execution) [62] and *liveness* (all good things can happen to a subject or object during execution) [5]. Safety and liveness are what all security systems and security mechanisms try to achieve. We approach maximum safety when we isolate systems. In a network or distributed system if we isolate or disconnect all computers from the network we no longer have a network. This maximizes safety as nothing can bad can happen over a network connection, but there is no liveness, we cannot do what we need to do, which is unacceptable. A common real-world example of a balance between safety and liveness is the firewall. A firewall allows a protected system to operate in a network or as part of a distributed system with protections from those who would do bad things. A firewall provides safety and allows for some liveness. However, a firewall can also be very strict and prevent a system user from using some utilities or preventing access to some necessary data, which can be frustrating. A firewall tries to maximize both safety and liveness, but sometimes liveness gives way to safety. *Security assurance in terms of specific MAC properties, and safety and liveness is be one of the foci of this dissertation, and has been widely cited as one of the paramount security concerns [4, 45, 71, 72].*

1.2 Security Solutions

There are many different realizations and solutions of security as presented in Section 1.1 in support of MAC, DAC, and RBAC. Historically, security has been data focused, concentrating on protecting information in databases. Commercially, MAC has been supported in database management systems such as Oracle [83], and as part of the SQL-2 ANSI/ISO standard. In SQL-2, DAC is supported with the ability to grant privileges (i.e., INSERT, DELETE, UPDATE, and SELECT) to users against an entire schema or individual tables. Moreover,

delegation is support, giving the ability to pass on granted privileges to other users. In addition, MAC has been supported in Sybase [107], and has been the subject of concentrated research [30, 65, 70, 122]. While these solutions are all relevant, the reality is that all of these solutions have an impedance mismatch between the secure data repository and the end-user application. Specifically, the majority of graphical-user interfaces (GUIs) for application-users have adopted an object-oriented or component-based solution via an object-oriented programming language (e.g., Java, C++, Ada95, etc.). As such, the GUI uses and manipulates object-oriented data via operations (methods), which is in a dramatically different format than the tabular data found in database systems such as Oracle and Sybase. This is the impedance mismatch, between the security of the information in the repository vs. the requirement to access this information via operations (methods) that focus on usage. To address this impedance mismatch, there has been a significant number of efforts that consider the inclusion of security (MAC, DAC, and RBAC) into object-oriented database systems [20, 69, 99, 128] and programming languages [9, 25, 48, 93, 102, 121].

In the object-oriented database area, there has been work by [99] that proposes a model of authorization for next-generation database systems. In this model, an authorization is defined as a 3-tuple (s, o, a) of subject, authorization object, and authorization type. Roles, organized into a lattice, are defined to reduce the number of authorization subjects. Authorization types and objects are also organized into lattices, with authorization as either implicit/explicit, strong/weak, or positive/negative. The work has also been extended by adding the class-superclass structure and by introducing an integer-based priority system as an overriding mechanism [20], and includes the concept of a permission tag that indicates the allowable and prohibited actions for a subject. In another effort, an object-oriented data model with multi-level security (Secure ORION) is proposed [128]. In this model, a set of security properties are held for objects, classes, methods, aggregate classes and objects, and relationship objects, so that each entity is assigned a security level. The ability of a subject to execute a method also depends on the security levels defined on the subject and the method. A role-based security scheme is proposed in [69], with roles organized into a network, to provide a finer-grained classification of the users of a DBMS via an object-oriented approach. To define authorization, one specifies, for each access right and for each data level, the roles included (permitted the access right) and the roles excluded (denied the access right).

In the object-oriented programming and design areas, there has been work on *aspects* as a mechanism for object-oriented models to extend a given class with new capabilities, including, roles [102]. In this work, new operations and data are possible for new roles. Another effort allows different classes in an application to have different subjective views [48], similar in concepts to views in a database system, but applying the concept to public interfaces of classes. Other work has

focused on composing these subjective views with only scant mention of implementation support in C++ [93]. An effort on role-based access control for object technology [9] works at a class level; when different roles require specific access to a class, subclasses for the roles are created to turn-on/turn-off the appropriate access. Shilling and Sweeney [121] have extended the object-oriented paradigm in three steps: defining multiple interfaces in object classes to allow an object to realize different and independent sets of behaviors; controlling the visibility of instance variables to provide information hiding; and, allowing multiple copies of an instance variable to occur within an object instance. The extensions are used to create view classes and view instances which can then provide the view facility. Finally, work in [25] defines a role-hierarchy and focuses on assigning and prohibiting methods of classes on a role-by-role basis, effectively allowing different roles to have customized access to the public interfaces of classes.

As we enter the 21st century, the emphasis on strictly an object-oriented design and development approach has been forced to evolve to react to the reality of the structure and organization of modern applications. Specifically, the focus on developing new applications from scratch has been replaced with an approach that assembles the software artifacts (i.e., legacy, commercial-off-the-shelf (COTs), government-off-the-shelf (GOTS), databases, new/old clients, new/old servers, etc.) that exist in a distributed environment into large-scale interoperating applications. These applications require stakeholders (i.e., software architects, system designers, security officers, etc.) to architect and prototype solutions that facilitate the interoperation of new and existing applications in a network centric environment. The software artifacts all interact with one another via middleware, which is available in a wide range of platforms (DCE [92, 106], CORBA [87, 133, 137], DCOM/OLE [74], J2EE/EJB [105, 132], JINI [6, 134], and .NET [103, 115]). All of these approaches are focused on the *application programmer interfaces (APIs)* of the software artifacts that are available for use by client applications. These APIs of the software artifacts represent the functionality that is provided, and these interfaces may be written in different programming languages (e.g., C, C++, Java, etc.), or be defined using a common interface definition langauge (e.g., IDL in CORBA). In a middleware approach, the software artifacts are referred to as *resources* that provide *services* (i.e., APIs) available for use by clients and other resources in the inter-operating environment. Regardless of the approach, the incorporation of security has often been an afterthought, dependent on programmatic effort rather than a cohesive mechanism seamlessly incorporated into the underlying technology.

Specifically, modern middleware platforms like CORBA, .NET, and J2EE, do offer a set of limited security capabilities. For example, CORBA, has security features for confidentiality, integrity, accountability, and availability [87]. However, these capabilities in CORBA are really a meta-model, intended to represent a breadth of security capabilities representing different security models, paradigms, and techniques, rather than actual security like MAC, DAC, and

RBAC. Platforms such as .NET and J2EE provide actual security capabilities via their respective runtime environments via their APIs, and include capabilities for code-based access control for security embedded in actual program code, secure code verification and execution which focuses on runtime security support, secure communication of the exchange of messages and information, and secure code and data protection to detail cryptographic capabilities [29]. But direct support for MAC, DAC, and RBAC is not provided in a direct fashion, rather they must be programmatically developed on an application-by-application basis. *A second foci of this dissertation is to provide a unified security model that is built on middleware concepts (resources and the services that they provide to clients) that includes select RBAC, MAC, and DAC capabilities, to yield an enforcement framework that guarantees a degree of security assurance.*

1.3 A Unified RBAC/DAC/MAC Model/Enforcement Framework for Assurance

The foci of this dissertation is two-fold. First, we seek to provide security modeling and enforcement techniques that integrate features of RBAC, DAC, MAC, temporal access, and value-based access to allow selective access to the APIs (methods) of software artifacts by users (executing client applications) and other artifacts through a security infrastructure and framework. Second, we strive to attain security assurance as embodied by the Simple Security and Simple Integrity Properties, and through safety and liveness. To balance our security modeling and assurance research, we detail our prototyping experiences that clearly illustrate the utility and feasibility of our efforts. In the remainder of this section, we discuss our approach to providing a security model and enforcement framework, the relevance of our security work with the proliferation of the Internet and the reliance on distributed operation, and we conclude this section by revealing the expected contributions of this work.

1.3.1 Problem Approach

Our intent is to provide a security model and framework in support of interacting software artifacts and client applications, that is focused about the APIs of the software artifacts. We assume that the interactions occur using a middleware platform (e.g., CORBA, .NET, etc.) where software artifacts (*resources*) register their APIs (*services*) of methods with a *look up service* that is available for discovery by client applications. In terms of security capabilities, we seek to provide an integrated, unified *security model* that incorporates features of RBAC (roles for client applications), DAC (delegation of authority among users), and MAC (clearance for users and classifications for roles and methods). The authorization (granting/revoking) of methods to roles establishes the security privileges, and can be constrained by security levels (e.g., U, C, S, TS), the time period when the

method can be invoked, and the values under which the method can be invoked. Once established, roles can be authorized to users, when then limit the actions of the users within the client applications.

The authorized privileges (of methods to roles and of roles to users) must satisfy specific security assurance requirements at design-time (during the security policy definition) and at runtime (by the enforcement framework). Thus, the unified RBAC/DAC/MAC security model must provide assurance, and will include capabilities for: domination of MAC security levels (e.g., role dominates method, user dominates role, etc.), simple security and integrity properties for MAC, consistency checks in terms of valid time interval for delegation, and overall, the attainment of a degree of safety and liveness. The RBAC/DAC/MAC security model provides the primitives for security officers to define a security policy, and provides design-time security assurance checks. These checks insure consistency and correctness in security policy definition. The RBAC/DAC/MAC security model is supported by an associated runtime *enforcement framework*, which provides the mechanisms and infrastructure for software artifacts to securely interact with clients and one another. The enforcement framework also offers runtime security assurance checks (also formally defined) which insure consistency of the realized security policy. To realize the security model and enforcement framework, we employ a middleware-based abstract model to allow the security to be integrated into the distributed application environment in a manner that is consistent with the interacting software artifacts and clients. To demonstrate the feasibility and utility of our approach, we have prototyped the model and associated enforcement framework with accompanying security policy administrative and management tools.

Specifically, as shown in the right hand side of Figure 1, we assume a distributed environment comprised of software artifacts (e.g., legacy, COTs, GOTs, databases, servers, etc.) that interact with each other and client applications via middleware (e.g., in our case, Visibroker (CORBA) and JINI). All of the interactions between artifacts and clients occur in a controllable fashion via APIs of the software artifacts; which are typically available for use by any and all client applications, without restriction. Our RBAC/DAC/MAC security model and enforcement framework supports secure access to APIs, allowing a security officer to define a security policy that precisely grants and revokes the methods of the artifact APIs to users executing client applications, which is then enforced across the distributed application. In particular, as indicated in Figure 1, we provide the means to control access to artifact APIs, namely, to the invocation of methods by users (via client applications), based on: the **role of the user**, the **delegation status of the role**, the **security clearance level** of the user, the **domination** of the classification level of the role over the level of the method, the **time period** when a user playing a role can invoke the method, and the method **signature values** for which a user playing a role is constrained to invoke the method. While there has been noteworthy research in each of these individual areas (role-based,

delegation, clearance/classification, time limited and value constrained access) the combination of all areas and support within a distributed setting has not been considered to the extent which we have been pursuing, particularly from the context of security policy definition and assurance.

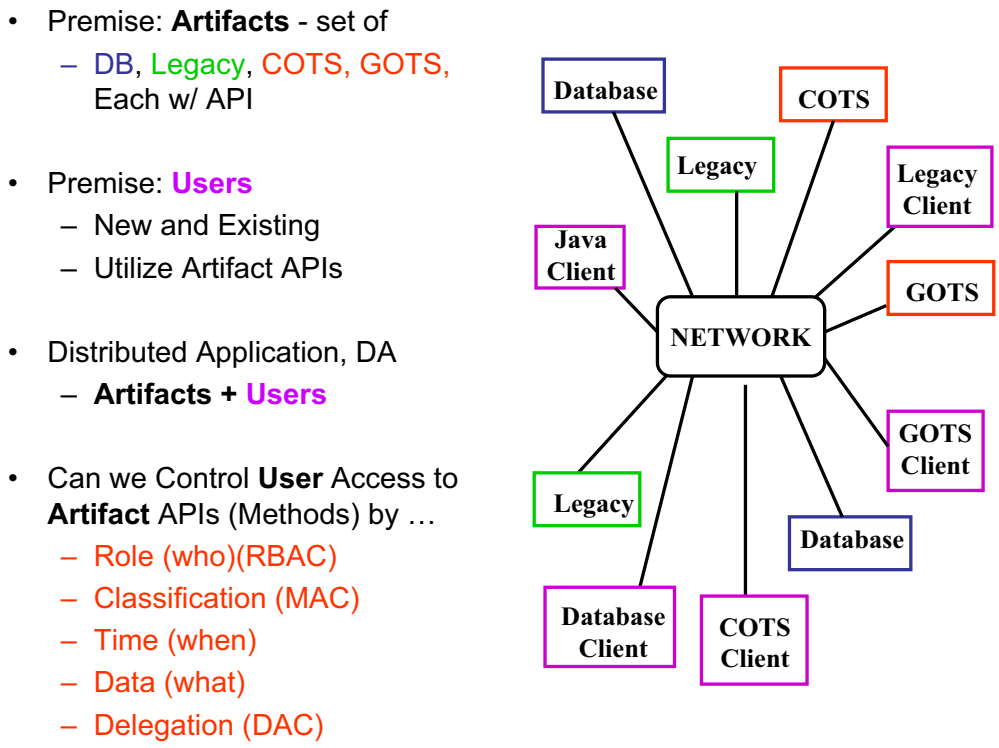


Figure 1: Security Assurance Problem

In the introduction to this chapter, and in Sections 1.1 and 1.2, we have motivated and discussed MAC, DAC, and RBAC in detail. Briefly, we do the same for time-based and value-based access in the context of our approach. *Controlling access based on time* requires temporal constraints for RBAC that limit a role by duration or have a time triggered activation [3] and temporal authorizations on objects [14]. Both of these works acknowledge the need to control when a user can access an application or parts of an application. For our research, temporal enforcement (at runtime) can insure, for example, that a user playing a role can only invoke a method if the user's temporal constraint is consistent with the temporal constraint on the method, insuring that only authenticated users authorized to a role can invoke a method at a specific time. The ability to provide *access control based on data* can be utilized to dictate what a user can and cannot do with respect to actual values. Research in this area includes value-based constraints for mutual exclusion and four-eyes only are critical to prevent error or misconduct [37], and separation of duty efforts for workflow systems using

consistency constraints [16]. For our research, value-based constraints limit the authorization of a method to a role by particular values for a methods parameters, to define the values under which an invocation is allowed. For example, Clerk and Supervisor roles are both authorized to the "cash checks" method, with the clerk limited to $100 and the supervisor $200, which is enforced at runtime on the basis of an authenticated user's role.

1.3.2 Relevance and Impact of Research

Security engineering is most often an after thought in the design and implementation of software systems. With the proliferation of the Internet and the reliance on distributed operation, organizations can no longer ignore security failures or shortcomings, and security needs to be elevated to a first-class citizen in the overall requirements, design, development, and maintenance process. We believe that the relevance and impact of the research in this dissertation is focused on the following six points:

- Security assurance is critical to the success of information sharing and our unified RBAC/DAC/MAC security model and enforcement framework with its security assurance capabilities addresses critical aspects of information sharing.

- We provide security assurance and management to access control in distributed environments. Our security framework and infrastructure for a distributed setting is based on an abstract middlware model, and as such, provides flexibility, portability, and platform independence.

- Current government systems achieve mandatory access controls and multi-level security by separating systems rather than providing a true MAC capability. Our formal RBAC/DAC/MAC security model unifies a user-focused approach (roles) with delegation with more stringent security levels (MAC) to address government requirements, and be available for other suitable organizations and domains.

- The incorporation of multi-level security is especially relevant, particularly in military and governmental (e.g., homeland security) applications.

- Our usage of RBAC, delegation (DAC), the inclusion of constraints based on time and data values, and security clearance/classifications, offers a flexibility of usage that spans a wide breadth of capabilities and domains.

- Our approach can be utilized ot federate users and resources in a distributed environment which is normally security prohibitive, and incorporate multiple security policies (RBAC, DAC, MAC) and assurance capabilities with targeted changes to artifacts and clients at the source code level.

The relevance of this work can be applied directly to the U.S. Military. In recent years, with shrinking budgets and force structure, the U.S. Military has become more reliant on the civilian sector for operational support and use of the Internet to execute necessary tasks. This requires interaction between legacy software systems, COTS, GOTS, and shared databases. There has only been limited success in this area due to the restrictive security requirements imposed on government systems. To demonstrate the utility and relevance with respect to this domain, the majority of the examples given throughout this dissertation involve the Global Command and Control System, GCCS [46].

Another relevant potential domain for our security research is the dynamic coalition problem, DCP, (NEED REFS) where there is the need to dynamically federate users and resources while simultaneously maintaining information assurance. For our purposes, DCP has inherent security risks incurred as a result of federating participants in a crisis quickly, yet still needing to share information. Crises will be concurrent and involve a different set of players and resources depending on the type of crisis and the political situation. Coalitions will be comprised of military, civilian, government and non-government agencies that need to share information. As each crisis evolves, so will the information sharing needs and security requirements. For these reasons, a security solution also needs to be rapidly deployable, easy to use, platform independent, and allow for dynamic policy configuration.

1.3.3 Expected Contributions

One most significant expected contribution of this dissertation is the unified RBAC/DAC/MAC security model based on an abstract middlware-based paradigm. This model embodies the significant aspects of security assurance (see Section 1.1) and has several unique features. First, this security model unifies RBAC, DAC and MAC. MAC is considered too inflexible to use in most situations because of the rigid requirements of MAC according to the Bell and La Padula Model [13]. Second, is our usage of time-based constraints for temporally controlled access. By defining time-periods of access, or *lifetimes* on objects and subjects, we are able to establish a window of access of when objects are available to subjects. Other approaches have used temporal constraints in various ways to control processing, but requiring a lifetime and the possibility of an additional time constraint allows for very fine-grained access control in a time sensitive and dynamic environment not realized before. A third feature of this security model is the use of value-based constraints to govern access to methods based on parameter values. The public methods that are part of the APIs are normally available to all in an unconstrained manner. Our approach provides fine-grained access to methods, allowing multiple roles which utilize the same public method to have different constraints under which an invocation can occur. This runtime check,

an aspect of assurance, is enforced on every method invocation. Finally, by supporting the delegation of authority from user to user (on a role-by-role basis), we provide a level of DAC in our approach, yielding a unified model that includes features from three of the major security approaches.

Another significant expected contribution is a proof of safety and liveness for our unified RBAC/DAC/MAC model under a certain set of assumptions. Using our model, we will prove that an authorized user can execute any authorized task, and only those authorized tasks, without violating the established security policy, which is *safety* (nothing bad will happen to a subject or object during execution) [62] and *liveness* (all good things can happen to a subject or object during execution) [5]. This is done thought a series of eight safety and liveness proofs which are based on eight formally defined security assurance rules. The *security assurance rules, SARs*, are derived from the security model by a combination of proven lemmas and security model definitions. These SARs dictate what is required for a user to become a user and for a user to execute (invoke a method) a portion of a user role. These SARs also cover the additional security assurance requirements necessary for role delegation. Simply put, these rules govern what is acceptable and unacceptable for both runtime and design time activities, and as such embody one aspect of the security assurance of our approach. The second assurance aspect embodied in these rules, and part of our unified model, focus on the MAC Simple Security and Simple Integrity Properties. Simple Security dictates that a subject can read information at the same or lower clearance level (read-down/no-read-up) and Simple Integrity Property dictates that a subject can write information at its own level and lower (write-down/no-write-up). This is very important when dealing with objects that can be manipulated by a combination of read or write methods.

The final expected contribution of this dissertation is the proof of concept prototype. We have realized our unified RBAC/DAC/MAC security model, based on an abstract middlware model, in support of interacting software artifacts and client applications. In order to accomplish this, we leveraged middleware capabilities, to easily and seamlessly incorporate our security model and associated enforcement mechanism. RBAC/DAC/MAC security model. We have demonstrated a degree of flexibility, portability, and platform independence in our solution approach, through a prototype that utilizes multiple middleware platforms (i.e., JINI, CORBA), databases (Oracle, Access), and operating systems (Linux, NT, Win), in support of applications in health care, a university setting, and military acquisition/logistics.

1.4 Credits

There have been a number of articles that have been published related to the research presented in this dissertation, and I was primary author on all but three of the articles. These first three articles represent the initial ideas (first article

on RBAC for distributed, middleware based applications) and the security model (second article) with MAC and delegation (DAC) capabilities (third article):

- Phillips, C., Demurjian, S., and Ting, T.C., "Security Engineering for Roles and Resources in a Distributed Environment," Proc. of 3rd Annual Intl. Systems Security Engineering Association Conf., Orlando, FL, March 2002.

- Liebrand, M., Ellis, H., Phillips, C., Demurjian, S., Ting, T.C., and Ellis, J., "Role Delegation for a Resource-Based Security Model," in Data and Applications Security: Developments and Directions II, E. Gudes and S. Shenoi (eds.), Kluwer, 2003. This is a revised version of the paper: Liebrand, M., Ellis, H., Phillips, C., Demurjian, S., and Ting, T.C., "Role Delegation for a Distributed, Unified RBAC/MAC," Proc. of 16th IFIP WG 11.3 Working Conf. on Database Security, Cambridge, England, July 2002.

- Demurjian, S., Ting, T.C., Balthazar, J., Ren, H., Phillips, C., and Barr, P., "A User Role-Based Security Model for a Distributed Environment," in Data and Applications Security: Developments and Directions, B. Thuraisingham, R. van de Riet, K. Dittrich and Z. Tari (eds.), Kluwer, 2001. This is a revised version of the paper: Demurjian, S., Ting, T.C., Balthazar, J., Ren, H., Phillips, C., and Barr, P., "Role-Based Security in a Distributed Resource Environment," Proc. of 14th IFIP WG 11.3 Working Conf. on Database Security, Scoorl, The Netherlands, Aug. 2000.

The second three publications involve the security assurance aspects of the research.

- Phillips, C., Demurjian, S., and Ting, T.C., "Towards Information Assurance in Dynamic Coalitions," Proc. of 2002 IEEE Info. Assurance Workshop, West Point, NY, June 2002.

- Phillips, C., Demurjian, S., and Ting, T.C., "Security Assurance for an RBAC/MAC Security Model," Proc. of 2003 IEEE Info. Assurance Workshop, West Point, NY, June 2003.

- Phillips, C., Demurjian, S., and Ting, T.C., "Safety and Liveness for an RBAC/MAC Security Model," in Data and Applications Security: Developments and Directions III, E. Gudes and S. Shenoi (eds.), Kluwer, 2004. This is a revised version of the paper: C. Phillips, S. Demurjian, T.C. Ting, "Assurance Guarantees for an RBAC/MAC Security Model," Proc. of the 17th IFIP 2002 11.3 WG Conf., Estes Park, CO, August 2003.

The final three publications involve the enforcement framework (first two) and the applicability of our research to complex applications such as dynamic coalitions (third article):

- Phillips, C., Demurjian, S., and Bessette, K., "A Service-Based Approach for RBAC and MAC Security," to appear in Service-Oriented Software System Engineering: Challenges And Practices, Z. Stojanovic and A. Dahanayake (eds.), Idea Group, 2004.

- Demurjian, S., Bessette, K., Doan, T., and Phillips, C., "Concepts and Capabilities of Middleware Security," to appear in Middleware for Communications, Q. Mohammed (ed.), John-Wiley, 2004.

- Phillips, C., Ting, T.C., and Demurjian, S., "Information Sharing and Security in Dynamic Coalitions," Proc. of 7th ACM Sym. on Access Control Models And Technologies, SACMAT, 2002, Monterey, CA, June 2002.

The material in this dissertation builds on the ideas included in the following research:

- Osborn, S., "Mandatory Access Control And Role-Based Access Control Revisited," *Proc. of the 2nd ACM Workshop on RBAC*, Nov. 1997.

- Sandhu, R., "Role-Based Access Control," in *Advancements in Computer Science*, Academic Press, 1998.

- Ferrailo, D.and R. Kuhn, "Role-Based Access Controls," *Proc. of 15th NIST-NCSC National Computer Security Conference*, Oct. 1992

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

Note that these are the key articles; throughout the dissertation other relevant articles will be cited as necessary.

1.5 Organization of the Dissertation

The remainder of this dissertation contains six chapters. Chapter 2 details our research interest and defines the problem by providing background information, specifically, an introduction to the Global Command and Control System (GCCS) and the Dynamic Coalition Problem (DCP). Both DCP and GCCS are the major motivation behind the research presented herein. In addition, Chapter 2 contains a conceptual examination of different access controls approaches, and a discussion of related work in all relevant areas.

Chapter 3 contains a formal definition of our unified RBAC/DAC/MAC security model, which is one of the main contributions of this dissertation. The core model definitions as given in Chapter 3 focus on the RBAC and MAC capabilities of the model, and comprise the majority of the chapter. We introduce and formalize concepts of lifetimes, sensitivity levels, and the assumptions of an

abstract middle model. These constructs are used to build security assurance rules and authorizations which will be presented in Chapter 5 and provide the basis for our security enforcement framework and prototype (see Chapter 6). The chapter details the design assumptions required to clearly establish the security model environment and security assurance requirements. This chapter concludes with a discussion of other security models and related work.

Chapter 4 extends the security model to support DAC with the support for role delegation. Role delegation adds a discretionary aspect to our security model which improves flexibility when defining a security policy. This chapter provides a comprehensive analysis of role delegation capabilities and the accompanying security challenges. The chapter includes formal model definitions for role delegation that extends and supplements the RBAC/DAC/MAC model. The chapter also details role delegation revocation rules, and concludes with related delegation work.

Chapter 5 contains formal definitions of our security assurance guarantees in terms of time, and the simple security and integrity properties of MAC, and provides proofs of safety and liveness. The main objective of this chapter is to explore assurance for the RBAC/DAC/MAC security model and build security assurance rules. The chapter focuses on which methods of APIs can be invoked based on user/user role relationships, sensitivity levels, data values, and temporal considerations. Assurance rules are used at both design time (security policy definition via the RBAC/MAC model) and at runtime (enforcement framework) to provide a confidence level to security officers regarding the attainment of their security policy during definition and execution. The chapter presents a series of theorems that corresponds to each assurance rule, proving that each rule attains a degree of safety and liveness under our assumptions.

Chapter 6 reviews our security enforcement framework and prototype, focusing on the conceptual underpinnings of processing and its realization in a working system. The chapter details the security resource that embodies the RBAC/DAC/MAC model, and its associated services that required to meet the security model, which results in an enforcement framework software architecture. The remainder of the chapter concentrates on the security research prototype that has been constructed, and includes a detailed discussion of the available administrative security tools for a security officer to define and manage a policy. Flow diagrams and bit-maps clearly illustrate the processing capabilities of the working system.

Finally, Chapter 7 summarizes the research contributions and future research. The chapter begins with a review of the distributed application security problem and our approach. We then highlight the contributions of our research with potential real-world applications and the goals achieved. We conclude by referencing a number of future research areas where finer-grained security can be achieved and different techniques can be implemented.

Chapter 2

Background Concepts

In this chapter we provide background information that is relevant for all of the chapters in the remainder of this dissertation. Our emphasis is on setting the context for the work presented herein; as necessary, background information will be introduced in subsequent chapters in support of the material presented in the chapter. This chapter focuses on relevant application domains that embody the interacting software artifacts and clients (see Chapter 1). Specifically, our focus in Section 2.1 is to delineate the critical challenges of the *dynamic coalition problem, DCP*, with an emphasis on information assurance, which will be illustrated by utilizing real-world scenarios and examples derived from a military version of DCP, in particular the *Global Command and Control System*, GCCS, as presented in Section 2.2. Using the material in Sections 2.1 and 2.2 as a basis, we re-examine RBAC, DAC, and MAC, for their relevance in support of DCP and GCCS in section 2.3. To conclude this chapter, in in Section 2.4 we briefly review middleware security capabilities for CORBA, .NET, and Java, and in Section 2.5 we detailed other related research.

2.1 The Dynamic Coalition Problem

Today, information sharing is critical to almost every institution. There is no more critical need for information sharing than during an international crisis, when international coalitions dynamically form. In the event of a crisis, whether it is humanitarian relief, natural disaster, combat operations, or terrorist incidents, international coalitions have an immediate need for information. These coalitions are formed with international cooperation, where each participating country offers whatever resources it can muster to support the given crisis. These situations can occur suddenly, simultaneously, and without warning. Often times, participants are coalition partners in one crisis and adversaries in another, raising difficult security issues with respect to information sharing. Our specific interest is in the Dynamic Coalition Problem (DCP) [8, 97, 119], with an emphasis on the

information sharing and security risks when coalitions are formed in response to a crisis. As motivation for our security model, we define the DCP and explore its intricate, challenging, and complex information and resource sharing, and security issues, utilizing real-world situations, which are drawn from a military domain.

Information security was recognized with the advent of the first multi-user computer system for sharing information resources, and as we begin the 21st century, this need has become more significant as countries join together to securely share information at the global level [112]. Information sharing in a secure fashion is a daunting challenge, since we must deal with information content that ranges from the simple to the complex (e.g., intelligence reports, financial information, travel records, citizenship records, military positions and logistical data, map data, etc.) in an interoperable environment that is constantly changing. Recently, numerous mandates have emerged to address information sharing. For example, a vital part of U.S. National Security Strategy states, "whenever possible we must seek to operate alongside alliance or coalition forces, integrating their capabilities and capitalizing on their strengths" [82]. This concept is refined further in our Department of Defense Directives [77] and NATO's interoperability and security concerns [80].

The same information sharing and distributed security concerns have driven many of the U.S. Military's automation plans and initiatives. However, **"currently, there is no automated capability for passing command and control information and situational awareness information between nations except by liaison officer, fax, telephone, or loaning equipment"** [80]. As motivation, our interest is in security assurance for information sharing that is required in response to a crisis, e.g., natural disaster (earthquake), humanitarian relief (refugee camps), international incidents (terrorism or spy plane), war (Gulf War), or combat operations other than war (Bosnia). Figure 2 depicts five near simultaneous crises in the European Theater. While these crises each have a different set of counties involved, there must be information sharing between both the countries acting on a crisis and the crises themselves, to effectively manage resources throughout the theater of operations. With every crisis solution, there is an accompanying information sharing risk. To handle a crisis, a coalition – an alliance of governmental, military, civilian, and international organizations – is formed with the primary concern being the most effective way to solve the crisis.

The **Dynamic Coalition Problem**(DCP) can be defined *as the inherent security, resource, and or information sharing risks that occur as a result of the coalition being formed quickly, yet still finding information and resource sharing a necessity for crisis resolution* [119]. The events of September 11 have clearly illustrated the DCP and the difficult issues facing coalitions in information sharing. In the three months following that event, the death toll went from 6,000 to

3,040, and most of the reduction has been traced to names being listed in multiple databases in the direct aftermath (within days and hours of the event), as reported on CNN.com, which for our purposes, corresponds to multiple databases and inconsistencies in reporting and updating information. The lack of management and sharing of information in this regard clearly illustrates one of the main problems facing a coalition in a crisis. In addition, information must be securely shared in an easy, efficient, scalable, and reliable way, to facilitate the tasks of the dynamic coalition without compromise and loss of confidentiality or violations of security policy; this is security assurance.

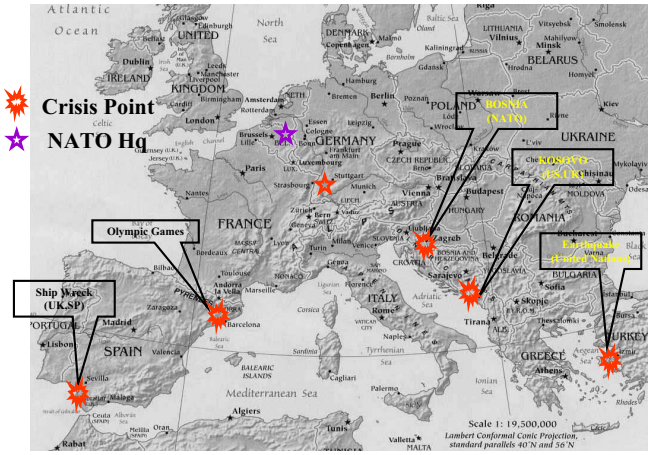


Figure 2: Near Simultaneous Crises.

2.2 Military: Global Command and Control System (GCCS)

Military forces are often used in crisis situations. In the U. S. Military, information or capabilities sharing is difficult even within our own military services (Army, Navy, Air Force, Marines, and Coast Guard). The problems are exacerbated in a situation where a coalition (possibly disparate national interests) is quickly formed [23]. As coalitions become more complex, the risk of security violations increases, which includes risk to classified intelligence information. In some cases, classified information may have to be downgraded temporarily or sanitized for the coalition, but such an act must be done within established security guidelines. The needs of information sharing and security must be balanced in time of crisis. Security mechanisms need to work in joint and combined environments. Joint refers to two or more branches of the Armed Forces (Army, Navy, Air Force, Marines, or Coast Guard) and combined is the participation of military from more than one country. Lines of communication include logistic and informational for all aspects of the crisis. Figure 3 depicts the information requirements between multiple countries to support combined operations [66].

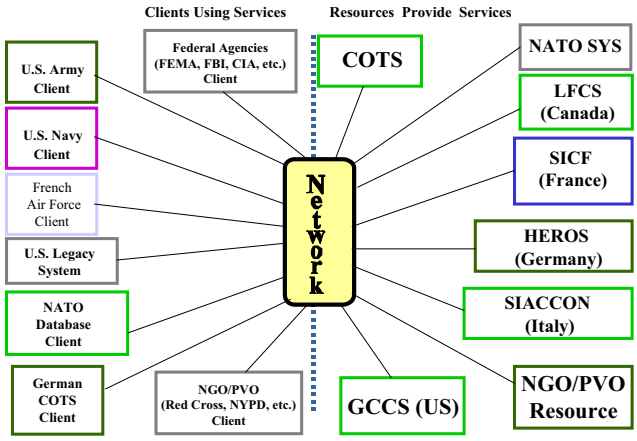


Figure 3: General Coalition Architecture (5 countries).

The information sharing problem is bigger than just classifying information, encrypting data paths, or interoperability; the problem also includes controlling multinational access to resources and adapting to different generations of technology. The current inability to effectively bring international users and their assets (resources) together in a crisis in both an efficient and secure way is very unfortunate, since the actual infrastructure (e.g. localized networks and information resources) can be easily and quickly linked to form an intranet.

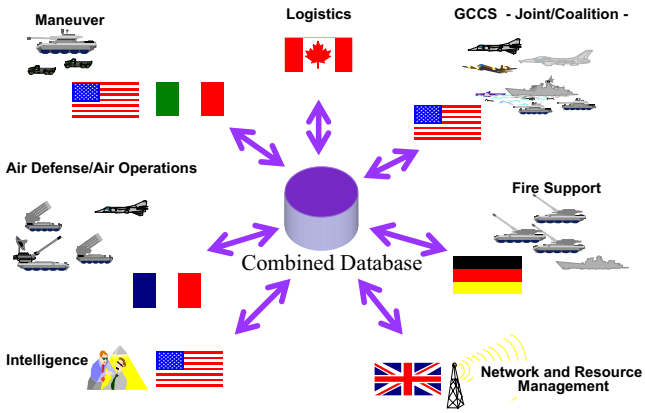


Figure 4: Combined Operations Information Sharing.

Since the U.S. and its military are often called upon in a crisis to supply necessary goods and services, or a unique capability quickly, there must be a system to coordinate this action. The U.S. Military uses the Global Command

and Control System (GCCS) to manage such activities. Unfortunately, GCCS does not satisfy all of the needs of a coalition. In a crisis, the flow of critical information and the access to necessary resources (Figure 4) is as depicted in Figure 5 [19]. At the present time, GCCS is not designed for the international environment. To be useful internationally, GCCS would need to include a security system that could make it a coalition asset while respecting both coalition and U.S. security policies.

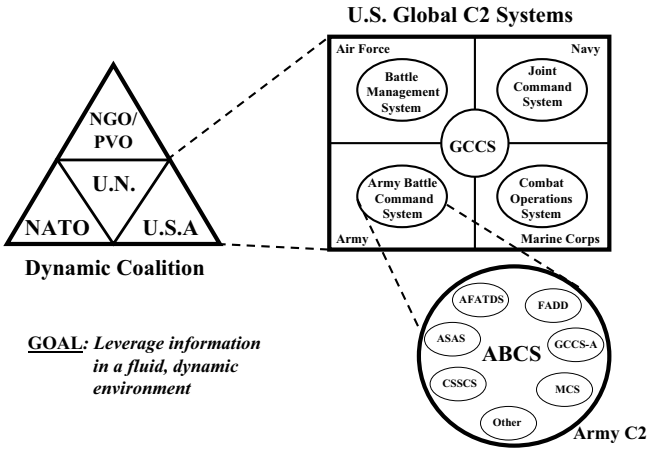


Figure 5: Coalition Artifacts and Information Flow.

GCCS is the automation tool that provides a local U.S. commander operational awareness of the situation (crisis) in near real-time through integrated sets of services as given in Figure 6. GCCS provides information-processing support to planning, mobility, sustainment, and messaging by bringing together 20 separate automated systems in over 625 locations worldwide [58] in a private (physically separate) network. In Figure 6, we present Joint and Component services used to query and change databases or deliver information. The Joint services are used by service members and contain various methods to query the databases of Joint Headquarters and NATO. The Component services allow service members access to individual component (Army, Navy, Air Force, and Marines) command and control systems as depicted in Figure 5. Because GCCS is a U.S. only system on its own private network, security and information sharing issues are different than in a coalition. In order to make the GCCS and other command and control systems acceptable for coalition use, from our analysis, several information sharing and security issues need to be addressed.

First, we believe that user roles can be a valuable technique to support multiple crisis situations like GCCS. Currently, there are no established roles in GCCS, yet individual service members do play specific roles in a crisis. GCCS users have one user profile that includes all of the permissions that allows access to resources within GCCS as determined by their position, supervisor, and

Joint Services:	**a.k.a**
Weather	METOC
Video Teleconference	TLCF
Joint Operations Planning and Execution System	JOPES
Common Operational Picture	COP
Transportation Flow Analysis	JFAST
Logistics Planning Tool	LOGSAFE
Defense Message System	DMS
NATO Message System	CRONOS
Component Services:	
Army Battle Command System	ABCS
Air Force Battle Management System	TBMCS
Marine Combat Operations System	TCO
Navy Command System	JMCIS

Figure 6: GCCS Services.

clearance level. In an international coalition, inconsistencies in organizational structure and security clearances will have to be mapped. Currently, *users have far more access to resources and information than is required for their positions.* Also, a host administrator builds and maintains the user profiles and only receives clearance verification from the security officer. If coalition partners are to share information and resources, there must be a mechanism to restrict access to only necessary information based on a user's role in a crisis. Using static profiles in a crisis, where user requirements are changed or added quickly, is very inappropriate since security is under the control of the host administrator and not the security officer. Using roles can eliminate these profile manipulations by allowing the security officer to change the characteristics of a role or add roles to users dynamically. Then, the host administrator will not need to be involved, since the security officer will have the authority (role permission) to enforce security policy and authorizations. Further, roles can be established dynamically by the security officer to constrain coalition partners to only that information necessary to execute their role. This meets the information security needs of the coalition by using the principle of least privilege.

Second, in analyzing DCP in general and GCCS in particular, *it appears that time controllable access for information security is required.* Typically, when an individual is assigned an organization, the user profile is provided for an indefinite period of time (much longer than a single crisis). For example, users are often assigned to a crisis for a fixed period of time and allowing access before or after that fixed period of time is a security violation. Recall that as the crisis evolves over time, the participants and hence their roles, must also change. Time constraints by role can be used on resources to fix time windows that facilitate database updates or resource allocation. This is the case with GCCS' Joint Operations Planning and Execution System (JOPES). According to policy, Junior

Planners must schedule air movements of equipment by air weeks in advance. If an airlift is required inside this window, only Senior Planners can make adjustments, and that is a different role. Currently, these constraints are not automated in JOPES since roles are not clearly established to assign constraints. However, it is clear that the need to constrain access based on time is an important and needed capability. Third, it seems apparent that *in addition to controlling access to information by time, control of actual values is crucial.*

For example, the common operational picture (COP) is a capability of the GCCS. COP provides a near real-time mapping of all deployed units worldwide. Figure 7 displays a simulated COP screen capture from a command and control system, where military units are placed onto a digital map, by doctrinal unit symbols [24]. The COP itself takes advantage of inputs from different intelligence sources to map both friendly and enemy positions. Certainly, if one does not have the need to know enemy positions, then there should be constraints on that information. In addition, constraints using map coordinates as parameters, can limit the map view to just the crisis area for a specific user, playing a particular user role. This would limit a non-U.S. coalition partner to viewing force positions only in the area of concern, allowing the user to do his/her job without access to potentially damaging information. Clearly, constraints on resources that focus on allowable values can protect sensitive information while still allowing coalition partners to effectively participate in the crisis.

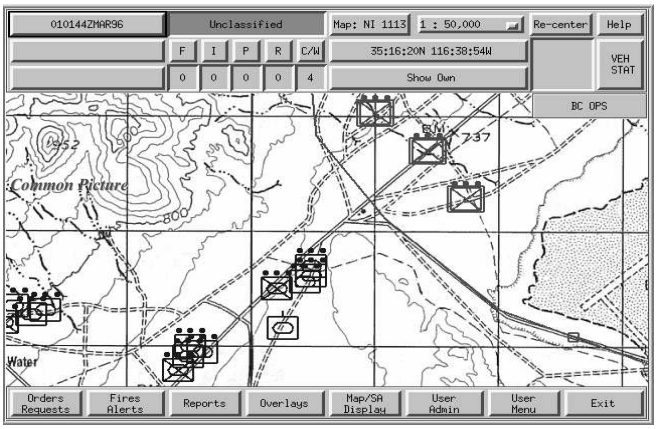


Figure 7: Common Operational Picture.

Finally, in order to manage the GCCS in a joint environment with U.S. forces and multinational partners, the organization and interoperability of coalition assets to yield a distributed environment are a paramount concern. Currently, in multinational crisis situations, *there is no dynamic way to effectively bring users and their automation assets (resources) together in an efficient way*, as depicted

in Figures 2, 3, 4, and 5. Security systems need to allow for quick administration, but still constrain U.S. and non-U.S. users from committing security policy violations. Clearly, users and resources must be federated in a crisis. By using middleware services like JINI or CORBA, resources from coalition partners can be federated with GCCS to make it a more robust and flexible coalition system. In addition, any security solution for DCP must also include an enforcement framework that allows for the management of federated resources and constrain users to security policy limits, limits that also need to be flexible. Our interest in GCCS is to investigate techniques to secure this system in a manner that would make it a coalition asset and respect both coalition and U.S. security policies.

2.3 Access Control: RBAC, DAC, and MAC

Successful information security in DCP will require a detailed and intricate security policy that defines what is considered acceptable and unacceptable with respect to access control (what operations are performed on what resource, by who) and information flow (system behavior with information objects) [35]. Authorization, authentication, and, in particular, enforcement mechanisms, will all be an integral part of any coalition. Discretionary and mandatory access control offer many of the capabilities that are needed by coalitions. In the upcoming discussion, we raise the critical issues related to the access control and their relevance to the DCP.

Discretionary access control (DAC) is a means of restricting access to objects based on the identity of the subject and/or groups to which they belong. The controls are discretionary in that a subject with a certain access permission is capable of passing that permission to any other subject [32]. When information is not sensitive, this type of control is adequate, in that it gives the individual control over distribution and manipulation. In a dynamic coalition, DAC must be carefully administrated to insure that the integrity of information is maintained, and to limit the ability to pass on access restrictions by changing ownership, which is easy to do [110]. For example, when using DAC for DCP, it would be inappropriate for an information owner to give unrestricted access to another user without oversight, since that user could then potentially pass unrestricted access on to another user, without the permission of the original owner. In a coalition, local commanders are not allowed to release information controlled by other owners without the permission of the Defense Intelligence Agency or a Foreign Disclosure officer [119]. Consequently, DAC security policies must be stringently managed and controlled for DCP. Role-based access control (RBAC), a realization of DAC, regulates a user's access to certain resources based on a user role. A user role is a collection of permissions the user needs to accomplish that role. A user may have multiple roles, with each role having a set of permissions. By controlling access using roles and permissions, a security policy can be realized that limits access to the need-to-know information/resources.

RBAC has been consistently touted for its ability and utility in support of non-traditional security applications, where flexibility of usage is crucial [113, 130, 131]. Not only does RBAC provide the best flexibility, it is the best for supporting the concept of least privilege, which is a key concern to the military and coalitions. Least privilege allows for access to only that information which is necessary to accomplish one's tasks [42]. In a dynamic coalition, countries are forced to share information and the least privilege is one way to limit access, and consequently, limit potential compromise or misuse. Using RBAC raises some difficult issues when dealing with coalitions such as: who creates the roles? who determines permissions (access)? who assigns users to roles? are there constraints placed on users within those roles? Currently, the U.S. Military has clearly defined crisis roles for U.S. participants; establishing coalition roles are as much a technical issue as a policy/political issue. There are different RBAC approaches that allow for fine-grained role definition, including our own work [25, 28, 96, 97]. A temporal approach defined in [17] is relevant to DCP due to its changing environment and the shifting of responsibilities. Likewise in [16, 41, 53], the importance of using constraints for identity and authorizations leads to improved granularity on access controls.

Mandatory access control (MAC) is a means of restricting access to objects based on the sensitivity level (classification) of the information object and the formal authorization level (clearance) of the subject [32]. MAC is required when classified information is involved. Classified information is national security information that needs special protection against unauthorized exposure [38]. This is not just sensitive information that an organization might want to protect for personnel privacy reasons, this is information that is considered damaging to national interests. There are three classification levels for information: "Top Secret" - expected to cause exceptionally grave damage to national security; "Secret" - expected to cause serious damage; and "Confidential" - expected to cause some damage [38]. When classified information is used, there are very strict access rules based on the Bell-LaPadula Model of enforcement, which establishes a relationship between classifications of objects and clearances of users and the authorized flow of information [13]. The details of these information flow and security requirements are detailed in [33]. These security requirements apply only to U.S. information. Different countries not only have different security requirements, but also apply different security labels to security objects making translation between sensitivity levels a problem for dynamic coalitions. Furthermore, it will be a tedious and difficult task to carefully define the classification levels for coalition partners, particularly since coalitions will likely include past adversaries. There is a strong likelihood that for coalitions, access control will be accomplished jointly using MAC and DAC. DAC provides discretionality within the boundaries of MAC, and access is only allowed when both DAC and MAC rules are satisfied. Incorporating MAC into RBAC models would allow for the flexibility of RBAC, while observing the strict rules of MAC, providing the best of both approaches

in support of DCP. Strict control and flexibility are very different concepts, but can be brought together and prove useful [91, 109]. Data association and aggregation is a problem with any access control mechanism, particularly MAC. A set of data values seen together may have a higher classification value than taken separately (name, unit, and location), a situation likely to occur in coalitions, and there must be security mechanisms sophisticated enough to handle this type of scenario [108].

2.4 Middlware Security Solutions

To complement our own approach to middleware security, it is relevant to examine the emerging trends for support of security in a middleware setting. Modern middleware platforms like CORBA [87], .NET [75], and J2EE [123] have begun to offer security capabilities. In assessing these three approaches [29], the major difference between the support for security in CORBA (as opposed to its realization in an actual CORBA product, e.g., Visibroker) and security in .NET/J2EE is that the CORBA security specification is a meta-model. As a meta-model, the CORBA security specification generalizes many security models and associated security principles (wide variety of security capabilities at the model level - RBAC, MAC, encryption, etc.) with language independence (not tied to Java, C++, .NET, etc.). .NET and J2EE provide actual security capabilities via their respective runtime environments, and APIs which provide security functionality. The remainder of this section explores the security capabilities of CORBA, .NET, and J2EE.

The CORBA Security Service Specification [87] focuses: confidentiality (limiting access to authorized individuals/programs), integrity (limiting modifications to authorized users), accountability (requiring users to be responsible for their actions) and availability. These aspects are part of the CORBA security reference model, given in Figure 8. In this model, the access control process verifies a subject's permissions (via privilege attributes) against the target objects which are managed via control attributes (grouped as domains) and operations (grouped as rights). Privilege attributes are associated with the user (principal), and the permissions tracked for each principal are: identity (user id), role(s), group(s) that the principal belongs to, security clearance (e.g., secret, classified, etc.), and target objects and operations to which has been granted access. From a complementary perspective, control attributes track the security privileges for each target, e.g., an access control list entry for a target object would track the security characteristics of the object (e.g., security classification), the rights of a target object (i.e., the set of operations that are available for assignment to each principal), and the principals who have been authorized.

In order to define privileges for principals and target objects, a security policy domain is used to represent the scope over which each security policy is enforced, assuming that an organization may have multiple policies. A security policy

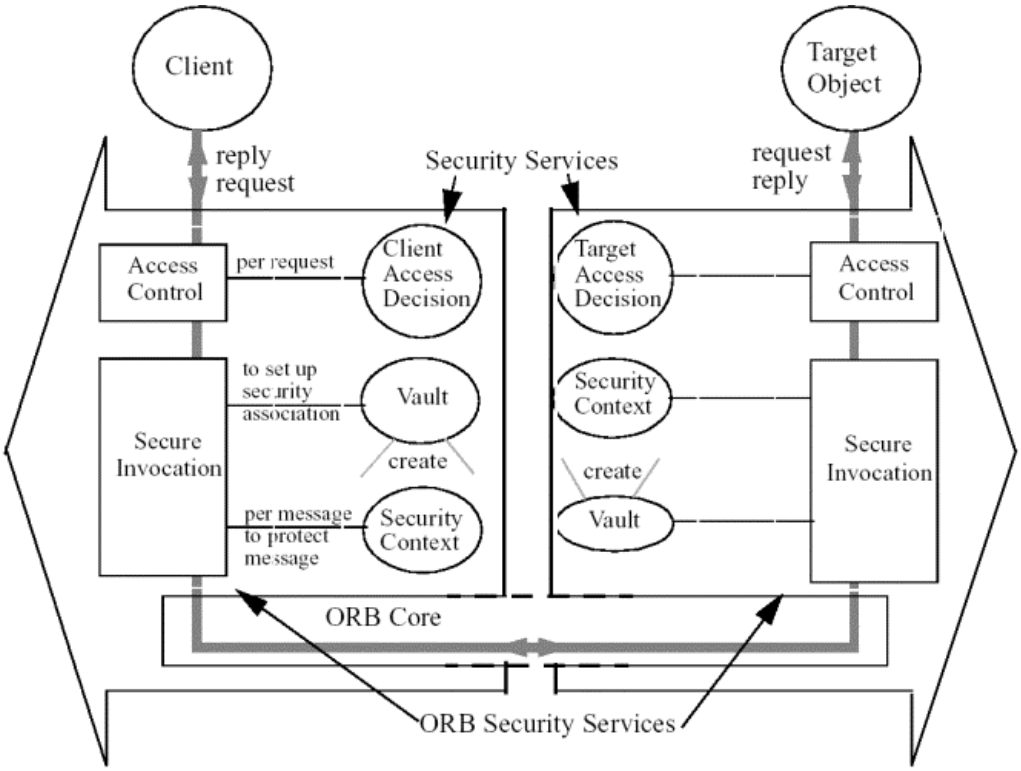


Figure 8: The CORBA Security Model.

domain permits the definition of security requirements for a group of target objects, allowing this group to be managed as a whole, thereby reducing the needed administrative effort. A policy domain hierarchy allows a security administrator to design a hierarchy of policy domains, and then delegate subsets of the hierarchy (sub-domain) to different individuals. As a meta-model, the CORBA security specification is robust enough to realize RBAC, MAC, or any other security model by customizing the concepts of principal, privilege attributes, target objects, control attributes, and policy domains to suit the desired security model nomenclature.

The structural model of security of .NET [75], as represented in Figure 9, consists of the Common Language Runtime (CLR), the Hosting Environment, and the Security Settings. For the Hosting Environment to execute an application, it must provide the code (via assembly - compiler generated code) and its identity (via evidence - proof that is supplied regarding identity) in its interactions with CLR. CLR contains the Security System, which realizes the security policy at enterprise, machine, user, and application domain levels. For an actual application, the different parameters related to security must be set within the Security System, as shown by the input from the Security Settings box in Figure 9, to establish the security at one or more policy levels. For execution to

occur within CLR, the assembly is used to identify the required permission set (e.g., allowances given to a piece of code to execute a certain method) and be provided with evidence from the Host to the Security System.

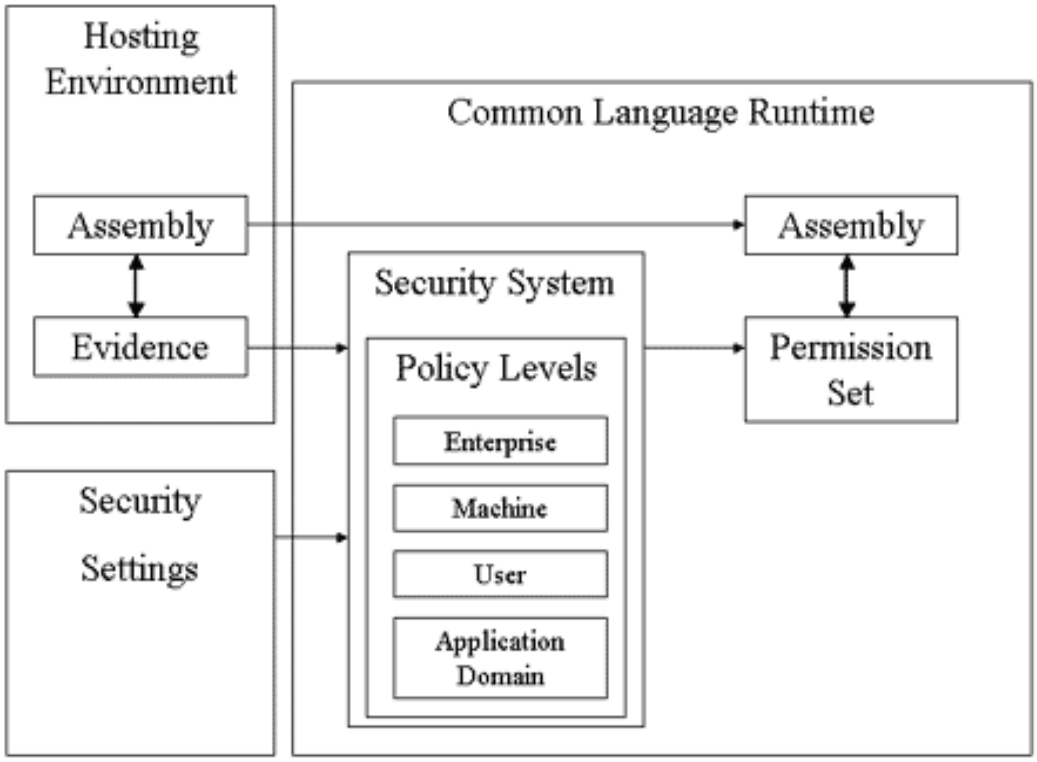


Figure 9: The .NET Security Structural Model from MSDN 2003.

Code-based access control, a part of CLR's security, dictates the situations where access by a code segment to a resource is permitted (prevented). The determination of what a piece of code is allowed to do is decided by evidence based security, permissions, and a security policy. During execution, the CLR reviews evidence of an assembly, determines an identity for the assembly, and looks up and grants permissions based on the security policy for that assembly identity (Open Web 2002). Evidence based security determines the origin(s) of an assembly. At runtime the CLR examines the meta-data of an assembly for the origin of the code, the creator of the assembly, and the URL and zone (e.g., Internet, LAN, local machine, etc.) of the assembly.

The successful verification of evidence, leads to the permissions of code and code segments, which is the ability to execute a certain method or access a certain resource. An assembly will request permissions to execute, and these requests are answered at runtime by the CLR, assuming that the assembly has provided apropos evidence. If not, CLR throws a security exception and an assemblys request is denied. Since numerous different permissions can be requested, permissions

are grouped into sets. Permissions and permission sets in .NET are similar to respectively, privilege/control attributes and domains in CORBA. As such, it is possible to establish permissions for MAC classification levels for code and resources, with the access permitted or denied based on the domination of the code's classification over the resource's classification.

Lastly, the grouping of assemblies based on different criteria establishes different security policies for different code groupings [75, 88]. In .NET there are the three different security policies that are supported: enterprise level for a cohesive and comprehensive policy for the entire enterprise; machine level for different policies for different machines; and user level to capture individual responsibilities. The .NET framework provides the means to organize security policy groups of assemblies into hierarchical categories based on the identity that the CLR determines from the evidence. Once related assemblies have been grouped and categorized, the actual security policy can be specified as permissions for all assemblies in a group. RBAC in .NET extends the policies and permissions concepts of code-based access control to apply to a user or role. .NET uses role-based security to authenticate an identity and to pass on that identity to resources, thereby authorizing the users playing roles, access to resources according to policies and permissions.

Security in the Java 2 Enterprise Edition (J2EE) [123] focuses on its ability to keep code, data, and systems safe from inadvertent or malicious errors. In Figure 10, the compilation of Java code creates bytecode, whose execution involves the class loader (with bytecode verifier), the Java class libraries (APIs), and the Java virtual machine (JVM). The JVM manages memory by dynamically allocating different areas for use by different programs, isolating executing code, and performing runtime checks. The block labelled Runtime System as shown in Figure 10, contains the Security Manager, Access Controller, and other features that all interact to maintain security of executing code. Security considerations in J2EE are important for both applications and applets, but applets are of particular concern for security, since they represent remote code that is brought in and executed on a local machine. To control applet behavior, Java uses a sandbox, which forces downloaded applets to run in a confined portion of the system, and allows the software engineer to customize a security policy. The Security Manager enforces the boundaries around the sandbox by implementing and imposing the security policy for applications. All classes in Java must ask the security manager for permission to perform certain operations. Java only has two security policy levels, one for the executing machine, and one for the user. Each level can expand or restrict on all of the permissions of another level, and there can be multiple policy files at each level.

Permissions in Java are determined by the security policy at runtime, and are granted by the security policy based on evidence. The evidence that Java looks for is a publisher signature and a location origin. Permissions are also grouped into protection domains (similar to security policy domains in CORBA and to

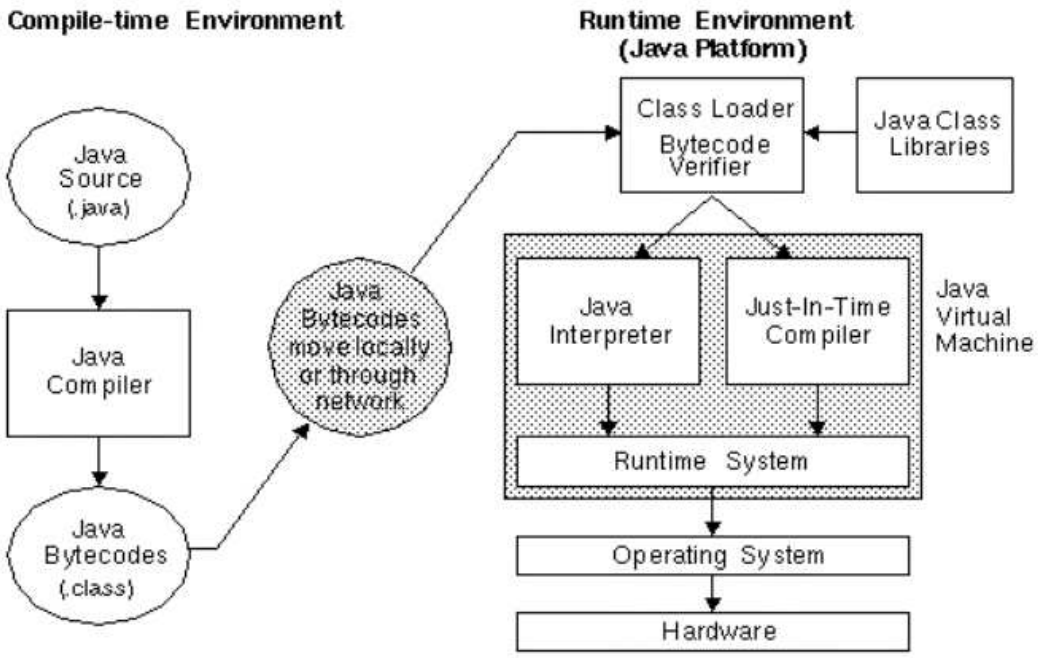


Figure 10: The Java 2 Platform - Compile and Execute.

security policy files in .NET) and associated with groups of classes in Java in much the way they are grouped into permission sets and associated with code groups in .NET. However, in Java, MAC is not automatic; it requires programmatic effort by the software engineer.

In support of RBAC, J2EE uses the Java Authentication and Authorization Service (JAAS), which implements a Java version of the Pluggable Authentication Module framework. Using JAAS, software engineers are allowed to modify and then plug-in domain/application specific authentication modules [31]. JAAS currently supports authentication methods including Unix, JNDI, and Kerberos, akin to OS level security. JAAS can only provide limited impersonation authentication because the user identity is different for the application and OS levels. User access checking can be done both declaratively and imperatively within different components of J2EE.

2.5 Related Work

The work presented herein aligns with many ongoing initiatives to solve information security and interoperability issues. One Department of Defense and NATO effort is the Command Control Systems Interoperability Program (C2SIP)

to bring NATO forces together using a database engine that accepts any NATO country formats [80]. The Air Force Research Laboratory in conjunction with Verdian is working on a comprehensive information tagging and release policy called Secure Information Releasability Environment [119]. There are products like e-Portal and Multi-domain Dissemination System, which concentrate on sensitive information access using secure transmission [119]. In addition, there are systems that use firewall technology to create secure network connections between hosts on any unclassified network [60]. All of this work is relevant for different aspects of DCP, but none address the critical issues of federation of resources/users, and the availability and access of resources/information in a secure fashion to support dynamic coalitions.

CORBA contains individual security services for confidentiality, integrity, accountability, and availability, but there is no cohesive CORBA service that ties these and other concepts (authorization, authentication, and privacy) together into a cohesive security solution. There has been significant progress in distributed authentication in Kerberos [81] and Cheron [43], security metric analysis and design [101], Internet security via firewalls [89], role-based access control on Web-based intranets [112], and security for mobile agents [124, 135]. All of these have interesting features that are part of an overall middleware security assurance solution.

Chapter 3

The RBAC/DAC/MAC Security Model

Our main objective in this chapter is to define our unified role-based, discretionary, and mandatory access control (RBAC/DAC/MAC) security model using an enforcement framework for software artifacts interacting via middleware [67, 96, 97]. Our approach focuses on which methods of APIs can be invoked based on the responsibilities and security classification level of a role, the security clearance level of the user, and the values (parameters), time, and classification level of the method; all of which must be *satisfied* in order for the invocation to successfully proceed. In order to achieve this capability, we provide assurance that validates the method invocation by a user playing a role in a number of different ways. First, we must guarantee the timeframe of the invocation, to insure that a user (with one lifetime) playing a role (with another lifetime) can invoke a method (yet another lifetime) at the current time. In fact, this assurance guarantee is partially checked when the security policy is being defined, and must be rechecked at execution time (i.e., actual method invocation at a specific time). Second, we must guarantee that the method invocation does not violate MAC domination. Specifically, we demonstrate that our approach satisfies the *Simple Security Property* [13] for invocations of read-only and read-write methods, and the *Simple Integrity Property* for invocations of read-write methods. Third, we must insure that roles that are delegated from user to user satisfy both domination and lifetime in order for the delegation to occur, which will be considered in Chapter 4. Finally, for a security policy defined using our RBAC/DAC/MAC model, we will provide in Chapter 5, a series of theorems that insure a degree of *safety* (nothing bad will happen when a user playing a role invokes a method) and *liveness* (all good things can happen when a user playing a role invokes a method). When combining RBAC, DAC, and MAC, MAC requirements take priority and the enforcement mechanisms must support this requirement.

In the remainder of this chapter, we concentrate on the RBAC and MAC capabilities, deferring DAC until Chapter 4. Our initial emphasis is on establishing the assumptions for an abstract middleware model in Section 3.1. Next, the main

portion of this chapter is presented, namely, the RBAC and MAC capabilities of the unified RBAC/DAC/MAC security model in Section 3.2. Section 3.2 explores the entire privilege definition process for roles authorized to invoke methods at certain times and under specific values, and for users authorized to roles. In addition, we motivate security assurance guarantees at a conceptual level. Once all of the concepts and modeling constructs of the security model have been presented, we examine a detailed set of its assumptions in Section 3.3. Finally, Section 3.4 discusses related research as compared to our own efforts.

3.1 An Abstract Middleware Model

In this section, we present our assumptions for an abstract middleware model that underlies our approach. Recall that we are assuming that distributed computing applications are constructed from software artifacts (legacy, commercial-off-the-shelf (COTS), government-off-the-shelf (GOTS), database, and new client/server applications), which require stakeholders (i.e., software architects, system designers, security officers, etc.) to architect and prototype solutions that facilitate the interoperation of new and existing applications in innovative ways. The interaction of software artifacts with clients occurs via middleware (e.g., DCE [92, 106], CORBA [86, 133, 137], DCOM [74], JINI [6] .NET [75, 31], etc.). Our specific interest is in distributed applications that plug-and-play, allowing us to plug in (and subtract) new "components" as needs, requirements, and even network topologies change over time. We refer to these plug-and-play software artifacts as *resources* which are comprised of *services*, which each service contains a set of *methods* that are published. The resources, their services, and their methods, interact with clients across the network, and as such, comprise the distributed resource environment for the distributed application.

To support the interaction of software artifacts with clients and other artifacts, we assume the presence of a *lookup service* as supported in middleware such as CORBA, JINI, DCOM, etc., which is a clearinghouse for resources to register services (of methods) and for clients to find services (and their methods). A lookup service allows stakeholders to construct distributed applications by federating groups of users (clients) and the resources that they require [6, 28]. With any lookup service, it is a key assumption that the services are registered or they will not be readily available and when registered, are only available through the lookup service. Resources *register* services provided for use by a person, program (client), or another resource, including a computation, a persistent store, a communication channel, a software filter, a printer, and so on. Figure 11 illustrates the interactions of a lookup service, client, and resource in a distributed resource environment (DRE). In Figure 11, the CourseDB resource joins the lookup service, by registering its services (methods). Each service consists of methods (e.g., AddCourse) that are provided for use by clients (and other resources). One limitation of this process is that once registered, all of the resource's services

are available to all clients, i.e., there is no security. Figure 11 also illustrates the steps that are taken when a client requests a service (AddCourse) from a resource (CourseDB).

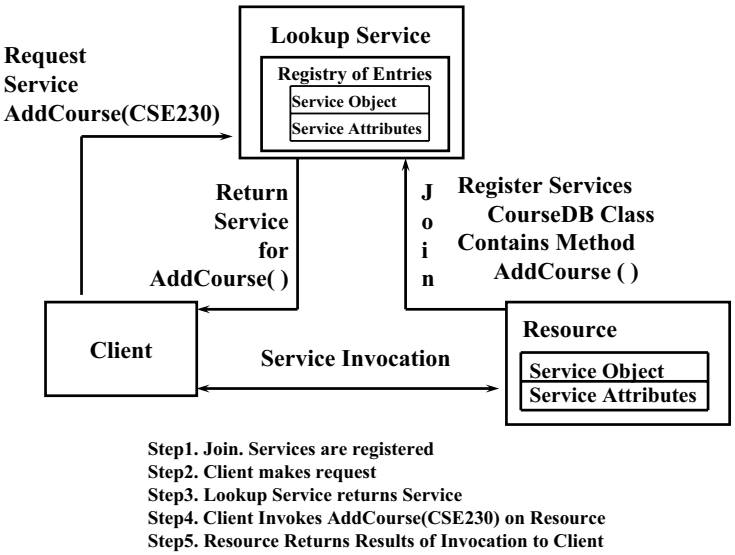


Figure 11: Join, Lookup, and Invocation of Service.

There are several other relevant assumptions we make about lookup services to maintain consistency in our security model, which have been captured in detail, Tables 1 to 4 of Section 3.3, and which we highlight. First, we assume that the lookup service has adequate protection against intrusion that would cause failure or subversion of the security model and enforcement framework; once registered with the lookup service, there is no other way to access resources, services and methods other than using the lookup service; and all services and methods can be registered and provide concrete APIs for use by client applictions and other artifacts (resources). Second, for our purposes, we assume that the resource dictates the availability of registered resources in terms of time. For example, in JINI, it is possible to register services for a given time period (milliseconds) and for them to continue to be available the resource must re-register the services before expiration. There may be situations where a client application may have access to a method (of a service of a resource) obtained at time X, and tries to invoke the method at time Y (Y later than X) when the method is no longer available. We assume that the invocation fails since the method is no longer registered with the lookup service. Thus, it is not an issue from a security perspective.

Our goal is to leverage the infrastructure of a distributed resource environment, to support and realize RBAC, DAC, and MAC. In such a setting, we

propose a software architecture with specialized security resources that interact with non-security resources and clients, to authorize, authenticate, and enforce security for a distributed application in a dynamic fashion. We propose a software architecture and security specific resources to support authorization (grant and revoke privileges to clients based on role), authentication (verify the identity of clients), and enforcement (insure that a client only uses those services of a resource to which it has been authorized). Our fundamental objective is to propose a security solution for distributed applications built around the concepts of clients and resources, operating within a distributed resource environment. Within this context we provide RBAC, DAC, and MAC services that limit the programmatic changes to a resource (software artifact), allowing the resource to dynamically discover security privileges from security resources when a client attempts access. This enforcement framework will be discussed in Chapter 6. Our objective for these assumptions is to set the context for the RBAC/DAC/MAC security model, since a discussion on the modeling features (design time) inevitably involves the underlying enforcement mechanisms (runtime).

3.2 RBAC and MAC Modeling Capabilities

This section reviews the RBAC and MAC modeling capabiliaties of our unified RBAC/DAC/MAC security model, which was initially formalized in [96, 97], and has been extended with delegation in [67] (see Chapter 4). To set the context for Chapter 5 on security assurnace, where relevant, we discuss conceptual assurance guarantees for the various modeling constructs. We divide the discussion of the non-DAC features of the security model into four parts: Section 3.2.1 reviews core concepts of a *lifetime* which is an interval of access and of a MAC *sensitivity level* which is a tag of the security concern of the data; Section 3.2.2 formalizes the abstract middleware model as discussed in Section 3.1, defining a distributed application of artifacts, a resource (artifact), its services, and their methods; Section 3.2.3 explores authorization issues involving methods to roles, with associated model constructs to capture all facets of the authorization; and, Section 3.2.4 mirrors Section 3.2.3, but focuses on authorization of users to roles.

3.2.1 Lifetimes and MAC Security Levels

A *lifetime* resprents a discrete time interval of access within the security model, and is an important property since both the lifetime itself, and the intersection of multiple lifetimes establishes the availability of access of roles to methods, users to roles, etc. Throughout Section 3.2, and in Chapter 4, every major modeling construct (e.g., user role, user, resource, service, method, etc.) will have lifetimes, and the security processes (e.g., authorization of role to method, authentication of user, delegation of authority, etc.) must all respect the lifetimes of the constructs in order to succeeed. Formally, we can define:

Definition 1: A *lifetime, LT,* is a time interval with start time (st) and end time (et), [st, et], et > st, and st/et is of the form (mo., day, year, hr., min., sec.). Concepts of LTs X and Y are: $X \triangleright Y$ means $Y.st \geq X.st$ and $Y.et \leq X.et$; $X \triangleleft Y \equiv Y \triangleright X$; If $ST = max\{X.st, Y.st\}$ and $ET = min\{X.et, Y.et\}$, then $Y \cap X$ is $\emptyset$ if $ET \leq ST$ or [ST, ET] if $ET > ST$; and $LT = [ct, \infty]$ is current time (ct) onward.

From an assurance perspective, LTs can guarantee that definition and access to privileges will always satisfy time limits.

Figure 12 illustrates the lifetime of a subject X (X.LT) could be compared to the lifetime of object Y (Y.LT). As given, the maximum amount of time Y is available to X is the intersection of the lifetimes, $X \cap Y$. Lifetimes in combination establish availability and provide assurance that an expired subject or object will not be accessed. Addition, the associations of lifetimes is not absolute, but must also be considered against the actual time. For example, in Figure 12 we include the current time (ct), which can be a factor to determine the limitations of the overlaps. When the current time is before the overlap starts, the entire overlap is valid. When the current time is after the overlap ends, then the overlap is irrelevant. When the current time occurs within the overlap interval, then the current time represents the lower limit of available access time. Using lifetimes helps establish a "least privilege" policy which means a user only has access to what is necessary to do their job at a particular time, and nothing more.

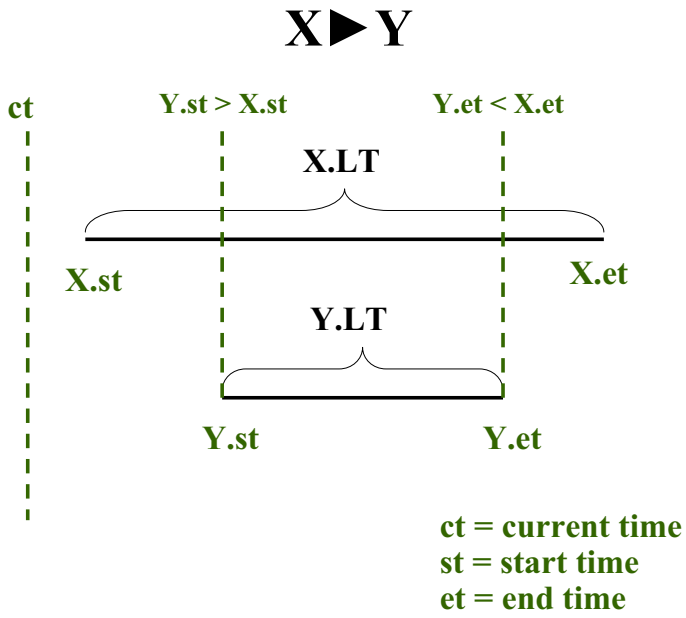


Figure 12: LT of X compared to LT of Y.

There are several relevant assumptions we make about lifetimes to maintain consistency in our security model, as captured in Tables 1 through 4 (see Section

3.3). In particular, we assume there is no delay time between the time that a current time is requested by a global clock and the time that a requestor receives the current time. In reality, there is always a delay (e.g., processing delay, network transmission delay, etc.) that is involved with asking for a current time; for analysis purposes of our security model, we assume this delay is zero (i.e., the current time is returned instantaneously). We also assume that a lifetime cannot be assigned if the end time (et) is less then or equal to the current time (ct) or if the only overlap in assignment is when et = st.

In support of MAC, we define concepts that allow the different modeling constructs (e.g., roles, users, resources, services, methods, etc.) to be tagged with a level that indicates the degree of security. Formally, we define:

Definition 2: MAC concepts are: *Sensitivity levels*, $SLEVEL = \{U, C, S, T\}$ with unclassified (U), Confidential (C), Secret (S), and Top Secret (T) forming a linear order: $U < C < S < T$; *clearance (CLR)*, the SLEVEL given to users; and, *classification (CLS)*, the SLEVEL given to roles, methods, etc.

From an assurance perspective, MAC will be used to guarantee the Simple Security and Integrity Properties [13, 18]. Note that we use the terms sensitivity level and security level interchangeably. Note also that the term SLEVEL is used generically, and can be applied to any organization; we use U, C, S, and T to conform to MAC requirements. If MAC is not required, the default level is U, which effectively turns off the sensitivity levels. Organizations can define their own sensitivity levels, as well.

There are several relevant assumptions we make concerning MAC to maintain consistency in our security model, as captured in Tables 1 through 4 in Section 3.3. In particular, we assume MAC is necessary for all government or government interfacing information systems, so that MAC constraints are checked with the invocation of every action (of a subject to objects in classic MAC, and as we will see shortly, of roles to methods in our approach) to increase assurance. We also assume only four security levels, but any number can work with a linear ordering. We do not support a security level hierarchy at this time.

3.2.2 Resources, Services, and Methods

Definitions 3 to 6 are for a distributed application of resources, services, and methods, with LTs (availability of resource/service/method) and CLSs (SLEVEL of a resource/service/method). These definitions are key to understanding how we secure a distributed application. Users should be limited to only what is necessary to do their job and nothing more. This concept is commonly referred to as "least privilege." In a distributed application, a user may have access to a resource, but access to services and methods will be limited to only what is necessary.

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

Definition 4: Each *method* $M_{ijk} = [M_{ijk}^{Name}, M_{ijk}^{LT}, M_{ijk}^{CLS}, M_{ijk}^{Params}]$ for $i = 1..m, j = 1..n_i, k = 1..q_{ij}$, of S_{ij} of R_i has name M_{ijk}^{Name}, LT M_{ijk}^{LT} (default [ct, ∞]), $M_{ijk}^{CLS} \in SLEVEL$ (default U), and M_{ijk}^{Params} parameters.

Definition 5: Each *service* $S_{ij} = [S_{ij}^{Name}, S_{ij}^{LT}, S_{ij}^{CLS}]$ for $i = 1..m, j = 1..n_i$, of R_i has name S_{ij}^{Name}, LT $S_{ij}^{LT} = [min\{M_{ijk}^{LT.st}\}, max\{M_{ijk}^{LT.et}\}]$ $\forall k = 1..q_{ij}$, and $S_{ij}^{CLS} = min\{M_{ijk}^{CLS} | k = 1..q_{ij}\}$.

Definition 6: Each *resource* $R_i = [R_i^{Name}, R_i^{LT}, R_i^{CLS}]$ for $i = 1..m$, has name R_i^{Name}, LT $R_i^{LT} = [min\{S_{ij}^{LT.st}\}, max\{S_{ij}^{LT.et}\}]$ $\forall j = 1..n_i$, and $R_i^{CLS} = min\{S_{ij}^{CLS} | j = 1..n_i\}$.

Note that names, LTs, CLSs, etc., are set when the resource, services, and methods are registered by a software artifact with the security enforcement framework middleware (please see Chapter 6). For our purposes, our model captures these characteristics, and in Definitions 4, 5, and 6, the associations between LTs and CLSs are an integral part of the security model. *From an assurance perspective, when a resource registers itself, we want to guarantee the dependencies that exist among the LTs (resource LT spans its services' LTs; service LT spans its resources' LT), and the minimality of CLS - least secure (resource CLS is minimum of its services' CLS; service CLS is the minimum of its methods' CLS).*

To illustrate the concepts in Definitions 3 to 6, we utilize two services (Joint and Component) of a Global Command and Control System (GCCS) Resource (see Section 2.2) as depicted in Figure 13. The GCCS is a military command and control systems which provides military commanders with near real-time operational awareness during a crisis. Using this example, we can illustrate both the LTs (Figure 14) and MAC levels (Figure 15). Figure 14 depicts an example use of lifetimes for the GCCS. Notice that the lifetime of the resource (Resource.LT) establishes limits on the lifetimes for the services (Service.LTs), which in turn limits the lifetimes for each method (Methods.LT). Lifetimes provide the assurance that a method will be neither authorized nor invoked outside of a very strictly enforced time interval.

In current military information systems, it is the maximum classification of any element of the system (i.e., object, file, hardware, cryptographic key, etc.) that determines the overall classification of the system. This approach is good for security assurance in that it maximizes safety (no bad things can happen). Unfortunately, a consequence is reduced liveness (all good things can happen), in that only users with the system-highest clearance can utilize the system. Users with a lower than system-highest clearance would not have system access. In

GCCS Resource with Two Services

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

Figure 13: Resource with Services/Methods.

the real world, this means redundant systems and/or a larger number of users with system-high clearances are needed to satisfy processing needs. Both redundant systems and increased numbers of high-level users are added security risks. Referring to our example in Figure 15, the GCCS would carry a TOP SECRET classification based on one method, the NATO Message System. Again, this means that all users would require a TOP SECRET clearance in order to use GCCS. On the other hand, limiting access to functionality based on MAC principles would preserve safety and maximize liveness, along with eliminating the need for redundant systems or increased numbers of folks with system-high clearance levels. These are a few of the benefits of our approach which emphasize our security assurance contribution.

There are several relevant assumptions we make about distributed applications, resources, services, and methods to maintain consistency in our security model, as captured in Tables 1 through 4 (see Section 3.3). In particular, we assume distributed applications are constructed from publicly available artifacts and users can execute the permissions of their role using Application Programmer Interfaces (API) or public methods. Once registered with our security service, there is no other way to execute the API or public method for the distributed application.

Resource.LT **Service.LT** **Methods.LT**

$$R_i^{LT} \triangleright S_{ij}^{LT} \triangleright M_{ijk}^{LT}$$

	LT.st	LT.et
GCCS Resource	1 Jan 02	31 Dec 02
Joint Service with Methods:	1 Jan 02	1 Dec 02
Weather ();	1 Jan 02	1 Jul 02
VideoTeleconference ();	2 Jan 02	5 Feb 02
JointOperationsPlannning ();	7 Feb 02	1 Nov 02
CrisisPicture ();	5 July 02	1 Dec 02
Component Service with Methods:	1 Jan 02	5 Jul 02
ArmyBattleCommandSys ();	1 Feb 02	4 Jul 02
AirForceBattleManagementSys ();	5 Mar 02	5 Apr 02
MarineCombatOpnsSys ();	1 Jun 02	4 Jul 02

Figure 14: Lifetimes for Resource/Services/Methods.

3.2.3 Authorization: Roles to Methods and Users to Roles

In this section, we focus on the concepts that are necessary to support the authorization of user roles to methods as part of the security definition process. Specifically, Definitions 7 to 18 involve the privilege specification process for users and user roles, and the granting of methods to roles, and roles to users. At this time, our model does not include role hierarchies; prior research had user-role definition hierarchies [25], and the inclusion of hierarchies is on our agenda for future work.

The role is the building block of RBAC, and represents the set of necessary responsibilities to execute a specific job or function. The roles for a distributed application are collected into a list to represent the available roles a user can assume. This makes for a natural constraint to what a user can do and provides the basis for security assurance. The key to maintaining a RBAC system is controlling what goes into the role (i.e., must only allow what is necessary to do the job) and controlling who can have what role (i.e., maintain the least privilege principle). Formally, we define:

Definition 7: A *user role* $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$, represents a set of responsibilities, and has name UR^{Name}, LT UR^{LT} (default [ct, ∞]), and $UR^{CLS} \in SLEVEL$ (default U).

Definition 8: *A user-role list,* $URL = \{UR_i | i = 1..r\}$, contains the r roles (Definition 7) for DAPPL.

(C) GCCS Resource C= min {Service CLSs}	
(S) Joint Service with Methods S = min{Method CLSs}	**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)LogisitcsPlanningTool (Token, CrisisNum);	**LOGSAFE**
(S)DefenseMessageSystem (Token);	**DMS**
(T)NATOMessageSystem (Token);	**CRONOS**
(C) Component Service with Methods: C = min{Method CLSs}	
(S)ArmyBattleCommandSys (Token, CrisisNum);	**ABCS**
(S)AirForceBattleManagementSys (Token, CrisisNum);	**TBMCS**
(S)MarineCombatOpnsSys (Token, CrisisNum);	**TCO**
(C)NavyCommandSystem (Token, CrisisNum);	**JMCIS**

Note: Access Classification Precedes Each Entry.

Figure 15: Resource/Services/Methods with SLEVELs.

From an assurance perspective, URs have specific LTs that will be utilized for checking the "when" of access (i.e., can a user invoke a method at current time) and CLSs that will be instrumental in checking the "if" of access (i.e., does a UR's CLS dominate a method's CLS). Representative URs for GCCS are shown in Figure 16, with the name, LT, and CLS given for each role. CDR_CR1 means commander of crisis 1, JPlanCR2 means joint planner of crisis 2, and so on. From a privilege perspective, URs can be granted access to methods which have CLSs at or below the role's CLS. Figure 16 has sample URs for CDR_CR1, JPlanCR1, etc., with LTs (using date) and CLS (T/S/C/U).

There are several relevant assumptions for roles to maintain consistency in our security model, as captured in Tables 1 through 4 (see Section 3.3). First, we assume consistency within roles with respect to mutual exclusion and lifetimes. Roles should not be created with permissions that violate lifetimes (we check this at design time) and that allow for conflict. A classic example of conflict is one role having both purchase authority and approval authority, which is not good practice in any organization. Second, we assume that within federated groups, all users have roles, all roles are clearly defined, and most importantly, a user can play only one role at a time. All other assumptions will be discussed in detail in Section 3.3.

Roles as presented are intended to characterize common responsibilities that may be appropriate for different users. Ultimately, the authorization and authentication of *users* are needed to define who can use the application. Like roles,

```
Roles: [CDR_CR1, [01dec02,01dec03], T]
       [JPlanCR1, [01dec02,01jun03], S]
       [JPlanCR2, [01jul01,01sep03], C]
       [ArmyLogCR1, [10dec02,01mar03], S]
       [ArmyLogCR2, [01jul03,01aug03], C]

Users: General DoBest:   [DoBest, [ct, infinity],  T]
       Colonel DoGood:   [DoGood, [01dec02,01jun03], T]
       Major DoRight:    [DoRight, [01dec02,01jan03], S]
       Major CanDoRight: [CanDoRight, [01jan03,01feb03, T]

URAs:  [JPlanCR1, CrisisPicture, [ct, infinity],true]
       [JPlanCR1, ArmyBattleCmdSys, [10dec02,16feb03], true]
       [ArmyLogCR1, CrisisPicture, [10dec02,16feb03],
                                   Grid1 > NA20 AND Grid2 < NC40]
       [ArmyLogCR2, LogPlanningTool, [10dec02,16feb03], CrisisNum=CR1]
```

Figure 16: Sample Users, User-Roles, and User-Role Authorizations.

users will be limited by a lifetime of the authorization, but unlike roles, users will have clearances to determine their MAC privileges. Formally, we define:

Definition 9: A *user*, $U = [U^{UserId}, U^{LT}, U^{CLR}]$ is an entity accessing a client, and has a unique U^{UserId}, LT L^{LT} (default [ct,∞]), and $U^{CLR} \in SLEVEL$ (default U).

Definition 10: A *user list,* $UL = \{U_i | i = 1..u\}$, contains the u users (Definition 9) for DAPPL.

From an assurance perspective, users have specific LTs for checking the "when" of access (i.e., can a user play a role at current time) and CLRs for checking the "if" of access (i.e., does a user's CLR dominate a UR's CLS). Representative users, General DoGood, Colonel DoBest, etc., for GCCS are shown in Figure 16, with the name, LT, and CLR given for each user.

There are relevant assumptions we make about users to maintain consistency in our security model, these are captured in Tables 1 through 4. Specifically, we assume a user can play only one role at a time during a session, but can change roles during a session. Upon login, authorization and authentication are made for all authorized roles.

The next three definitions involve different ways that authorizations can be constrained. Specifically, we constrain: the values under which a method can be invoked; the allowable time values for various authorizations; and the conditions under which a MAC subject dominates a MAC object, where subject and object vary in our model. The first type of constraint involves the *signature* of a method, which is defined as the method name, return type, and parameter names and

their types. In any method invocation, ideally from a security perspective, we would like to control two facets: 1. the actual parameter values under which the method is invoked, and 2., the valid values that can be returned at the end of the invocation. In this research, we concentrate on the first facet; the second will be the subject of future research. One critical assumption we make regarding the methods of the services is that all of the method's parameters are *value parameters*, i.e., we assume that the method does not return any value back to the caller via the parameters, only through the return of the method. Thus, our security control of the method invocation will be on/off, and not involve data being sent back through the parameters. Future work will consider controlling access to return values; for now, that is outside the scope of this dissertation. Formally, we define:

Definition 11: A *signature constraint, SC,* is a boolean expression on M_{ijk}^{Params}, for $i = 1..m, j = 1..n, k = 1..g_{ij}$, to limit allowable values.

SCs limit the conditions under which a method may be invoked. For example Figure 17, an ArmyLogCR1 UR can use method CrisisPicture from the Joint Service, but needs an SC (Grid1 < NA20 AND Grid2 < NC40) to limit the picture view. Thus, methods are off/on based on a specialization of the parameter/return values.

GCCS Methods Used:

CrisisPicture (Token, CrisisNum, Grid1, Grid2);
ArmyBattleCommandSys (Token, CrisisNum);

Example: **Signature Constraint, SC**
Method: CrisisPicture
Grid1 < NA20 and Grid2 < NC40

Example **Time Constraint ,TC**
Method: ArmyBattleCommandSys
10dec02 < date < 16feb03

Figure 17: Signature and Time Constraint for GCCS.

The second type of constraint limits a variety of authorization actions based on type, and also limits the runtime (invocation or delegation). Lifetimes limit actions of individual constructs, e.g., a lifetime of a method or a lifetime of a

role. Time constraints limit the association between constructs, i.e., a role may exist for 60 days, a method may exist for 90 days, and have a 30 day overlap with a role, but within that overlap, a time constraint can further limit access to 15 days. Formally, we define:

Definition 12: A *time constraint, TC,* is a LT (default [ct,∞]), and is utilized as follows: LTs of UR and method constrain the method assignment; LTs of UR, method, and user, constrain the method invocation; LTs of UR and user constrain the user authorization to the role; LTs constrain the time span of a delegation.

The JPlannerCR1 has a TC on ArmyBattleCommandSys Figure 17 (10dec02 < date < 16feb03). In 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: (10dec02 < date < 16feb03).

The third type of constraint is for MAC to support CLR and CLS. As discussed in Definition 2, the CLS level is assigned to individual methods that comprise each service of a resource. 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 [13].

Definition 13: A *mandatory access control constraint, MACC,* is the *domination* of the SLEVEL of one entity over another, e.g., U's CLR ≥ UR's CLS or UR's CLS ≥ M's 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. Since the CLS level of a role or a method may have changed between the time that a client was authorized and attempts a method invocation, this run-time check is required.

From an assurance perspective, we seek to provide guarantees that a user playing a role can invoke a method limited by parameter values (SC - Definition 11), method availability (TC - Definition 12), and the Simple Security and Integrity Properties applied by a user against a role invoking a method (Definition 13). For example, an ArmyLogCR1 UR can invoke method CrisisPicture (see Figure 16) limited by the SC ($Grid1 > NA20$ AND $Grid2 < NC40$), while a JPlanCR1 UR can have a TC of [10dec02, 16feb03] on the ArmyBattleCmdSys method of the Component service. MACC verifies if the user (with a CLR level) playing a role (with a CLS level) can invoke a method (with a CLS level) at ct.

In addition to the method's parameters being *value parameters*, mentioned earlier in this section, there are several other relevant assumptions we make about TCs, SCs, and MACCs to maintain consistency in our security model. These are

captured in Tables 1 through 4 (see Section 3.3). First, we assume all method interactions among resources satisfy all MAC, e.g., the method's CLS includes the method return value and the role's CLS must dominate the method in order to invoke it and obtain the return value. Second, there is a Signature Constraint Oracle that correctly compares invocation parameter values against the Signature Constraints and returns true or false for every invocation. Third, the same method may have different TC's and SC's for different roles.

The final set of definitions is for authorizations of method(s) to user role(s) and of user role(s) to user(s), to bring together all of the concepts. Recall that the pillars of security as has been discussed earlier are: authorization, authentication, and enforcement. In order to have any enforcement, there must be clearly defined authorizations both for a role to a set of one or more methods, and for a user to a role. We begin by defining a user-role authorization, collect all authorizations for every user role into a matrix, and organize all of the valid user role authorizations into a list. Formally, we define:

Definition 14: A *user-role authorization,* $URA = [UR, M, TC, SC]$, means UR (Definition 7) authorized to invoke M (Definition 4) within time TC (Definition 12 - default $[ct, \infty]$), limited by values SC (Definition 11 - default true). Note that Figure 16 has URAs for JPlanCR1, ArmyLogCR1, and ArmyLogCR2, and illustrate the combinations of all earlier constructs.

Definition 15a: A $r \times q$ *UR authorization matrix, URAM,* where $q = \sum_{i=1..m, j=1..n_i} q_{ij}$, is:

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

Initially URAM contains all 0 entries. An entry with value 1 is a *valid URA, VURA*. At design time, a URA must have a UR's CLS $\geq$ a M's CLS and the overlap of TC and LTs to become a VURA; at runtime, a VURA must be activatable after the current time.

Definition 15b: A *valid user-role authorization list,* $VURAL = \{VURA_i \ \forall \ i = 1..v\}$, $v \leq r \times q$, contains all *valid URAs*, VURAs (URAM(-,-) = 1).

In Figure 16, for the role Joint Planner Crisis 1 (JPlanCR1), there is the tuple: [JPlanCR1, [01dec02, 01jun03], S]. The tuples [JPlanCR1, CrisisPicture, [ct, infinity], true] and [JPlanCR1, ArmyBattleCmdSys, [10dec02, 16feb03], true] define two user-role authorizations (URA) for JPlanCR1. Note that each URA is constructed with different methods', TC's and SC's, to fully represent the desired privileges of the role. This also allows each URA to have different constraints based on the method being used. In the same vein, the same method can be used with different UR's, with the same or different constraint applied. For example,

in Figure 16, URA's [JPlanCR1, CrisisPicture, [ct, infinity], true] and [ArmyLogCR1, CrisisPicture, [10dec02, 16feb03], Grid1 > NA20 AND Grid2 < NC40] both refer to the same method, CrisisPicture, with different SC's and TC's. The TC for JPlanCR1 is [ct, infinity] and the TC for ArmyLogCR1 is "[10dec02, 16feb03]". JPlanCR1 basically has no TC and ArmyLogCR1 is constrained to "[10dec02, 16feb03]". The SC for ArmyLogCR1 is "Grid1 > NA20 AND Grid2 < NC40" and the SC for JPlanCR1 is "true." The Boolean "true" means there is no Grid constraint so that the entire crisis picture is available to JPlanCR1. On the other hand, the ArmyLogCR1 crisis picture is limited to grid coordinates from NA20 to NC40.

The user-role authorization matrix (URAM), Figure 18, is the repository for which methods are authorized for each user-role. A "1" means the method is authorized and a "0" indicates no authorization. The authorization also means that at design time, when the roles were created, all TC's, SC's, and MACC's were met. During runtime, this matrix is validated again to insure there is no constraint violations. These checks are key to our enforcement framework. By examining Figure 18, one can determine the valid user-role authorizations by simply choosing the matrix entries under each UR; this list is known as the valid user-role authorization list (VURAL), as given in Definition 15b.

User Authorization Matrix (UAM)
1 = authorized, 0 = not

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 18: UR Authorization (URAM) and User Authorization (UAM) Matrices.

Users, of course, need to be assigned a role in order to execute any function or privilege. A user authorization is simply assigning a user to a role for a given time. In order for a user to be authorized to the role, the user's CLR must

dominate the role's CLS, and the role must be available during the lifetime of the user. Formally, to mirror Definitions 14, 15a, and 15b, we define:

Definition 16: A *user authorization,* $UA = [U, UR, TC]$, means U (Definition 9) is authorized to play UR (Definition 7) for time TC (Definition 12 - default $[ct, \infty]$) for when the role is available to U.

Definition 17a: An $r \times u$ *user authorization matrix, UAM,* is:

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

Initially UAM contains all 0 entries. An entry with value 1 is a *valid UA, VUA*. At design time, a UA must have a user's CLR $\geq$ a role's CLS and the overlap of TC and LTs to become a VUA; at runtime, a VUA must be activatable after the current time.

Definition 17b: A *valid user authorization list*, $VUAL = \{VUA_i | i = 1..w\}$, $w \leq r \times u$, contains all *valid UAs, VUAs* (UAM(-,-) = 1).

Definition 18: A *client, C,* is an authorized user U, identified by *client token* C = [U, UR, IP-Address, Client-Creation-Time].

Like the UR has a URAM, there is a user authorization matrix (UAM) to use for security assurance enforcement for user authorizations. The UAM, Figure 18, is annotated with a "1" only if all TC, LT, and MACC requirements are met. These requirements are checked during user to user-role assignment (design time) and before a user can execute a role (runtime). We check by checking the UAM matrix at runtime, for assurance reasons. In a dynamic environment when user and user-role attributes can change, it is prudent to check authorizations during runtime. Of course, once the UAM is populated, Figure 18, one can generate a valid user authorization list (VUAL), Definition 17b.

Figure 19 shows the relationship between a user (U in Definition 9) and a client (C in Definition 18). The client (C) is used to track user activity and allows for a user (U) play more than one role during a session. The client or client token is always unique because it is based on a tuple that includes a Client-Creation-Time, which is the current time at client initiation. It is not possible for one user to initiate two clients from the same IP-Address at the same time, assuring uniqueness. Since every client is unique, it is useful for logging activity during a given session. Since a unique token is required throughout the enforcement framework, the uniqueness and randomness of the token adds extra security assurance.

From a security assurance perspective, a valid URA (Definitions 14, 15a and 15b) can only be created if all required checks of a UR's capabilities against a M's characteristics are satisfied (i.e., TC and MACC checks), and the VURA

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

Users:

General DoBest:	[DoBest, [ct, infinity], T]
Colonel DoGood:	[DoGood, [01dec02, 01jun03], T]
Major DoRight:	[DoRight, [01dec02, 01jan03], S]
Major CanDoRight:	[CanDoRight, [01jan02, 01feb03], T]

C = [U, UR, IP-Address, Client-Creation-Time]

Client Token:

[DoRight, ArmyLogCR1, 100.150.200.250, 16nov02-14:50:04]
[U, UR, IP-Address, Client-Creation-Time]

Figure 19: Example Users and Client Token.

then sets the criteria (UR, M, TC, SC) under which the invocation can occur. A corresponding set of guarantees also holds for a valid UA (Definitions 16, 17a, and 17b) to set the criteria (U, UR, TC) for which a U is authorized a UR.

3.3 Security Model Assumptions

The research objective of this dissertation is to address issues related to security policy definition, authorization, authentication, enforcement, and security assurance. For completeness, we collect all of the critical assumptions into four separate tables.

Table 1: Assumptions I - X.

Assumption #	**Description**
I	RBAC is considered an appropriate approach to flexible security policy realization.
II	MAC is necessary for all government or government interfacing information systems, so therefore needs to be considered with every distributed application. This means that every invocation is checked with respect to MAC constraints to increase assurance that a classified method is not accidentally invoked
III	Assume four Sensitivity Levels (SLEVEL), any number will work with any hierarchy in our approach to make it viable for non-government use.
IV	A distributed application is constructed from publicly available software artifacts and federated groups of users that have access to public artifacts.
V	Compiler, Operating Systems, and Network protocols are stable and not prone to intrusions that would allow unauthorized access to application methods in contradiction to our model.
VI	Within federated groups, users have roles and roles have clearly identified permissions. Permissions are realized through Application Programmer Interfaces (APIs) or public methods.
VII	We call the software artifact a resource, which represents a legacy, GOTS, COTS, database, etc., accessible through its Application Programmer Interfaces (APIs).
VII	Software artifacts can register and publish their APIs, so that the APIs are available to clients and users.
IX	A resource is comprised of services, where a service is a logical grouping of methods. All services belong to a resource and all methods belong to a service. There are no registered rouge methods that can be accessed bypassing the enforcement framework.
X	All services and methods can be registered with middleware lookup services and provide concrete interfaces with components.

Table 2: Assumptions XI - XX.

Assumption #	**Description**
XI	Once registered with the Lookup/Security Service, there is no way to access Resources, Services or Methods other than using the Lookup/Security Service.
XII	Methods perform specific functions and have a signature consisting of method name, return type, and parameters with types.
XIII	The method classification dominates the method return value. Method return values are at the same or lesser classification then the method from which they were generated.
XIV	The delay time between requesting a current time and the global clock is not significant. This means the global clock time is the current time (ct).
XV	The delay time between a clearance or classification change and the effect on the enforcement framework is not significant. This means it is acceptable to enforce the policy or authorization change at the next invocation.
XVI	The application resource dictates security policy. The security model and enforcement framework realizes that policy.
XVII	A User can play only one role at a time during a session, but can change roles in that secession. During secession login, authorization and authentication, the lookup service issues a proxy that is valid for that secession for the authorized roles.
XVIII	The lookup service middleware has adequate protection against intrusion that would cause failure or subversion of the security model and enforcement framework. Middleware services provide the bridge between distributed software artifacts, so artifacts can publish and register APIs, making APIs available to other clients, users, and resources.
XIX	Security Model and Enforcement Framework have sufficient: backup, concurrently updating, dual homed and redundant database support to provide consistency and survivability to application security enforcement.
XX	Privacy by means of encryption or channel tunneling is an implementation detail that does not effect access control.

Table 3: Assumptions XXI - XXX.

Assumption #	**Description**
XXI	Locations of resources, services, and methods are transparent to the application user. Resource location is part of the registration process. It does not matter where artifacts reside.
XXII	No assignment can take place before current time (Role to User, Method to Role). This means, for example, that a method with a lifetime that ends before the current time (Method.LT.et is ¡ current time), cannot be assigned. However, if the Method.LT.et is ¿ current time, the method can be assigned.
XXIII	During assignments (X to Y), if the only overlap (intersection) of lifetimes occurs when X.LT.et = Y.LT.st, there will be no assignment.
XXIV	Must assume there is internal role consistency with lifetimes, according to resource policy.
XXV	When assigning a role to a user, there will be a Design Time Assurance Check (lifetime)User.LT vs. Role.LT. It is assumed that the role is internally consistent. Individual Method assurance checks for lifetime will be executed at Run Time.
XXVI	Method Name, LT, CLS, and Parameters are set for each method during resource registration with security middlewear.
XXVII	Overall classification of resource is calculated using minimum classification of methods.
XXVIII	The start time of a service is the start time of the earliest method start time.
XXIX	The Name, LT, and CLS of each role is set by the security officer when designing security policy for the DAPPL.
XXX	User identifiers cannot be duplicated with different LTs or CLRs.

Table 4: Assumptions XXXI - XXXV.

Assumption #	**Description**
XXXI	Signature Constraint does not yet include return values.
XXXII	Same Method may have different TC's and SC's for different roles, but not within the same role.
XXXIII	Authorization Matrices are initialized with all zero (0) values to reflect no authorizations.
XXXIV	There is a Signature Constraint Oracle that is a constraint checker that takes in the parameter values of the methods invocation and compares these values against the Signature Constraints, SC, and correctly returns true or false.
XXXV	Assume that all method interactions among resources satisfy all required MAC (satisfies the *Simple Security Property* [13] for invocations of read-only and read-write methods, and the *Simple Integrity Property* for invocations of read-write methods) and also a degree of *safety* (nothing bad will happen when a user playing a role invokes a method) and *liveness* (all good things can happen when a user playing a role invokes a method).

3.4 Related Work

There have been many efforts on RBAC [15, 24, 39, 84, 111, 129], including he RBAC 96 model [2, 85, 90, 111]; the NIST Model [10, 40, 45, 52]; and the University of Connecticut model [24, 25, 28, 49, 129]. A temporal approach defined in [17] is relevant due to its changing environment and the shifting of responsibilities. Likewise in [16, 41, 53], the importance of using constraints for identity and authorizations leads to improved granularity on access controls. The major difference between our approach and these models is our use of a method signature constraint to enforce a more fine-grained access policy. The NIST and RBAC approaches seek to protect data objects [40, 110, 111, 128]. These approaches almost always require a change to the object structure itself, which is not very flexible. They view protection of the object as the means to protect the computer system [110]. Our approach is directed towards the method, which is used to manipulate objects [28, 96, 97]. In this way, we can not only develop mutual exclusion constraints, but also achieve a fine-grained access control, which can use time, signature, mandatory access control, and lifetime constraints without changing the method code or object structure. Also, we feel using a role hierarchy, as depicted in these models is problematic. The NIST and RBAC96 models use role hierarchy to simplify creation and administration of roles. In a large organization, this may be true, but the use of these hierarchies actually causes problems with respect to role deconfliction and mutual exclusion [54]. Our model does not dictate a specific role hierarchy or inheritance policy, but both can be modeled with proper role and user administration. The authorization responsibilities are constrained by policy.

For MAC, we have reviewed many of the classic approaches [13, 32, 33, 38]. Our interest in this section is on work that has attempted, like our own, to unify RBAC and MAC policies. Strict MAC control and RBAC flexibility are very different concepts, but can be brought together and prove useful [91, 109]. Data association and aggregation is a problem with any access control mechanism, particularly MAC. There has not been a lot of interest in this area since MAC is considered to be too rigid for most applications, but [2, 23, 85, 90, 91, 109, 111, 112, 119, 128] have provided a basis for our research. Most efforts with respect to MAC have been concentrated on the object [111, 128]. The majority of the research has used an object-level approach, but our specific focus is at the method level, to customize these "published" APIs on a role-by-role basis. This customization will constrain the invocation of a method by a user playing a role by: Mandatory access control (user's clearance dominates method's classification), time limits (user's lifetime subsumes method's time limit for the role), and data values (user's invocation of the method with actual parameter values is within value constraint of the method for that role). Our approach greatly improves the granularity of access control from the on/off method level of our prior research [25, 49]. By controlling access of users playing roles to

methods based on parameter values, time limits, and classifications, we have effectively established value-based security without having to explicitly control access to individual objects.

In another effort, the Air Force Research Laboratory in conjunction with Verdian, is working on a comprehensive information tagging and release policy called Secure Information Releasability Environment [119]. There are products like e-Portal and Multi-domain Dissemination System, which concentrates on sensitive information access using secure transmission [119]. In addition, there are systems that use firewall technology to create secure network connections between hosts on any unclassified network [60]. All of this work is relevant for different aspects of MAC, but none address the critical issues of federation of resources/users and the availability and access of resources/information in a secure fashion based on classification and clearances. Our approach considers MAC at design and runtime to ensure the user/client possess the prerequisite clearance before method invocation. This is in contrast to object security controls that require inspection of the object before access is considered. By using a method approach, an object is not accessed unless the user is authorized and has the clearance to access the method.

Finally, in the constraints area, both the NIST and RBAC96 models [111, 112] support them, but not in the same way as our approach. In RBAC96 and NIST, constraints are discussed for both permissions and users with mutual exclusion as the major consideration. User Constraints prevent users from holding conflicting roles and permission constraints prevent certain permissions from being applied to the same role. These constraints are important and will be incorporated into our model. However, these constraints are not fine-grained. We feel constraints should be more than just "on or off". Other models discuss the use of temporal constraints to control access [3, 14, 17]. These constraints are used for access control, but in a workflow context, where one process is required before another can be executed. However, assigning time restrictions is a valuable concept, which we will modify for our role-based approach. There should be ranges of acceptable values and times that can dictate access. Our model will take a unique approach to these constraints by allowing a time window (time constraint) when the method can be used. This time window can be set differently depending on the role to which the method is assigned, effectively allowing the same method to have different time constraints depending on the role. In addition, our model allows for constraints to be attached to the method parameters (method signature). Again, the parameters can be levied differently depending on the role the method belongs to. The use of LTs is another form of time constraint, which is different from the time constraint leveraged on a method. LTs are assigned to users, roles, resources, services, and methods to bound access to only users that meet all of the lifetime restrictions. The combination of these constraints is innovative.

Chapter 4

Delegation in the RBAC/DAC/MAC Model

The security capabilities for a security model must be able to both represent the privileges 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 be allowed to *delegate* all or part of his/her authority to another individual, increasing security risk, and raising interesting security assurance implications [11, 68, 79, 138]. Large organizations often require delegation to meet demands on individuals in specific roles for certain periods of time. The main objective of this chapter is to examine role delegation, extending the RBAC/MAC features of the unified security model as given in Chapter 3 to include DAC. Specifically, we propose a means to allow individuals to delegate roles within security policy guidelines (during the definition of a policy), while simultaneously maintaining security assurance at runtime.

The major focus of this chapter is to extend the RBAC/DAC/MAC security model and enforcement framework from Chapter 3 with role delegation, allowing a security officer to assign delegation authority at design time, which can then be enforced, at runtime. 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 (this chapter), and to incorporate delegation and its enforcement into the run-time environment (to be covered in Chapter 6). The delegation modeling extensions presented in this chapter are based on work conducted by M. Liebrand, a M.S. graduate at Renssalear at Hartford, as part of his seminar paper for his degree. M. Liebrand worked with C. Phillips and S. Demurjian, and focused on conceptual extensions to the security model as presented in Chapter 3. C. Phillips formalized those extensions for role delegation, as they appear hear in, and as published in [67].

The remainder of this chapter investigates and analyzes the extensions to the unified RBAC/DAC/MAC security model [28, 96, 97], presented in Chapter 3, to support all aspects of role delegation. In Section 4.1, the focus is on security model extensions to support delegation. In Section 4.2, the emphasis is on

the enforcement framework modifications, exploring the inclusion of role delegation and revocation rules [138]. Next, in Section 4.3, 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 [11]. Finally, Section 4.4 compares our work to other related research efforts.

4.1 Delegation Model Extensions

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 [68]; 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 [79]. 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 [114].

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 into the unified RBAC/DAC/MAC model whose RBAC and MAC capabilities have been detailed in Chapter 3.

To begin, we focus on the concept of *delegatable* which indicates whether a role can be delegated from one user to another. Users cannot dictate whether a role is delegatable, this is an administrative function determined during design time. However, once declared delegatable, a user does has some discretion in terms of when to initiate the delegation. Formally, we define:

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:

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

This is why it is key that each role holds a vector value for delegation (DURV) if delegation is going to be used at all. Initially, DURV contains all 0 entries. This insures that all delegation action is initiated by the security administrator during design time. For the URs given in Figure 16, the roles CDR_CR1, JPlanCR1, JPlanCR2, are delegatable (respective DURV(-) = 1) and ArmyLogCR1, and ArmyLogCR2 are not (respective DURV(-) = 0).

The next two definitions involve a differentiation between users of roles with respect to delegation. From an enforcement perspective, it will be critical to know if the role that a user is playing is as the result of an *original* authorization to that user, or the result of a delegation. Formally, we define:

Definition 21: An *original user, OU* $\in UL$, of a UR is authorized to the UR via the security policy ($\exists$ a VUA for the OU/UR, i.e., UAM(UR,OU) = 1), and not by a delegation.

Definition 22: A *delegated user*, $DU \in UL$, is a user U that can be delegated a UR by an OU/DU (there is not a VUA for the DU/UR, i.e., $UAM(UR, DU) \neq 1$)), where a DU cannot be an OU for that same UR.

It is important we distinguish between a delegated and original user. If a user is an original user of a certain role, then that role can never be delegated to that user. And why should it? If you are an original user of a role, there is no need to be a delegated user for that role.

The OU and DU differentiation is utilized to track, for each user role, whether a user is an OU, a DU, or not authorized. Thus, in a similar fashion to the authorization matrices in Chapter 3, we introduce an authorization matrix for delegation. Formally, we define:

Definition 23: The $r \times u$ *user delegation/authorization matrix, UDAM,* indexed by roles and users, is:

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

Initially UDAM contains all 0 entries. As users are authorized to roles via VUAs (Definition 17a), the relevant entries are set to 1.

From a assurance perspective, there must be guarantees on the capabilities of an OU with respect to the characteristics of a DU, specifically, to insure that LTs/TCs and CLS/CLR are consistent, which will enable the delegation to be defined and checked at runtime.

The OU and DU identification is utilized to maintain some control over delegations, which is referred to as delegation authority (DA), which is the authority to grant and revoke delegation. Definition 23. For example, the OU should always be able to revoke delegation, but the DU cannot, unless the DU is given the authority to delegate, called pass-on delegation authority (PODA), Definition 24. Even then, the DU can only revoke his/her action.

The final three definitions establish, for a given user of a role, the ability to have delegation authority, DA, and/or pass-on delegation authority, PODA (able to pass on the authority to delegate). DA is the authority to grant and revoke delegation, as given by a security officer at design time, and as initiated by a user at runtime. PODA is an authority that allows a delegated user to pass on DA to another user. Clearly, there is the potential for this process to be infinite, with authority passed on and on down a chain. However, in our model, we have chosen to limit delegation as follows: 1. A user is not given delegation authority (DA), so that the role stays at a given level. 2. A user can be given delegation authority, but not pass-on delegation authority, which means a role can be passed on to a second user (not currently an OU or DU) and no further. And 3. A user can be given both delegation authority and pass-on delegation authority, which means a role can be passed on from a second party to a third party (not currently an OU or DU). Only an OU can grant pass-on delegation authority, so the delegation chain can go no further than the third party. Formally, we define:

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

Definition 25: *Pass-on delegation authority, PODA,* is authority given to an OU/DU to pass on DA for a DUR to another OU/DU.

Definition 26: The $r \times u$ *delegation authority matrix, DAM,* indexed by roles and users, is:

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

Initially DAM contains all 0 entries.

From an assurance perspective, there must also be guarantees about DA and PODA, since having DA allows an OU/DU to delegate, and more critically, having PODA, allows an OU/DU to pass on that delegation authority to another DU, which is a potentially dangerous privilege. To illustrate delegation concepts, Figure 20 provides populated delegation matrices. To drive home the concepts, we offer two examples below on delegation.

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 20: Example: UAM, URAM, UDAM, URAM.

Example 1: To illustrate delegation, suppose that DoBest wishes to delegate his (CDR_CR1) to DoGood with DA, with DoBest, CDR_CR1, and DoGood as given in Figure 20. This delegation can occur since DoBest is an OU (UDAM(CDR_CR1, DoBest) = 1) of CDR_CR1, DoGood is not an OU nor DU (UDAM(CDR_CR1, DoGood) = 0), the UR is delegatable (assume DURV(CDR_CR1) = 1), and DoGood dominates CLR ((CDR_CR1 CLS = T) $\leq$ (DoGoodCLR= T)).
DoBest can also grant DA because he has PODA (DAM(CDR_CR1, DoBest) = 2). Note that DoGood can execute the UR CDR_CR1, but is limited to his

own LT. Also note that UAM(CDR_CR1, DoGood)=1, UDAM (CDR_CR1, DoGood)=1 (DU), DAM(CDR_CR1, DoGood)=1, (has DA) and VUA = [DoGood, CDR_CR1, [ct, ∞]] is created, as given in Figure 20.

Example 2: Continuing from Example 1, suppose DoGood wishes to also delegate UR: CDR_CR1 to 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), CanDoRight does have the prerequisite clearance ((CDR_CR1CLS = T) (CanDoRightCLR= T)) and the DA (DAM(CDR_CR1, DoGood) = 1) from Example 1. However, this delegation will be limited to the LT of CanDoRight (CanDoRightLT = [01jan03, 01feb03]). Note, that the UAM, UAM(CDR_CR1, CanDoRight)= 1 ("authorized"), the UDAM, UDAM (CDR_CR1, DoGood) = 1 (delegated user, DU), the DAM, DAM(CDR_CR1, CanDoRight), remains "0" (has no delegation authority) and a VUA, VUA = [CanDoRight, CDR_CR1,[ct, ∞]] is created (TC by default is [ct, ∞], but can further constrain the delegation).

4.2 Delegation Model - Delegation and Revocation Rules

In addition to the RBAC/DAC/MAC security model extensions for role delegation in the previous section, there are also infrastructure-related extensions that are needed in support of the enforcement framework. In the definitions given earlier, 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, we employ a useful set of definitions and rules for delegation which underlie a proposed delegation language [138]. Recall the conceps from the previous section, namely: original user (OU), delegated user (DU), delegation authority (DA), pass-on delegation authority (PODA), and delegatable user role (DUR), and define the following:

- 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.).

- 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 [138]. 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.

Given these definitions and assumptions, 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 [138], 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, and at design time, incorporated into the security administrative tools, which will be discussed in Chapter 6.

4.3 Delegation Model - Analysis of Delegation Capabilities

This section analyses the role-delegation extensions of the RBAC/DAC/MAC security model and enforcement framework 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 [11] 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 [11] 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 [11] 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. [138] extended [11] 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 [11]. 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 modelled 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.

4.4 Delegation Related Work

May different research efforts on role delegation are discussed in detail in [11, 68, 79, 114, 138]. Related research for role delegation has been discussed when relevant throughout the chapter. In large part, our work is a continuation of work by [138], delegation definitions 19 to 26 have been influenced this work. However, we have extend their efforts significantly by:

- Incorporating delegation into a MAC environment.
- Applying security assurance assertions to delegation.
- Applying delegation and delegation authority constraints. Using these constraints an original user of the delegatable role will maintain control of delegation authority. This will provide limited delegation capability within a security policy.
- Simplified the delegation and revocation rules for our enforcement framework with two main differences: 1. Simplification of the rules with the PODA and DA matrices used to manage delegation depth rather than an integer; and, 2. Prohibition of independent (by another OU) revocation, i.e., revocation only by the security officer.

In combination with our RBAC/MAC capabilities in the security model, role delegation becomes a true user controlled option, but security administrators retain overall control if necessary and more importantly, security policy will always be enforced. It is this combination that provides security assurance and distinguishes our role delegation approach from others.

Chapter 5

Security Assurance

The importance of security assurance in large-scale applications development continues to grow, as evident by numerous assurance research efforts [4, 45, 71, 72]. In one approach [4], assurance is supported by a sequential model (via OSI model) and a network-centric model, for sharing that ranges from the user to the transmission or communication method used. Another effort improves the INFOSEC Model [72] by adding security services and countermeasures. A third effort [71] proposes key security services for assurance, including, availability, integrity, authentication, confidentiality, non-repudiation, etc. In a fourth effort, the 1998 NIST Model [45] has improved administration tools (higher assurance) and database consistency checks. In the area of mandatory access control (MAC), assurance-related properties include: the *Simple Security Property*, a subject can read information at the same or lower clearance level, i.e., read-down/no-read-up [13]; the *Strict *-Property*, a subject can only write information at exactly the level for which it is cleared, i.e., write-equal [91]; the *Liberal *-Property*, a subject with a low clearance level can write an object with the same or higher clearance level, i.e., write-up/no-write-down [13, 91]; and the *Simple Integrity Property*, a subject can write information at its own level and lower, i.e., write-down/no-write-up [18]. In addition, assurance also must focus on application behavior with respect to its defined and realized security policy, specifically, with *safety* (nothing bad will happen to a subject or object during execution) [62, 63] and *liveness* (all good things can happen to a subject or object during execution) [5].

Our main objective in this chapter is to explore assurance for our RBAC/DAC/MAC security model as presented in Chapters 3 and 4. Our approach focuses on which methods of APIs can be invoked based on the responsibilities and security classification level of a role, the security clearance level of the user, and the values (parameters), time, and classification level of the method; all of which must be *satisfied* in order for the invocation to successfully proceed. In order to achieve this capability, we provide assurance that validates the method invocation by a user playing a role in a number of different ways.

First, we must guarantee the timeframe of the invocation, to insure that a user (with one lifetime) playing a role (with another lifetime) can invoke a method (yet another lifetime) at the current time. In fact, this assurance guarantee is partially checked when the security policy is being defined, and must be rechecked at execution time (i.e., actual method invocation). Second, we must guarantee that the method invocation does not violate MAC domination. Specifically, we demonstrate that our approach satisfies the *Simple Security Property* [13] for invocations of read-only and read-write methods, and the *Simple Integrity Property* for invocations of read-write methods. Finally, for a security policy defined using our RBAC/DAC/MAC model, we provide a series of theorems that insure a degree of *safety* (nothing bad will happen when a user playing a role invokes a method) and *liveness* (all good things can happen when a user playing a role invokes a method). When combining MAC and RBAC, MAC requirements take priority and the enforcement mechanisms must support this requirement.

Security assurance is critical in our RBAC/DAC/MAC model to allow the consistency of URs, CLR/CLS levels, LTs, role delegations, user authorizations, etc., to be verified when the security policy is defined, changed, and executed. Towards that end, in the remainder of this chapter: Section 5.1 examines assurance guarantees on the *available time* of access for a method invocation; Section 5.2 details the attainment of MAC guarantees with regards to assurance, namely, the *Simple Security Property* [13] for invocations of read-only and read-write methods and the *Simple Integrity Property* for invocations of read-write methods; Section 5.3 explores the degree of *safety* (nothing bad will happen) and *liveness* (all good can happen during an invocation). In all three areas, lemmas and/or theorems are provided to demonstrate the attaintment of assurance guarantees, with RBAC/DAC/MAC model extensions as needed. Overall, our research in this regard has been influenced by numerous others [13, 17, 18, 62, 63, 73, 91, 109, 116].

5.1 Time-Based Guarantees

To support time-based guarantees, we introduce *available time*, AT, which represents the maximum amount of time derived from various intersections of LTs and TCs of the RBAC/DAC/MAC modeling constructs in the Definitions in Chapters 3 and 4. To explain AT, recall that each LT and TC (which is also a LT) is a discrete time interval. Each of the modeling constructs in Chapters 3 and 4 (e.g., URA, UA, OU, DU, etc.) are all combinations of other constructs (e.g., $URA = [UR, M, TC, SC]$) which involve LTs (e.g., UR and M each have LTs) and a TC. When defining URA, there must be LT overlap, AT. Formally, let $\Sigma^{LT} = \cap_{alli}\Sigma_i^{LT}$, where each Σ_i^{LT} represents a LT or a TC, using intersection for LTs/TCs as given in Definition 1. Then, any time t is in the maximum amount of time, AT, or $t \in \cap\Sigma_i^{LT} \forall\ i \ \Leftrightarrow\ t \in \Sigma^{LT}$. Finally, to assist us in proving lemmas, we offer the following definition to compare two LTs:

Definition 1a: Let $Compare(X, Y)$ be a function that returns the overlap of LTs X and Y. $Compare(X, Y) = \{Y$ if $X \triangleright Y$; X if $Y \triangleright X$; $\emptyset$ if $Y \cap X = \emptyset$; $Y \cap X$ otherwise$\}$.

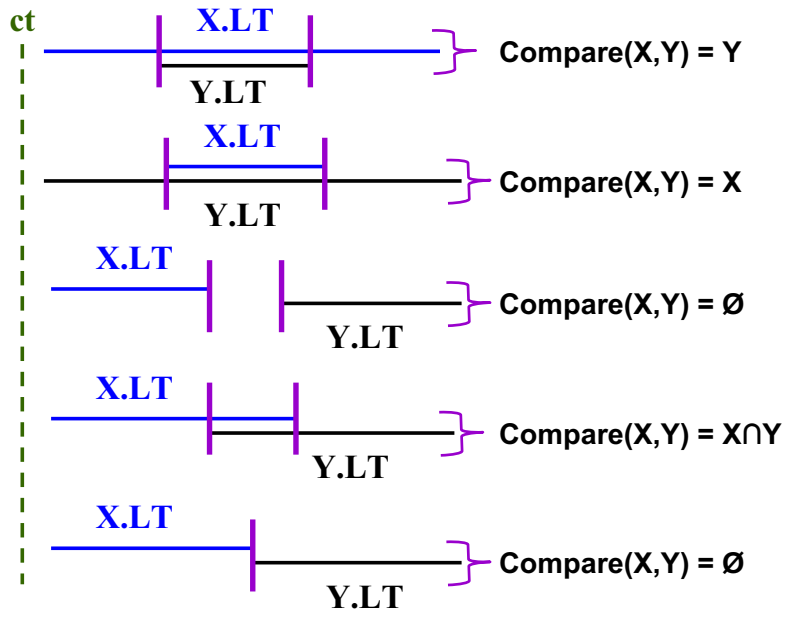


Figure 21: The Compare Function

The Compare Function is very important to our security model and security assurance, as the compare function is used to determine the available time (at), which is the maximum amount of time functions are available to a User. Limiting a function to the maximum available time is good assurance practice. Figure 21 shows the possible outputs of the Compare Function. Using Definition 1a, we offer lemmas for the available time, AT, for URAs (Definition 14), URs (Definition 7), UAs (Definition 16), DUs (Definition 22) and Users (Definition 9). ATs are useful in demonstrating the degree of safety and liveness that can be supported (see Section 5.3).

Lemma 1 is the first of several Lemmas we will build upon to prove security assurance in our MAC/DAC/RBAC Security Model. Lemma 1 , illustrated in Figure 22, states that the time a user-role authorization is available is the intersection of lifetimes.

Lemma 1: If $URA_i = [UR, M, TC, SC]$ is a VURA, then $URA_i^{AT} = M^{LT} \cap UR^{LT} \cap TC$.
Proof:

1. By Definitions 14 and 15a, $URA_i = [UR, M, TC, SC]$ as a VURA means $URAM(UR, M) = 1$. By Definition 7, $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$. Set $URA_i^{AT} = UR^{LT}$.

2. By Definition 4, $M = [M^{Name}, M^{LT}, M^{CLS}, M^{Params}]$. Apply Definition 1a with $URA_i^{AT} = Compare(URA_i^{AT}, M^{LT})$.
3. By Definition 12, a TC is a LT. In $URA_i = [UR, M, TC, SC]$, TC constrains when UR authorized to M. Apply Definition 1a with $URA_i^{AT} = Compare(URA_i^{AT}, TC)$.
4. If $URA_i^{AT} \cap [ct, \infty] = \emptyset$, then $URA_i^{AT} = \emptyset$ and proof completes.
5. Otherwise, $URA_i^{AT} = M^{LT} \cap UR^{LT} \cap TC$ since it is equivalent to $URA_i^{AT} = URA_i^{AT} \cap TC$ (by steps 2 and 3 of the proof and substituting for URA_i^{AT} on the r.h.s. with $M^{LT} \cap UR^{LT}$). Note that if $URA_i^{AT} = \emptyset$, then there is no overlap.

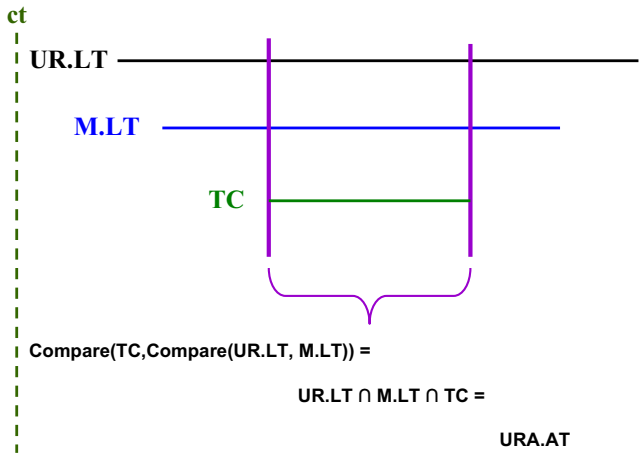


Figure 22: Lemma 1, User-Role Authorization, Available Time

Lemma 2 ,illustrated in Figure 23, states that the time a user-role is available ,UR.AT, is bound by the minimum start time, min_st, of all of the User-Role Authorizations (URA) for that UR and the maximum end time, max_et, of all of the User-Role Authorizations (URA) for that UR. Recall from Definition 1, that lifetimes are bound by a start time, st, and an end time, et and from Definition 4, that all Methods have a lifetime, M.LT, and it is the M.LT with the TC that determines the URA.LT.

Lemma 2: If $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$ is a user role, then $UR^{AT} = UR^{LT} \cap URA\_Range \cap [ct, \infty]$ where $min\_st = min(allURA^{AT.st})$, $max\_et = max(allURA^{AT.et})$, and $URA\_Range = [min\_st, max\_et]$.
Proof:

1. By Definitions 14 and 15a, $URA_i = [UR, M, TC, SC]$ as a VURA means $URAM(UR, M) = 1$. By Definition 7, $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$. Set $URA_i^{AT} = UR^{LT}$.

2. By Definition 1, A *lifetime, LT,* is a time interval with start time (st) and end time (et), [st, et], et > st.
3. If $URA_i^{AT} \cap [ct, \infty] = \emptyset$, then $URA_i^{AT} = \emptyset$ and proof completes. Else, there must be a URA^{LT} with $URA^{LT.st}$ and $URA^{LT.et}$
4. If there is more than one URA for a UR, Then the UR is available at the earliest start time, $min\_st = min(allURA^{AT.st})$.
5. If there is more than one URA for a UR, Then the UR is no longer available at the latest end time, $max\_et = max(allURA^{AT.et})$.
6. If we repeat the previous step with each additional URA, we build an $URA\_Range = [min\_st, max\_et]$.
7. If we intersect URA_Range with UR^{LT} we have the earliest possible start time of a UR and this is precisely Lemma 2.

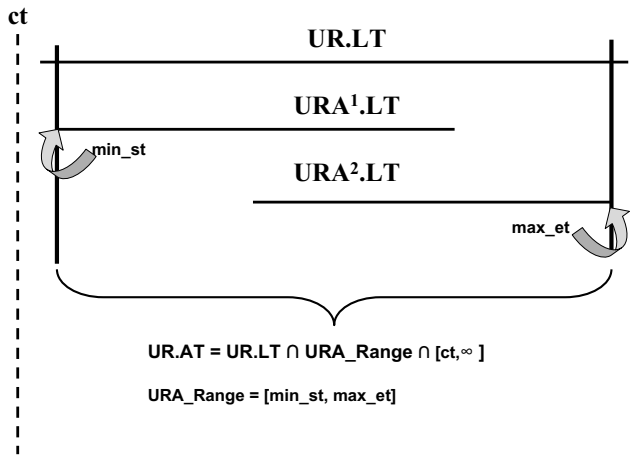


Figure 23: Lemma 2, User-Role, Available Time

Lemma 3 states that the authorized user's available time for a specific user-role is limited by the intersection of the lifetimes of the role, the user, and the assigned time constraint. This Lemma is similar to Lemma 1 in its construct.

Lemma 3: If $UA = [U, UR, TC]$ is a VUA, then $UA^{AT} = U^{LT} \cap UR^{AT} \cap TC$.
Proof:

1. By Definitions 16 and 17a, $UA = [U, UR, TC]$ as a VUA means $UAM(UR, U) = 1$. By Definition 9, $U = [U^{UserId}, U^{LT}, U^{CLR}]$. Set $UA^{AT} = U^{LT}$.
2. By Lemma 2, UR^{AT} is the AT of UR for all of its URAs. Apply Definition 1a with $UA^{AT} = Compare(UA^{AT}, UR^{AT})$. Note that UA^{AT} represents the time that U can engage an authorized UR.

3. By Definition 12, a TC is a LT. In $UA = [U, UR, TC]$, TC constrains when UR authorized to U. Apply Definition 1a with UA^{AT} =$Compare(UA^{AT}$,$TC)$.
4. If $UA^{AT} \cap [ct, \infty] = \emptyset$, then $UA^{AT} = \emptyset$ and proof completes.
5. Otherwise, $UA^{AT} = U^{LT} \cap UR^{AT} \cap TC$ since it is equivalent to $UA^{AT} = UA^{AT} \cap TC$ (by steps 2 and 3 of the proof and substituting for UA^{AT} on the r.h.s. with $U^{LT} \cap UR^{AT}$). Note that UR^{AT} may still equal $\emptyset$, i.e., there is no overlap.

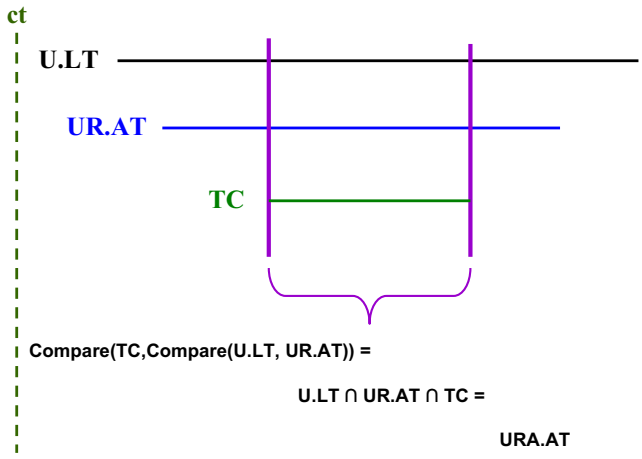


Figure 24: Lemma 3 , User Authorization, Available Time

Lemma 4 sets the lifetime terms for user-role delegation. Lemma 4 sets the maximum available time of a delegation to the intersection of the available time of the original user and the lifetime of the delegated user. This is important because a user should not be able to increase the availability of one of their roles by delegating the role to another user. Also, a delegated user, should not have more availability to a given role than the original user. This Lemma also requires that the proper matrices are considered to validate the delegation or there is no available time.

Lemma 4: If OU delegates one of its DUR, UR, to DU, then a delegated user authorization $DUA = [DU, UR, TC]$ has $DUA^{AT} = OUA^{AT} \cap DU^{LT}$.
Proof:

1. Let $OUA = [U, UR, TC]$ be a VUA for OU. By Lemma 3, OUA^{AT} is the time available for an OU to a UR. Set $DUA^{AT} = OUA^{AT}$.
2. For OU to delegate UR to DU, $UAM(UR, OU) = 1$. When OU delegates UR to DU, then $UAM(UR, DU)$ must be set to 1, and a

$DUA = [DU, UR, TC]$ is created. The UR and TC in the DUA are as given in the OUA (Step 1), and thus do not impact DUA^{AT} since they are already in OUA^{AT}.

3. By Definition 9, $DU = [DU^{UserId}, DU^{LT}, DU^{CLR}]$. Apply Definition 1a with $DUA^{AT} =$ $Compare(DUA^{AT}, DU^{LT})$.
4. If $DUA^{AT} \cap [ct, \infty] = \emptyset$, then $DUA^{AT} = \emptyset$ and proof completes.
5. $DUA^{AT} = OUA^{AT} \cap DU^{AT}$ since it is equivalent to $DUA^{AT} = DUA^{AT} \cap DU^{LT}$ (by Step 3 of the proof and substituting for DUA^{AT} on the r.h.s. with OUA^{AT}).

Lemma 5 is similar to Lemma 2 in that Lemma 2 states that the time a user-role is available, is the intersections of all same user-role authorizations. If a user-role is not part of any user-role authorizations, then the user-role is never available. Conversely, the role will be available if there is a valid user-role authorization. Lemma 5 states that the time a user is available, is the intersections of all same user authorizations. If a user is not part of any user authorizations, then the user-role is never available. Conversely, the user will be available if there is a valid user authorization.

Lemma 5: If $U = [U^{UserId}, U^{LT}, U^{CLR}]$ is a user, then $U^{AT} = \cap_{alli} UA_i^{AT}$, where each UA_i^{AT} represents one VUA where $UAM(UR_j, U) = 1$ for some $j = 1..r$.
Proof:

1. Consider UAM as given in Definition 17a. If $UAM(UR_j, U) = 0$ for some j, then U not authorized to UR_j, and there is no need to use this entry to calculate U^{AT}.
2. Consider all j where $UAM(UR_j, U) = 1$ (VUA of UR to U). If $\exists$ at least one j s.t. $UAM(UR_j, U) = 1$ and $UR^{LT} = \emptyset$, then set $U^{AT} = \emptyset$, and proof completes.
3. Let $UAM(UR_j, U) = 1$ and $UAM(UR_k, U) = 1$ for some j, k. Apply Definition 1a with $U^{AT} = Compare(UA_{i_j}^{AT}, UA_{i_k}^{AT})$, where i_j/i_k is the VUA for UR_j/UR_k.
4. Repeat this step for each remaining entry $UAM(U, UR_j) = 1$, apply Definition 1a with $U^{AT} = Compare(U^{AT}, UA_{i_k}^{AT})$.
5. If $U^{AT} \cap [ct, \infty] = \emptyset$, then $U^{AT} = \emptyset$ and proof completes.
6. Otherwise, $U^{AT} = \cap_{alli} UA_i^{AT}$.

In summary, the ATs for each of the RBAC/DAC/MAC modeling constructs (UR, URA, UA, DUA, U) allow us to constrain when each construct is available.

5.2 MAC Guarantees

Mandatory access controls are governed by strict rules for subjects (users) to access stored information or objects. These rules are so strict that few researchers have tried to incorporate them into current access control systems. These rules lack flexibility because, for assurance reasons, there can be no comprise of classified information. We have chosen to incorporate these rules into our security model in order to provide a flexible MAC/DAC/RBAC Security Model that is flexible enough to handle real-world needs, yet maintain security assurance over classified material. Government classified information rules are detailed in the JOS Orange Book Security Requirements [35] and include many different properties. For our work, we concentrate on the *Simple Integrity* and *Simple Security* Properties.

The *Simple Security Property* dictates that a subject can read information at the same or lower clearance level (read-down/no-read-up) [13]. The idea here is that a subject can not have access to information above his/her clearance level; This eliminates the chance of comprise. There is a basic assumption that subjects will only share information at the appropriate levels and that is fine, but access is a problem. Having access to higher level information violates the "need to know" principle which is a foundation of security assurance. The *Simple Integrity Property* dictates that a subject can write information at its own level and lower (write-down/no-write-up) [18]. Again, since one can only create information at their own level or lower, this makes since. The problem comes with information flow. It is a security violation to send information from high to low. In this section, we prove that our RBAC/DAC/MAC model (see Chapters 3 and 4) and associated enforcement framework [67, 96] satisfies both the Simple Security Property and the Simple Integrity Property.

In support of our model, we must recast the Simple Security and Simple Integrity Properties and apply them to our model with methods, roles, and users. To do so, we extend Definition 4 to include read-write and read-only methods, allowing us to distinguish among methods that modify an object (for simple integrity) and ones that only read an object (for simple security). Formally, we define:

Definition 4a: Each method set, $M_{ij} = \{M_{ijk} \mid k = 1..q_{ij}\}$, is partitioned into read-only methods, $M_{ij}^{RO} = \{M_{ijk}^{RO} \ for\ some\ k\}$ and read-write methods, $M_{ij}^{RW} = \{M_{ijk}^{RW} \ for\ some\ k\}$, with $M_{ij}^{RO} \cap M_{ij}^{RW} = \emptyset$ and $M_{ij}^{RO} \cup M_{ij}^{RW} = M_{ij}$.

Next, the function *Dominate* compares two CLR and/or CLS (Definition 2 and Definition 13) to return true when one dominates the other and false otherwise.

Definition 28: Let $Dominate(X, Y)$ be a boolean function, with X and Y either CLR/CLS (Definition 2) with $U < C < S < T$ [35]. $Dominate(X, Y) = \{true\ if\ X \geq Y; false\ if\ X < Y\}$.

Given these definitions, we can now present and prove a series of lemmas on user role authorizations (URAs), user roles (URs), and user authorizations (UAs) in regards to the Simple Security and Integrity Properties.

Lemma 6.1: If $URA_i = [UR, M, TC, SC]$ is a VURA, then URA_i satisfies the Simple Security Property of role (subject) versus method (object) for all M^{RO} and M^{RW}.
Proof by contradiction:

1. By Definitions 14 and 15a, $URA_i = [UR, M, TC, SC]$ as a VURA means $URAM(UR, M) = 1$. By Definition 7, $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$, where UR^{CLS} is the role's CLS. By Definition 4, $M = [M^{Name}, M^{LT}, M^{CLS}, M^{Params}]$, where M^{CLS} is the method's CLS. For MAC purposes, treat UR as the *subject* and M as the *object*.
2. If $M \in M_{ij}^{RO}$ or $M \in M_{ij}^{RW}$, it has a CLS, M^{CLS}. Suppose $Dominate(UR^{CLS}, M^{CLS}) = false$. Then, $URAM(UR, M) = 0$ (Definitions 14, 15a, and 15b).
3. This contradicts Step 1 where $URAM(UR, M) = 1$.
4. Therefore, $Dominate(UR^{CLS}, M^{CLS}) = true$ ($UR^{CLS} \geq M^{CLS}$).
5. This is exactly the Simple Security Property (read-down/no-read-up) in [13].

Lemma 6.2: If $URA_i = [UR, M, TC, SC]$ is a VURA, then URA_i satisfies the Simple Integrity Property of role (subject) versus method (object) for all M^{RW}.
Proof by contradiction: Exact Steps as Lemma 6.1 except for Steps 2 and 5 (below).

1. By Definitions 14 and 15a, $URA_i = [UR, M, TC, SC]$ as a VURA means $URAM(UR, M) = 1$. By Definition 7, $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$, where UR^{CLS} is the role's CLS. By Definition 4, $M = [M^{Name}, M^{LT}, M^{CLS}, M^{Params}]$, where M^{CLS} is the method's CLS. For MAC purposes, treat UR as the *subject* and M as the *object*.
2. If $M \in M_{ij}^{RW}$, it has a CLS, M^{CLS}. Suppose that $Dominate(UR^{CLS}, M^{CLS}) = false$.
   Then, $URAM(UR, M) = 0$ (Definitions 14, 15a, and 15b).
3. This contradicts Step 1 where $URAM(UR, M) = 1$.
4. Therefore, $Dominate(UR^{CLS}, M^{CLS}) = true$ ($UR^{CLS} \geq M^{CLS}$).
5. This is exactly the Simple Integrity Property (write-down/no-write-up) [18].

Lemma 6.3: Let $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$ and $\alpha = \cup_{alli} URA_i$, where each URA_i is a VURA ($URAM(UR, M_j) = 1$ for some $j = 1..q$). If $Dominate(UR^{CLS}, M_j^{CLS}) = true$ for all $URA_i \in \alpha$, then UR satisfies the Simple Security Property for all methods and Simple Integrity Property for M^{RW}.
Proof:

1. For each $M \in M_{ij}^{RO}$ and $M \in M_{ij}^{RW}$, by Lemma 6.1, $Dominate(UR^{CLS}, M^{CLS}) = true$, and, UR satisfies the Simple Security Property for all methods.
2. For each $M \in M_{ij}^{RW}$, by Lemma 6.2, $Dominate(UR^{CLS}, M^{CLS}) = true$, and, UR satisfies the Simple Integrity Property for all M^{RW}.
3. Steps 1 and 2 mean that UR satisfies the Simple Security Property for all methods and the Simple integrity Property for M^{RW}.

Lemma 7: If $UA = [U, UR, TC]$ is a VUA, then UA satisfies the Simple Security and Integrity Properties for user (subject) CLR vs. role (object) CLS.
Proof by contradiction:

1. By Lemma 6.3, UR satisfies Simple Security Property for all methods and the Simple Integrity Property for M^{RW}. Hence, $Dominate(UR, M_k) = true$ for all M_k authorized to UR.
2. By Definitions 16 and 17a, $UA = [U, UR, TC]$ as a VUA means $UAM(UR, U) = 1$. By Definition 9, $U = [U^{UserId}, U^{LT}, U^{CLR}]$, with U^{CLR}.
3. Suppose that $Dominate(U^{CLR}, UR^{CLS}) = false$.
4. Then, $UAM(UR, U) = 0$ (Definitions 16, 17a, and 17b), which contradicts Step 2.
5. Therefore, $Dominate(U^{CLR}, UR^{CLS}) = true$ ($U^{CLR} \geq UR^{CLS}$).
6. Since UR satisfies both Simple Security and Integrity Properties, UA also does.

Lemma 8: If OU delegates one of its DUR, UR, to DU, then a delegated user authorization $DUA = [DU, UR, TC]$ satisfies both the Simple Security and Integrity Properties for delegated user (subject) vs. role (object).
Proof:

1. By Lemma 6.3, UR satisfies Simple Security Property for all methods and the Simple Integrity Property for M^{RW}. Hence, $Dominate(UR, M_k) = true$ for all M_k authorized to UR.
2. By Lemma 4, if an OU delegates to a UR to a DU, the available time is $DUA^{AT} = OUA^{AT} \cap DU^{LT}$. IF AT is $\emptyset$, Delegation cannot happen and the proof ends.

3. By Definition 22 and Definition 23, $UDAM(UR_i, U_j) = 2$. else, delegation cannot happen and proof ends.
4. For $UDAM(UR_i, U_j) = 2$ delegation, DU must be a valid user with a VUA, $UAM(UR_i, U_j) = 1$. For our purpose, DU = U and we continue in the same manner as Lemma 7.
5. By Definitions 16 and 17a, $UA = [U, UR, TC]$ as a VUA means $UAM(UR, U) = 1$. By Definition 9, $U = [U^{UserId}, U^{LT}, U^{CLR}]$, with U^{CLR}.
6. Suppose that $Dominate(U^{CLR}, UR^{CLS}) = false$.
7. Then, $UAM(UR, U) = 0$ (Definitions 16, 17a, and 17b), which contradicts Step 4.
8. Therefore, $Dominate(U^{CLR}, UR^{CLS}) = true$ ($U^{CLR} \geq UR^{CLS}$).
9. Since UR satisfies both Simple Security and Integrity Properties, UA also does.

In summary, Lemmas 6.1, 6.2, 6.3, 7, and 8, collectively, allow us to demonstrate that our RBAC/DAC/MAC model (and enforcement framework) will satisfy two important MAC properties.

5.3 Safety and Liveness Guarantees

In this section, we explore safety and liveness and their attainment with our framework. Our approach has been strongly influenced by research on *safety* (nothing bad will happen to a subject or object during execution) [62, 63] and *liveness* (all good can happen to a subject or object during execution) [5]. Remember, in our work, at different times the subject will be a user or a role, and the object will be a method or a role. To support the work in this section, in Section 5.3.1 we review a set of *security assurance rules, SARs*, that are intended to represent the conditions under which authorizations and delegations can occur (design time) and enforcement can occur (runtime) [67]. Then, in Section 5.3.2, we demonstrate the satisfaction of these rules in terms of both safety and liveness. Overall, the work in this section culminates the contributions of this dissertation by verifying the capabilities of the security model as presented in Chapters 3 and 4.

5.3.1 Security Assurance Rules

A *security assurance rule, SAR*, is a logical statement that must be satisfied in order for a privilege to be set (design-time) or an action to be performed (runtime). The first four SARs are non-delegation checks when a security officer is defining privileges (Rule I for when an officer attempts to assign a method to a UR and Rule II for when an officer attempts to authorize a UR to a User) and

at runtime (Rule III for when a user selects a UR to play for a session and Rule IV for when a user attempts to invoke a method, via a client application).

Rule I: Let $A \in UR$ and M be a method. $URA = [A, M, TC, SC]$ is a VURA iff $A^{CLS} \geq M^{CLS}$, $TC = A^{LT} \cap M^{LT} \cap TC \neq \emptyset$, $TC.et > ct$; Set $URAM(A, M) = 1$, $VURAL = VURAL \cup URA$.

Rule II: Let $A \in UR$ and $X \in UL$. $UA = [X, A, TC]$ is a VUA iff $X^{CLR} \geq A^{CLS}$, $TC = X^{LT} \cap A^{LT} \cap TC \neq \emptyset$, $TC.et > ct$; Set $UAM(A, X) = 1$, $UDAM(A, X) = 1$, $VUAL = VUAL \cup UA$.

Rule III: Let $A \in UR$ and $X \in UL$. A can be authorized to X at runtime iff $UAM(A, X) = 1$ (Rule II is satisfied), and for the $VUA = [X, A, TC] \in VUAL$, $TC.et > ct$.

Rule IV: Let $A \in UR$, $X \in UL$, and M be a method. X with role A can invoke M at runtime iff $UAM(A, X) = 1$ (Rule II is satisfied), $UDAM(A, M) = 1$ (Rule I is satisfied), for $VUA = [X, A, TC] \in VUAL$, $TC.et > ct$, for $VURA = [A, M, TC, SC] \in VURAL$, $TC.et > ct$, $SCOracle([M^{Name}, M^{Params}, M^{ActualValues}], SC) = true$.

Notice that SCOracle compares actual values of the invocation against the SC and returns true if M.ActualValues satisfies SC and false otherwise. For Rules III and IV, if the security information (i.e., UAM, UDAM, VUA, VURA, etc.) is up-to-date, there is no need to recheck Rules I and II; however, to guarantee a higher-level of assurance, rechecks of rules can be performed. Note that in support of SCOracle, we assert the following:

Assumption: There exists a SC Oracle that does not fail. The number and type of parameters vary with each method and SC will vary as well, to adapt to these different situations. This is a reasonable assumption since SC is simply checking to see if the Boolean is satisfied and this can be done with a very high expectation of success.

The remaining five SARs are for delegation [67]. Rules V and VI are for the assignment of DA and PODA to a user X. Rule VII is for design-time delegation of a UR by a user to another user. Rule VIII is for setting DA or DA/PODA from one user (either OU or DU) to another DU (Rule VII satisfied).

Rule V: Let $X \in UL$ and $A \in URL$. X can have DA for A ($DAM(A, X) = 1$) iff $UDAM(A, X) = 1(X \text{ an } OU)$, $DURV(A) = 1(A \text{ a } DUR)$, and $\exists$ a $VUA = [X, A, TC]$.

Rule VI: Let $X \in UL$ and $A \in URL$. X can have DA and PODA for A ($DAM(X, A) = 2$) iff $UDAM(A, X) = 1(X \text{ an } OU)$, $DURV(A) = 1(A \text{ a } DUR)$, and $\exists$ a $VUA = [X, A, TC]$.

Rule VII: Let $X \in UL$ and $A \in URL$, s.t. $DAM(A, X) \geq 1$ (Rules V or VI). X can delegate A to user Y limited by TC iff $UDAM(A, Y) \neq 1, Y^{CLR} \geq A^{CLS}, TC = (Y^{LT} \cap A^{LT} \cap TC) \neq \emptyset$, and $TC.et > ct$. Set $UAM(A, Y) = 1, UDAM(A, Y) = 2$, and $VUAL = VUAL \cup UA = [Y, A, TC]$.

Rule VIII: Let $X \in UL$ be an OU/DU, $A \in URL$, and $Y \in UL$ be a DU of A. Y can have DA for A (DAM(A,Y) = 1) if X has at least DA for A (DAM(A,X) $\geq$ 1). 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). (Rule VIII limited to 2 levels in our framework).

5.3.2 Proof of Safety and Liveness

In this section we present a series of theorems related to safety and liveness, which correspond to the rules that have been presented in Section 5.3.1. For consistency in our use of terms, we define the following: a *legal access* is an access authorized by a security policy; *liveness* means that every legal access is authorized (all good things can happen) [5]; and, *safety* means that no illegal access is authorized (no bad things will happen) [62, 63]. To begin, consider Theorems I and II below, which correspond to Rules I and II as given in Section 5.3.1.

Theorem I: Rule I meets security requirements for both liveness and safety.
Rule I: Let $A \in UR$ and M be a method. $URA = [A, M, TC, SC]$ is a VURA iff $A^{CLS} \geq M^{CLS}$, $TC = A^{LT} \cap M^{LT} \cap TC \neq \emptyset$, $TC.et > ct$; Set $URAM(A, M) = 1$, $VURAL = VURAL \cup URA$.
Proof:

1. Let A be a user role and M be the method being authorized.
2. Initially, the URAM is set to all 0s. This is perfect access control as there are no UR authorizations to any methods, M_{ijk}, in this system. This meets safety (no bad things can happen), but not liveness, as no good things can happen.
3. Lemma 1 ensures that a URA is a VURA and $URA_i^{AT} = M^{LT} \cap UR^{LT} \cap TC$. If $URA_i^{AT} \neq \emptyset$ and $URA_i^{AT} \cap [ct, \infty] \neq \emptyset$, then M can be assigned to UR. Otherwise M cannot be assigned to UR because it can never be available.
4. Lemma 6.1 insures $A^{CLS} \geq M^{CLS}$.
5. To change an entry from 0 to 1 in a URAM, there must be a URA (Definition 14).
6. In order to build liveness, yet maintain safety, 0 entries in the URAM must be changed to 1 ($URAM(A, M) = 1$) for only those URs authorized to Ms (Definitions 15a and 15b). Validating changes to URAM

allows for both safety and liveness; there is no access except for these specific authorizations (all legal, no illegal access) and this is precisely Rule I.

Theorem II: Rule II meets security requirements for both liveness and safety.
Rule II: Let $A \in UR$ and $X \in UL$. $UA = [X, A, TC]$ is a VUA iff $X^{CLR} \geq A^{CLS}$, $TC = X^{LT} \cap A^{LT} \cap TC \neq \emptyset$, $TC.et > ct$; Set $UAM(A, X) = 1$, $UDAM(A, X) = 1$, $VUAL = VUAL \cup UA$.
Proof:

1. Let X be a user and A be the role being authorized.
2. Initially, the UAM and UDAM are set to all 0s. This is perfect access control as there are no users, U, authorized to any user roles, UR, in this system. This meets safety requirements, but not liveness, as no user can do anything.
3. U and UR must be available for access and $UR^{AT} \cap U^{AT} \cap [ct, \infty] \neq \emptyset$. Lemma 3 insures the maximum AT for a U to a UR. The intersection with the current time assures that available time has not passed, which is obviously a requirement for use.
4. Lemma 7 insures $X^{CLR} \geq A^{CLS}$.
5. To change an entry from 0 to 1 in UAM/UDAM, there must be a UA (Definition 16).
6. In order to build liveness, yet maintain safety, 0 entries in the UAM/UDAM must be changed. Validating changes to UAM and UDAM allows for both safety and liveness, as there is no access allowed except for these specific authorizations and this is precisely Rule II.

Both Theorems I and II are for the design time verification of security privileges as authorizations of being defined.

Theorems III and IV are for runtime checks of privileges against the RBAC/DAC/MAC model. The safety and liveness of Rule III (Theorem III) focuses on the runtime authorization of user to role. This is necessary for safety and liveness in a dynamic environment where tranquility of CLR and CLS is not guaranteed, URs can change, and user and time constraints can expire.

Theorem III: Rule III meets security requirements for both liveness and safety.
Rule III: Let $A \in UR$ and $X \in UL$. A can be authorized to X at runtime iff $UAM(A, X) = 1$ (Rule II is satisfied), and for the $VUA = [X, A, TC] \in VUAL$, $TC.et > ct$.
Proof:

1. Let X be a user and A be the role being authorized.
2. If $UAM(A, X) \neq 1$, then deny runtime authorization. End of proof.

3. Revalidate Rule II: $X^{CLR} \geq A^{CLS}$, $TC = X^{LT} \cap A^{LT} \cap TC \neq \emptyset$, $TC.et > ct$.
   - If $X^{CLR} < A^{CLS}$, then Rule II fails.
   - If $TC = X^{LT} \cap A^{LT} \cap TC = \emptyset$, then Rule II fails.

   If either case is true, deny runtime authorization. End of proof.
4. Otherwise, Rule II is revalidated at runtime, then by Theorem II, Rule II satisfies safety and liveness. Thus, Rule III does as well.

Recall that Rule IV (Section 3.3.1) checks to see if the invocation of a method by a user playing a specific role can occur. As such, there must be re-verifications of privileges and constraints in a similar fashion to Rule III/Theorem III.

Theorem IV: Rule IV meets security requirements for both liveness and safety.
Rule IV: Let $A \in UR$, $X \in UL$, and M be a method. X with role A can invoke M at runtime iff $UAM(A, X) = 1$ (Rule II is satisfied), $UDAM(A, M) = 1$ (Rule I is satisfied), for $VUA = [X, A, TC] \in VUAL$, $TC.et > ct$, for $VURA = [A, M, TC, SC] \in VURAL$, $TC.et > ct$, $SCOracle([M^{Name}, M^{Params}, M^{ActualValues}], SC) = true$.
Proof:

1. Let X be a user, A be its role, and M be the method being invoked.
2. If $SCOracle([M^{Name}, M^{Params}, M^{ActualValues}], SC) = false$, then deny runtime authorization. End of proof.
3. If $UAM(A, X) \neq 1$, then deny runtime authorization. End of proof.
4. If $URAM(A, M) \neq 1$, then deny runtime authorization. End of proof.
5. If $U^{AT} = \emptyset$, (set by Lemma 3) then deny runtime authorization. End of proof.
6. If $U^{TC.et} \leq ct$, then deny runtime authorization. End of proof.
7. Otherwise, Rules I and II are revalidated enusring both safety and liveness. Thus, Rule IV does as well.

Note that since U^{AT} by Lemma 3 includes UR^{AT} which includes M^{LT} by Lemma 1, there is no need to consider the latter two for the proof. We present the remaining five theorems for role delegation. You will find that the following proofs are similar to the first four proofs in that the users and user-roles still need to meet and often time are more constrained with delegation. Certainly, delegated users need to be held to at least the same security assurance standards set for an original user.

Rule V is for assigning delegation authority, DA, for a user role, UR at design time. The proof of Theorem V verifies delegation authority by examining whether the UR is delegatable and whether the user is already a delegated or an original user of a UR. The proof also examines for a valid user authorization and time

constraints like Lemma 2, then proceeds to verification of delegation authority by examining whether the UR is delegatable and whether the user is already a delegated or an original user of a UR.

Theorem V: Rule V meets security requirements for both liveness and safety.
Rule V: Let $X \in UL$ and $A \in URL$. 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 $VUA = [X, A, TC]$.
Proof:

1. Let X be a user and A is a $UR, VUA(UAM(A, X) = 1)$. Definitions 17a,b. In order for $UAM(A, X) = 1$, Rule II must be satisfied (Theorem II). $UDAM(A, X) = 1$ implies that $UAM(A, X) = 1$ also by Rule II.
2. If $VUA(UAM(A, X) \neq 1)$ there can be no delegation authority assigned because X is not a valid user to begin with. End of proof.
3. Initially, the $DURV, DAM$, and $UDAM$ are set to all 0s. This is perfect safety as there are no users authorized to delegate any user roles, UR, in this system. This, of course, meets safety requirements, but not liveness, as delegation cannot happen. Definitions 20, 26 and 23.
4. If $DURV(A) = 0$ or $UDAM(A, X) = 0$ or $DAM(A, X) = 0$ there can be no delegation authority assigned because one of the following is true: The UR is not delegatable (Definition 20), the user does not have delegation authority (Definition 26), or the user is not authorized to user role (Definition 23). All of which prevent delegation. End of Proof.
5. Therefore, the only way delegation authority can be assigned to X is if $DURV(A) \neq 0$ and $UDAM(A, X) \neq 0$ and $DAM(A, X) \neq 0$, which is Rule V. Since only thing that can happen is delegation authority, which is good, we prove liveness. The only thing that can happen is good, nothing bad can happen, which proves safety.

Theorem VI for proving Rule VI is very similar to Theorem V with the difference in the consideration of the Delegation Authority Matrix, DAM (Definition 26). Rule VI is for assigning delegation authority, DA, and Pass-on Delegation Authority, $PODA$, at design time. $UDAM(A, X) = 1$ implies that $UAM(A, X) = 1$ by Rule II. Note that Rule VI establishes, DA or $DA/PODA$ for user X to role A when X is an OU.

Theorem VI: Rule VI meets security requirements for both liveness and safety.
Rule VI: Let $X \in UL$ and $A \in URL$. X can have DA and PODA for A ($DAM(X, A) = 2$) iff $UDAM(A, X) = 1(X$ an $OU)$, $DURV(A) =$

$1(A \text{ a } DUR)$, and $\exists$ a $VUA = [X, A, TC]$.

1. Let X be a user and A is a $UR, VUA(UAM(A, X) = 1)$. Definitions 17a,b. In order for $UAM(A, X) = 1$, Rule II must be satisfied (Theorem II). $UDAM(A, X) = 1$ implies that $UAM(A, X) = 1$ also by Rule II.
2. If $VUA, (UAM(A, X) \neq 1)$, there can be no delegation authority assigned because X is not a valid user to begin with. End of proof.
3. Initially, the $DURV, DAM, andUDAM$ are set to all 0s. This is perfect safety as there are no users authorized to delegate any user roles, UR, in this system. This, of course, meets safety requirements, but not liveness, as delegation cannot happen. Definitions 20, 26 and 23.
4. If $DURV(A) = 0$ or $UDAM(A, X) = 0$ or $DAM(A, X) = 0$ there can be no delegation authority assigned because one of the following is true: The UR is not delegatable (Definition 20), the user does not have delegation authority (Definition 26), or the user is not authorized to user role (Definition 23). All of which prevent delegation. End of Proof.
5. Therefore, the only way delegation authority can be assigned to X is if $DURV(A) \neq 0$ and $UDAM(A, X) \neq 0$ and $DAM(A, X) \neq 0$, which is Rule V.
6. Since the only thing that can happen is delegation authority, which is good, we prove liveness. Since the only thing that can happen is good, so nothing bad can happen, which proves safety.

Theorem VII is similar to and uses Theorems II, IV, and VI. Theorem VII for Rule VII is for executing the actual delegation of a role from one user to another user. In order to be able to delegate, the delegating user must have delegation authority, the user-role must be delegatable and the gaining user must be a valid user. This Proof examines how the user must have proper authority or the delegation cannot happen. Theorems II and IV are for checking for valid user authorizations at design time and runtime, respectively. Next, the proof examines the authority of the delegating user to delegate a role. Rule VII is for the design-time delegation of a UR by a user to another user, which creates entries $UAM(A, Y) = 1, UDAM(A, Y) = 2$, and $VUAL = VUAL \cup UA = [Y, A, TC]$. if successful.

Theorem VII: Rule VII meets security requirements for both liveness and safety.

Rule VII: Let $X \in UL$ and $A \in URL$, s.t. $DAM(A, X) \geq 1$ (Rules V or VI). X can delegate A to user Y limited by TC iff $UDAM(A, Y) \neq 0, Y^{CLR} \geq A^{CLS}, TC = (Y^{LT} \cap A^{LT} \cap TC) \neq \emptyset$, and $TC.et > ct$. Set

$UAM(A,Y) = 1, UDAM(A,Y) = 2$, and $VUAL = VUAL \cup UA = [Y, A, TC]$.
Proof:

1. User, Y must satisfy the requirements of Theorems II and III. This insures the user meets the same TC, SC and $MACC$ requirements as the delegating user. Else, no delegation, end of proof.
2. User Y cannot be an OU or DU of the delegated UR, $(UDAM(A,Y) \ngeq 1)$. If the user Y is already a DU or OU, the delegation cannot happen. End of proof.
3. Let X be delegating user $(DAM(A,X) \geq 1)$ and A a delegatable UR, $(DURV(A) = 1)$. If U is not authorized to delegate or UR not delegatable, delegation cannot occur, end of proof.
4. If $VUA, (UAM(A,X) \neq 1)$, there can be no delegation authority assigned because X is not a valid user to begin with. End of proof.
5. Initially, the $DURV, DAM$, and $UDAM$ are set to all 0s. This is perfect safety as there are no users authorized to delegate any user roles, UR, in this system. This, of course, meets safety requirements, but not liveness, as delegation cannot happen. Definitions 20, 26 and 23.
6. If $DURV(A) = 0$ or $UDAM(A,X) = 0$ or $DAM(A,X) = 0$ there can be no delegation authority assigned because one of the following is true: The UR is not delegatable (Definition 20), the user does not have delegation authority (Definition 26), or the user is not authorized to user role (Definition 23). All of which prevent delegation. End of Proof.
7. Therefore, the only way delegation authority can be assigned to X is if $DURV(A) \neq 0$ and $UDAM(A,X) \neq 0$ and $DAM(A,X) \neq 0$, $Y^{CLR} \geq A^{CLS}$,$TC = (Y^{LT} \cap A^{LT} \cap TC) \neq \emptyset$, and $TC.et > ct$ which is Rule VII.
8. Since delegation authority can happen, which is good, we prove liveness. Since the only thing that can happen is good, nothing bad happens, which proves safety.

Rule VIII is for executing $PODA$ of a UR from an OU to a DU to another DU. This proof is similar to Theorem VII in that the DU must meet the VUA from Lemmas 3 and 7 and Theorem II. In addition to Theorem VII, OUs and DUs are validated according to the Delegation Authority Matrix, DAM, Definition 26. In order for the delegation to occur, Rule VII requires Role Delegation and User DA Authority Checks, and MACC Domination Check, LT/TC Check, with the overlap of LTs and TC after the current time. Note that Rule VII is utilized to

establish that Y is a DU of A, if X has at least DA for A (X satisfies Rules V or VI).

Theorem VIII makes sure that an OU/DU must grant PODA before a DU can grant DA. In our framework, we can limit the delegation to two vertical by only allowing an OU to grant $PODA$. That way a DU can only DA to another U.

Theorem VIII: Rule VIII meets security requirements for both liveness and safety.

Rule VIII: Let $X \in UL$ be an OU/DU, $A \in URL$, and $Y \in UL$ be a DU of A. Y can have DA for A ($DAM(A,Y) = 1$) if X has at least DA for A (DAM(A,X) $\geq$ 1). 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). (Rule VIII limited to 2 levels in our framework).

Proof:

1. User, Y must satisfy the requirements of Theorems II and III. This insures the user meets the same TC, SC and $MACC$ requirements as the delegating user. Else, no delegation, end of Proof
2. User Y cannot be an OU or DU of the delegated UR, ($UDAM(A,Y) \ngeq 1$). If the user Y is already a DU or OU, the delegation cannot happen. End of Proof.
3. Let X be delegating user ($DAM(A,X) \geq 1$) and A a delegatable UR, ($DURV(A) = 1$). If U is not authorized to delegate or UR not delegatable, delegation cannot occur, end of proof.
4. If $VUA, (UAM(A,X) \neq 1)$, there can be no delegation authority assigned because X is not a valid user to begin with. End of proof.
5. Initially, the $DURV, DAM$, and $UDAM$ are set to all 0s. This is perfect safety as there are no users authorized to delegate any user roles, UR, in this system. This, of course, meets safety requirements, but not liveness, as delegation cannot happen. Definitions 20, 26 and 23.
6. If $DURV(A) = 0$ or $UDAM(A,X) = 0$ or $DAM(A,X) = 0$ there can be no delegation authority assigned because one of the following is true: The UR is not delegatable (Definition 20), the user does not have delegation authority (Definition 26), or the user is not authorized to user role (Definition 23). All of which prevent delegation. End of Proof.
7. Therefore, the only way delegation authority can be assigned to X is if $DURV(A) \neq 0$ and $UDAM(A,X) \neq 0$ and $DAM(A,X) \neq 0$, $Y^{CLR} \geq A^{CLS}$,$TC = (Y^{LT} \cap A^{LT} \cap TC) \neq \emptyset$, and $TC.et > ct$ which is Rule VII.

8. If $PODA$ is to occur from X to Y, X must have $PODA$, $DAM(A, X) < 1$, Def 26. If $DAM(A, X) \neq 2$, $PODA$ cannot happen, End of Proof.
9. We know delegation authority can happen from Theorem VII iff $DAM(A, X) \neq 0$ and we also know $PODA$ can happen iff $DAM(A, X) = 2$, which is good, proving liveness. Since only good things can happen when $DAM(A, X) \neq 0$, nothing bad happens, which proves safety.

5.3.3 Summary of Rules and Theorems

As stated earlier, our main objective in this chapter is to explore assurance for our RBAC/DAC/MAC security model. We focus on which methods of APIs can be invoked based on the user role attributes, the security classification and clearances, method parameter values, and time constraints, all of which must be satisfied in order for the invocation to be successfully. We leverage the best qualities and principles established by role-based, discretionary, and mandatory access control methods to include the Simple Security and Simple Integrity Properties required in government systems for protecting access to sensitive information. We also consider the required balance between safety (no bad things can happen) and liveness (all good things happen) that must be achieved in spite of the strict controls imposed by mandatory access controls and the user freedom provided by discretionary access controls. We accomplish all this by establishing a strict set of security assurance rules that enforce the necessary qualities. We conclude the chapter by formally proving that each rule, within the context of our security model, can provide both safety and liveness in support of security assurance. This is one of the significant contributions of this dissertation.

Chapter 6

Security Enforcement: Framework and Prototype

In this chapter, we present the security enforcement framework and prototype that realizes our RBAC/DAC/MAC security model as given in Chapters 3 and 4, which includes the security assurance capabilities as detailed in Chapter 5. We separate our presentation into two parts. First, we explore the security enforcement framework that is built upon our abstract middleware model (see Chapter 3), and the realization of the RBAC/DAC/MAC security model as: a set of security tools that allow security officers to define a security policy; the definition of security services that support the RBAC/DAC/MAC model within an abstract middleware framework; and, the associated processing steps that occur when a user of a client application, playing a role, attempts an action in the client that causes a method invocation against a software artifact (resource). Second, we examine the actual prototyping effort that has been ongoing over the past four plus years. The prototype has been developed as part of numerous independent studies and design laboratories by graduate students, including: Spring 2000 (two MS students) Fall 2000 (four MS students), Spring 2001 (six MS students), Summer 2001 (two MS students), Fall 2001 (three MS students), Spring 2002 (three MS students) and Fall 2002 (five MS students), Spring 2003 (one MS student). The prototyping effort, in almost its entirety, has been designed and supervised by C. Phillips.

The remainder of this chapter is organized into three sections. In Section 6.1, we explore the underlying concepts of the security enforcement framework that coincides to the RBAC/DAC/MAC model as presented in Chapters 3 and 4, which includes assurance capabilities as discussed in Chapter 5. In Section 6.2, we emphasize the prototype of the framework from Section 6.1, which includes security administrative tools to define and manage the security policy (e.g., authorize methods to roles, users to roles, delegation, assign CLS and CLR, etc.) and the interactions that occur within the enforcement framework at runtime when a user playing a role tries to invoke a method of a client application. We also

include a discussion of the client applications (and associated database resources - artifacts) that have been developed to demonstrate the concepts.

6.1 Security Enforcement Framework

In this section, we present a security enforcement framework for a combined RBAC/DAC/MAC security model, which enforces selective access of users (executing client applications) to the software artifacts (resources) that comprise a distributed application, as shown in Figure 25. The associated security administrative tools and enforcement framework for this research are shown in the bottom half of Figure 25. The enforcement framework, the Unified Security Resource (USR), is modeled as another resource in a middleware setting, that is dedicated to handling security interactions, and embodies the capabilities of the RBAC/DAC/MAC model as discussed in Chapters 3 and 4, and the assurance needed in support of the material in Chapter 5. The USR consists of three sets of services: *Security Policy Services* to manage the definition of roles and the authorization of their privileges (i.e., methods); *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. To provide an interaction point for security officers seeking to define and manage their security policy, the enforcement framework as given in Figure 25 also depicts: 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; and the *Security Delegation Client (SDC)* for use by the security officer and users to grant, update, and revoke delegations.

In this section, we emphasize conceptual issues and detail the capabilities of the enforcement framework from a number of different perspectives. First, we explore the Unified Security Resource (USR) in detail. Second, we present security enforcement actions that occur for an active client. Third, we examine the participation of software artifacts (resources) in the security paradigm, which includes the actions taken for a resource to register its services for secure access. Fourth, we discuss the capabilities of the security administration and management tools (SPC, SAC, and SDC). Fifth, we detail security assurance checks that are provided at design and runtime, and we finish by discussing related prototyping research. research prototype, and related prototype work.

6.1.1 The Unified Security Resource (USR)

The Unified Security Resource (USR) consists of three sets of services: **Security Policy Services** managing roles and their privileges; **Security Authorization Services** to authorize roles to users; and, **Security Registration**

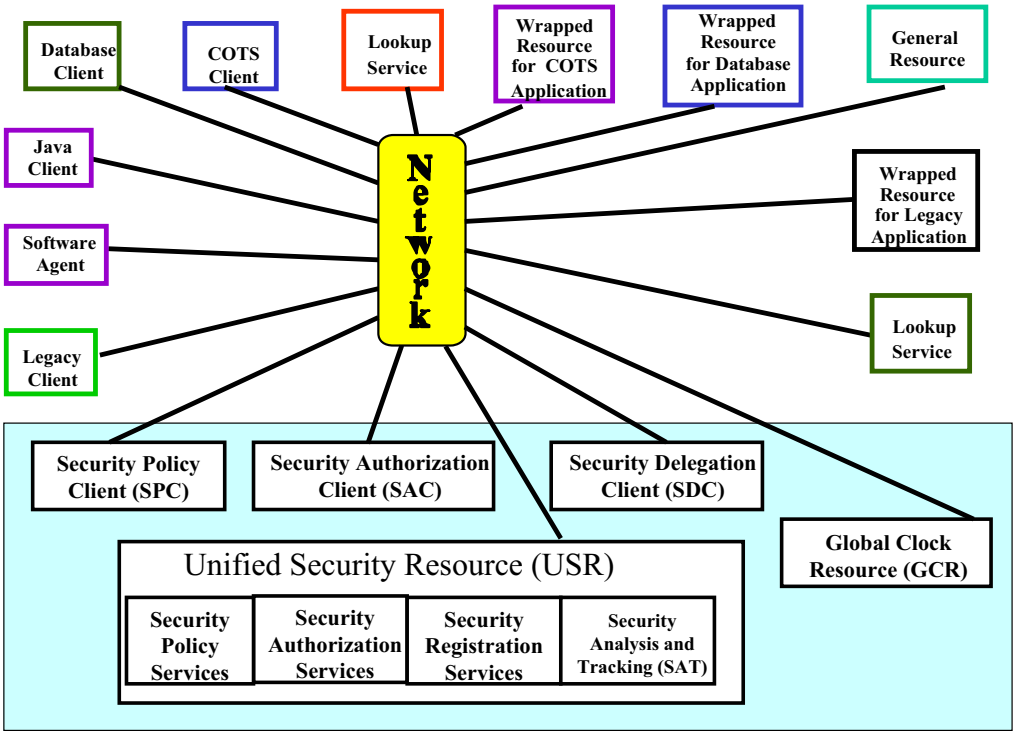


Figure 25: Security Enforcement Framework Software Architecture.

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., and is organized into a set of services, as given in Figure 26. **Security Policy Services** are utilized to define, track, and modify user roles, to allow resources to register their services and methods (and signatures), and to grant/revoke access by user roles to resources, services, and/or methods with optional time and signature constraints. These services are used by a security officer to define a policy, and by the resources (e.g., database, Java server, etc.) to dynamically determine if a client has permission to execute a particular [resource, service, method] under a time and/or signature constraint. There are five different **Security Policy Services**: *Register* for allowing a resource to (un)register itself, its services and their methods (and signatures), which is used by a resource for secure access to its services; *Query Privilege* for verification of privileges; *User Role* to allow the security officer to define and delete user roles; *Constraint* to allow time and signature constraints to be defined by the security officer, and for these constraints to be dynamically verified at runtime; and, *Grant-Revoke* for establishing privileges.

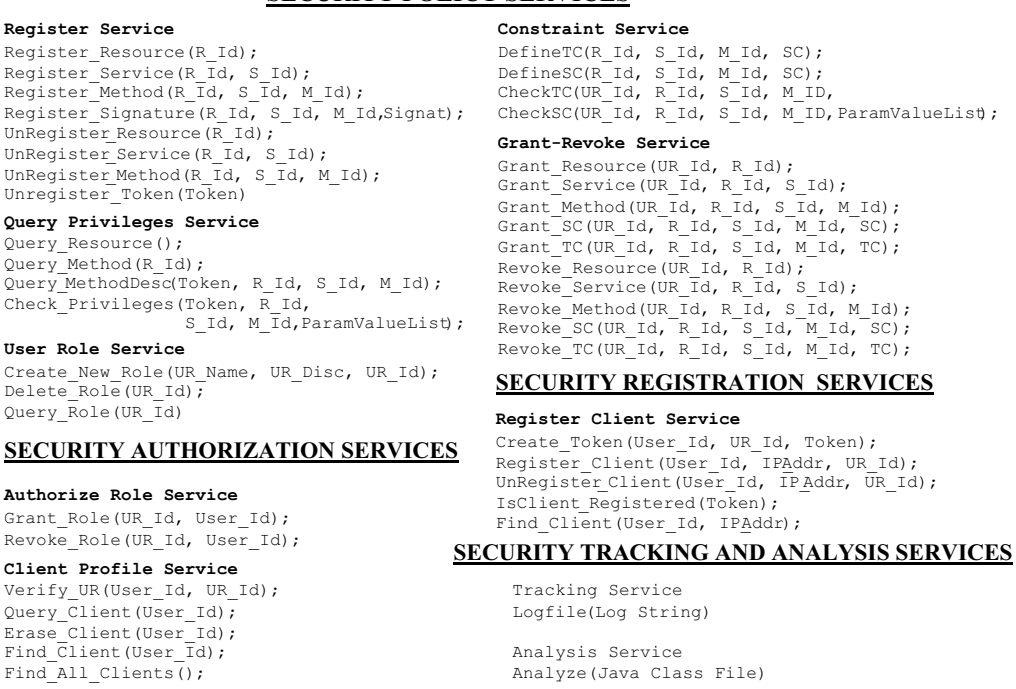


Figure 26: The Services of USR.

Security Authorization Services are utilized to maintain profiles on the clients (e.g., users, tools, software agents, etc.) that are authorized and actively utilizing non-security services, allowing a security officer to authorize users to roles. There are two services: *Authorize Role Service*, for the security officer to grant and revoke a role to a user with the provision that a user may be granted multiple roles, but must play only a single role when utilizing a client application; and, *Client Profile Service* for the security officer to monitor and manage the clients that have active sessions. **Security Registration Services** are utilized by clients at start-up for identity registration (client id, IP address, and user role), which allows a unique Token (see Definition 18) to be generated for each session of a client. The two Security Registration services are: The *Register Client Service* and the *Security and Analysis Services*. The tracking and analysis services allow for a security officer to follow all activity for reasons of nonreputiation and recovery. Finally, the **Global Clock Resource** (GCR) , Figure 25, is used by Security Policy Services to verify a TC when a client (via a UR) is attempting to invoke a method, and Security Registration Services to obtain the time, which is then used in the generation of a unique Token.

6.1.2 Security Enforcement: Client Application

The processing required by a client application is best illustrated with an example, which is given in Figure 27. In the first step in the process, the user of the GCCS Client must be authenticated to play a particular role. To do so, the GCCS Client registers with USR via the Register_Client method (step 1), which must verify the user role (steps 2 and 3), and return a generated Token via the Create_Token method (step 4). To generate the Token in step 4, the Security Registration Service interacts with GCR (steps not shown) in order to construct a Token which consists of User-Id, UR-ID, IP address, and creation time. This token is unique even if the user has multiple active sessions of same role on one machine. Assuming that the registration and Token generation was successful, the user can then attempt to utilize services from the GCCS Resource. The GCCS Client consults the lookup service for the desired [resource, service, method] (step 5) which returns a proxy to CrisisPicture, allowing the method to be invoked (step 6) with the parameters Token, CR1, NA20, and NC40. The GCCS Resource has two critical steps to perform before executing CrisisPicture. First, the GCCS Resource verifies that the Client has registered with the security services (steps 7 and 8). If this fails, a negative result is sent back via the RegisterCourse result (step 11). If this is successful, then the GCCS resource must perform a privilege check (privileges assigned by role) to verify if the user role can access the method limited by signature constraints and/or time constraints (both may be null). This is done in step 9, as part of the Check_Privileges method, which uses method calls from Figure 11 such as: 9a CheckTC (Token, UnivDB, Modification, RegisterCourse); 9b CheckSC (Token, UnivDB, Modification, RegisterCourse, ParamValueList). The CheckTC method interacts with GCR to verify that the current time is within the limits of the time constraint (if present). If the privilege check is successful (step 10) then the method executes as called and the result (DoRight executing CrisisPicture) is returned as a success in step 11. Otherwise, the result (step 10) denies the registration via step 11. Note that the signature constraint is verified in two phases. The parameters constraint (if present) must be checked prior to method invocation, while the return-type constraint (if present) must be checked after execution and before the result has been returned.

6.1.3 Participation of Resources in Security Paradigm

In our enforcement framework in Figure 25, resources that want to utilize the security capabilities, must announce themselves to USR. Specifically, each resource must register with the Security Policy Services, so that the master resource list, service list, and method list (and their signatures) can be modified to include resources as they dynamically enter and leave the environment. Resources must be allowed to both register and unregister themselves, their services, and methods (with signatures) via the Security Registration Service (see Figure 25

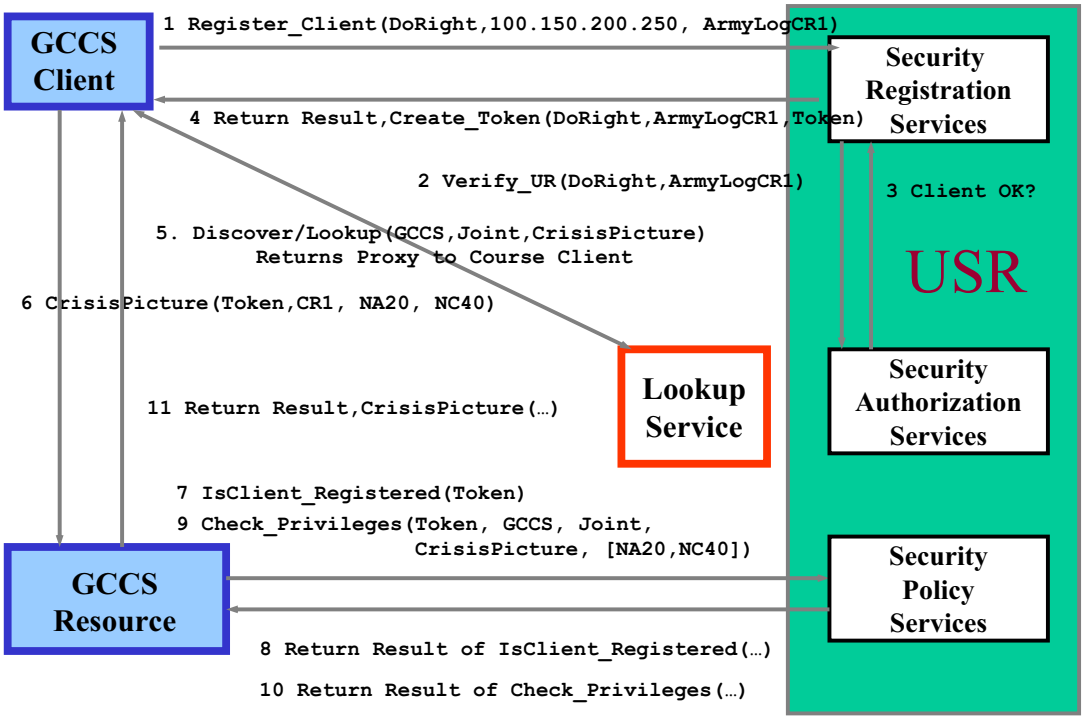


Figure 27: Client Interactions and Service Invocations.

and Figure 26). For our example, the GCCS resource, Chapter 3, Figure 13, would register its Joint and Component services and all of their methods and signatures. This registration is critical, since it stores the information into a relational database which is then accessible by the administrative and management tools. In addition, resources like GCCS utilize USR to dynamically verify if the client is authorized to a [resource, service, method] under optional TCs and/or SCs, Definitions 11 and 12.

6.1.4 Security Administrative/Management Tools

Recall that in Figure 25, there are administrative and management tools, namely: the *Security Policy Client* (SPC) manages user roles by granting/revoking privileges (TCs, resources, services, methods, and SCs); the *Security Authorization Client* (SAC) authorizes roles to end users; a *Security Analysis and Tracking Tool* (SAT) analyzes Java source code of resources to track nested method invocations; and, a *Global Clock Resource* (GCR) supports time-constrained access to

resources (and their services/methods) by role. In the remainder of this section, we focus the discussion on the three tools (SPC, SAC, and SAT) to illustrate the techniques and processes that are utilized to engineer the security for a distributed application using our security model and enforcement framework.

The Security Policy Client (SPC) can be used to define and remove user roles, and to grant and revoke privileges (i.e., time constraints, resources, services, methods, and/or signature constraints). The security officer can also inspect and monitor security via the tracking capabilities of SPC. The results track status of the attempted access (Success/Error) of the method (method-id) of a resource (resource-id) by the particular user (user-id) playing a specific role (role-id).

The Security Authorization Client (SAC) supports authorization of role(s) to users. A user may hold more than one role, but can only act in one role at a time. Playing multiple roles at the same time may lead to conflicts if different SCs and/or TCs are defined on the same resource, service, or method. If more broad access is desired, a role with expanded capabilities can be defined to encompass privileges of multiple roles without SC/TC conflicts. The SAC provides the security officer with the ability to create a new User, which has an identifier, password, a time constraint (begin/end date) that indicates the valid time interval for the user, and a text description.

The Security Delegation Client (SDC) 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 original user will be able to invoke SDC. Grant allows the user to choose a delegated user, the delegatable roles, set the lifetime, and authorize re-delegation. Update allows a user to modify any existing delegation, while Revoke enables the user to cancel any delegation. The delegation of a role by a user is similar to the security officer assigning roles. Delegation works in conjunction with the other elements of the Unified Security Resource [67], and utilizes the same underlying databases. With SDC, an organization has the flexibility to give certain users delegation authority, while still maintaining administrative control and security assurance.

Finally, the **Static Analysis Tool** (SAT), is utilized by the security officer to analyze source code of Java resources, allowing us to track not only the method on a resource that has been directly authorized to a role, but also the other resources (and services/methods) that are called. Given the directory location of a Java source code, SAT analyzes a class by inspecting all of the method definitions to find any other method called inside the one under inspection. The SAT tracks the information on the method being analyzed, the methods that are invoked by the method, and the user roles assigned to the method. As mentioned earlier, this gives a security officer the capability to track user activity for both recovery purposes and non-reputation. An important consideration in security assurance is the ability to hold an individual accountable (non-repudiation)for actions and to know at what point recovery must be initiated.

6.1.5 Security Assurance Checks in the Prototype

The administrative tools and enforcement framework for the RBAC/DAC/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 runtime (for enforcing the delegation and revocation rules, and the critical delegation concepts in Chapter 4). 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 (Chapter 4), the lifetime of the DU must be within the lifetime of the delegating user (OU or DU), verified at design time.

5. **Modified Boolean Delegation**: In regard to the levels of delegation criteria (Chapter 4), 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.

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 runtime to any user that meets the prerequisite conditions of the role. This rule must also be checked during runtime.

4. **Lifetime Consistency**: In regard to the permanence criteria (Chapter 4), the lifetime of the DU must be within the lifetime of the OU. 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 (Chapter 4), 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 runtime, 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 [138], 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 runtime.

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 runtime.

6.1.6 Related Prototyping Research

There are many ongoing developmental and prototyping projects in different areas that have a relationship to our security enforcement framework and associated prototype. The National Institute of Standards and Technology (NIST)

Model, developed in 1992, was an attempt to get away from the DAC/MAC model, which was considered as inappropriate for many, commercial and civilian government needs [39]. The NIST Model has strong parallels with our research, and is a good reference for the concepts of least privilege, separation of duty, static constraints (for mutual exclusion and separation of duty), dynamic constraints (conflict-of-interest) and role hierarchy (for administration), along with other accepted RBAC concepts. The NIST Model was improved in 1998 [45] to be a hybrid between the RBAC96 model [111] and the original 1992 NIST Model. The improvements include improved administration tools (higher assurance) and improve database consistency checks. However, there are some critical differences between their approach and our own efforts. The NIST Model is not multi-level secure, is not readily portable between UNIX and NT platforms, is primarily object based, and does not achieve the same high level of security granularity of our model/framework. In addition to these efforts, there have been other related initiative and products that are addressing similar problems to our research. One Department of Defense and NATO effort is the Command Control Systems Interoperability Program (C2SIP) to bring NATO forces together using a database engine that accepts any NATO country formats [1]. The Air Force Research Laboratory in conjunction with Verdian is working on a comprehensive information tagging and release policy called Secure Information Releasability Environment [119]. There are products like e-Portal and Multi-domain Dissemination System, which concentrates on sensitive information access using secure transmission [119]. In addition, there are systems that use firewall technology to create a secure network connections between hosts on any unclassified network [60].

6.2 Security Enforcement Prototype

This section reviews the proof-of-concept prototyping efforts for our RBAC/DAC/MAC model and enforcement framework as shown in Figure 25. The objectives of the section are to familiarize the reader with the prototype interfaces and show consistency between the security model, enforcement framework and a prototype. We have designed and implemented the entire security framework with two different resources and two separate lookup services, Figure 28. Our prototype includes the USR, administrative and management tools, and global clock resource which are all capable of interacting with either CORBA or JINI as the lookup middleware. The current version of USR implements all of the services from Figure 26. To verify the security framework, we prototyped two distributed applications, consisting of a Java client and associated database resource. We have utilized a university application and have also prototyped a hospital application where a client can access information in a Patient DB Figure 28. The hospital portion of the figure (left side) interacts with CORBA as the lookup middleware; the university portion (right side) uses JINI. Note that

the middleware lookup service is transparent to the user; however, a client could be constructed to use CORBA and/or JINI depending on which one is active in the network. Note that details on both applications are given in Section 6.4.

From a technology perspective, the university application in Figure 28(right side) is realized using Java 1.3, JINI 1.1, Windows NT 4.0 and Linux, and Oracle 8.1.7. The hospital application (left side) uses the same technologies except for Visibroker 4.5 for Java as middleware. Both of the resources (Patient DB and University DB) operate in a single environment using the same security system transparently, and both resources are designed to allow registration of their services with both CORBA and JINI. The University DB Java Client allows students to query course information and enroll in classes, and faculty to query and modify the class schedule. The Patient DB Java Client supports similar capabilities in a medical domain. The individual methods associated with each resource are also depicted in Figure 28. Additionally, the prototype includes administrative and management tools, namely: the Security Policy Client (SPC) manages user roles by granting/revoking privileges (TCs, resources, services, methods, and SCs); the Security Authorization Client (SAC) authorizes roles to end users; a Static Analysis Tool (SAT) (not shown) analyzes Java source code of resources to track nested method invocations; and, a Global Clock Resource (GCR) supports time- constrained access to resources (and their services/methods) by role. In the remainder of this section, we focus the discussion on the three tools (SPC, SAC, and SAT) to illustrate the techniques and processes that are utilized to engineer the security for a distributed application using our model and enforcement framework. We conclude with a short overview of one of the prototyped resources.

The remainder of this section is organized into five parts. First, we examine the initialization steps that are required to enable the prototype, for the databases, middleware, USR, and client. Second, we detail the capabilities of the Security Policy Client. Third, we explore the functions of the Security Authorization Client. Fourth, we examine the Security Delegation Client, and all of the associated interactions that are needed in support of the delegation process. Last, we present the client applications and associated resources that have been developed in support of the prototype. In all, we make extensive use of bit-maps from the prototype to illustrate the requisite capabilities.

6.2.1 Security Prototype Initialization

We start with a quick review of the behind-the-scene implementation work needed to support the security prototype and enforcement framework. Figure 26 depicts the necessary services as explained earlier in this chapter. Figure 29 depicts the database scheme used as the repository for all of the user and resource data required to make the security assurance checks at design and runtime. This prototype uses the Oracle Database Management System, but can support any

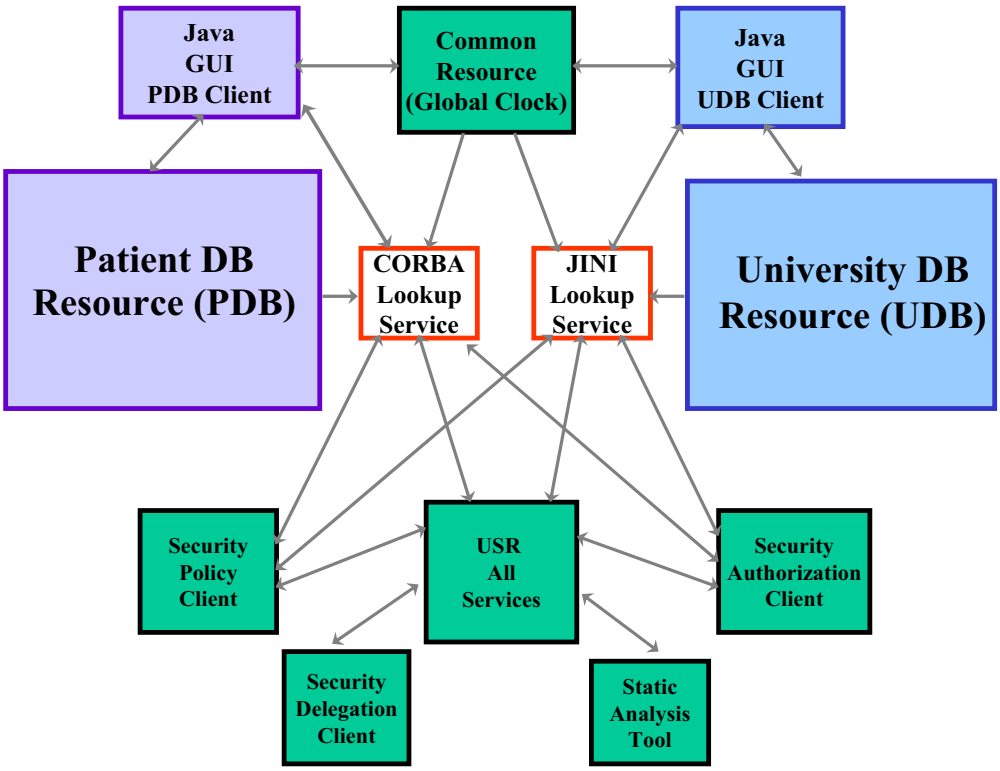


Figure 28: Prototype Security and Resource Architecture.

DBMS supported by Java. The important point here is that the database elements in Figure 29 are directly related to the method parameters.

As most distributed applications, the security prototype observes the client/server paradigm. This paradigm requires the servers to be running before a client application can operate. The next two figures, Figure 30 and Figure 31, depict the initialization of the various servers. We start with the Global Clock Server, Figure 30. The Global Clock Server, as mentioned throughout this work, is a key element to maintaining security assurance. All elements of this prototype security system consult the Global Clock for current time. We make the assumption that the delta-time required to garnish a response from the Global Clock is not significant. What is critical is that there is only one clock to determine validity of all temporal constraints. Next comes the Security Service, Figure 30, followed by the Patient and University Database Servers, Figure 31. Not shown is the database server, which, of course, must be running to effectively operate the security system. In this prototype, only the resource servers that will be used, need to be started.

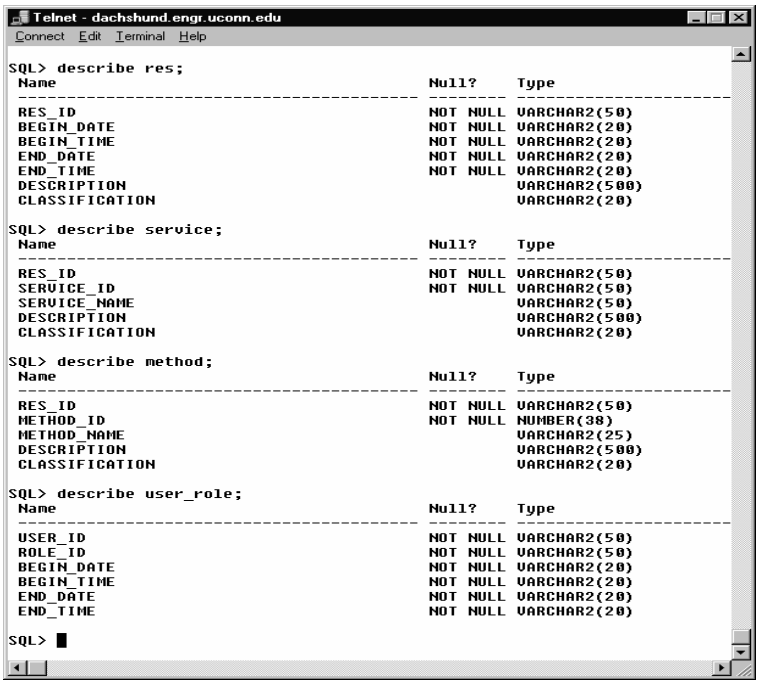


Figure 29: Resource Database Scheme.

The next aspect of this prototype initialization that requires comment is the Lookup Service Architecture, Figure 32. Our initial prototype effort only used Jini. In order to show flexibility and middleware independence, we developed a combined CORBA/Jini solution. CORBA and Jini are two distributed system architectures; CORBA is a well-established distributed system architecture which supports communicating objects specified in a language- independent manner, with possible implementations in a large variety of languages. CORBA supplies an Object Request Broker (ORB) whereby these objects can locate and communicate with each other. The requirements of language independence place certain limitations on the mechanisms that can be used in implementation. Jini is a more recent distributed system technology, based on Java. Jini services must be written in pure Java, and this allows the Jini architecture to exploit certain features of Java. In particular, Jini makes heavy use of mobile Java code, moving "live" objects from one Java Virtual machine (JVM) on a server, to other JVM's on clients. In order for services in one system to be used by another, there must be a bridge between the two. The bridge, depicted in Figure 32, encapsulates its inner details through two classes: Register and Lookup. Register provides method register (int whichServer, Object o, int corbaOrJini,) for servers to register services with either CORBA, Jini or both. The servers do not need to know how to communicate with CORBA and Jini. They only need to provide which server they want to register with, the lookup services, the server object to be registered,

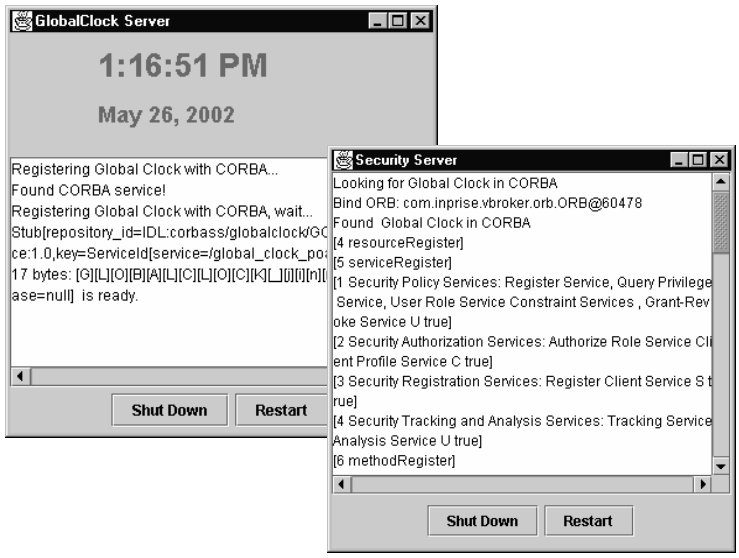


Figure 30: Start Global Clock and Security Servers.

and which lookup services they want to register with as parameters and the call Register.register() method; Lookup provides method lookup (int whichServer) for clients to look for services; the clients do not need to know which lookup services that servers register with, CORBA or Jini. They only need to provide which server they want to find as the parameter and call the Lookup.lookup() method.

6.2.2 The Security Policy Client

The Security Policy Client (SPC) in Figure 28 can be used to define and remove user roles, and to grant and revoke privileges (i.e., time constraints, resources, services, methods, and/or signature constraints). The SPC is a set of tabbed panels for each major function of the GUI, as shown in the following figures, Figure 33 to Figure 42. These figures detail how the security model and enforcement framework is realized.

The first figure (Figure 33) is simply the Security Policy Client (SPC) Login. The login is required by most distributed systems for authentication purposes, so the system can verify you are who you say you are based on the userid and password. The SPC login also provides a measure of authorization by allowing a specific userid to only pick from authorized roles in a pull-down menu. This authorization is accomplished using stored data already entered into the Unified Security Resource (USR). Allowing a particular userid to only see authorized roles is a measure of security assurance because the user only sees what is necessary and nothing more. Allowing any user to see all the system roles is providing unnecessary information and is a violation of the "need to know" concept. In this example, Figure 33, even the security admin has constraints.

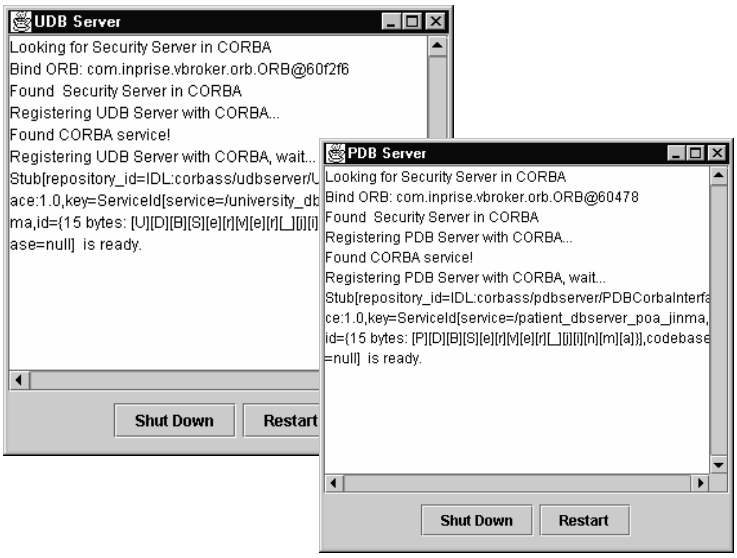


Figure 31: Start Patient and University Database Servers.

Figure 34 depicts the Security Policy Client (SPC) registration of two resources; the Patient Database Resource (PDB) and the University Database Resourse (UDB). The SPC registration GUI supplies a listing of available resources, of course, but also provides a series of pull-downs to set the lifetime of the resource. This corresponds directly the definition of a resource (Definition 6) and the definition of a lifetime (Definition 1). Note: There is the opportunity to add both a start time, st, and an end time, et. Both of which will be checked against the current time, ct, in a design time check that will not allow and expired resource to be registered. The security system automatically provides a default resource registration of one year from the ct. An additional note, is the fact a classification is associated with each resource. In order to leverage mandatory access control, MAC, (Definition 13) each resource is assigned a classification. The user is responsible for assigning the classification and in the absence of a classification, an "Unclassified", U, will be the security system default.

The next GUI, Figure 35, depicts the addition of a service (Definition 5) to a resource. In this case, the three available services; Query, Update, and Membership, are services available to the PDB resource. Notice the PDB resource is highlighted in the resource pull-down menu. For security assurance purposes, the service classification must dominate the resource classification. Note: This version of the security system prototype does not force the entry of a lifetime like that accomplished with the resource. Using the lifetime with the resource is adequate for the system proof of concept.

After the service GUI is, Figure 36, which depicts the addition of methods (Definition 4) and subsequent confirmation of the methods to a resource. In this case, there are several available methods available to the PDB resource. Again, notice the PDB resource is the chosen resource from the pull-down menu. For

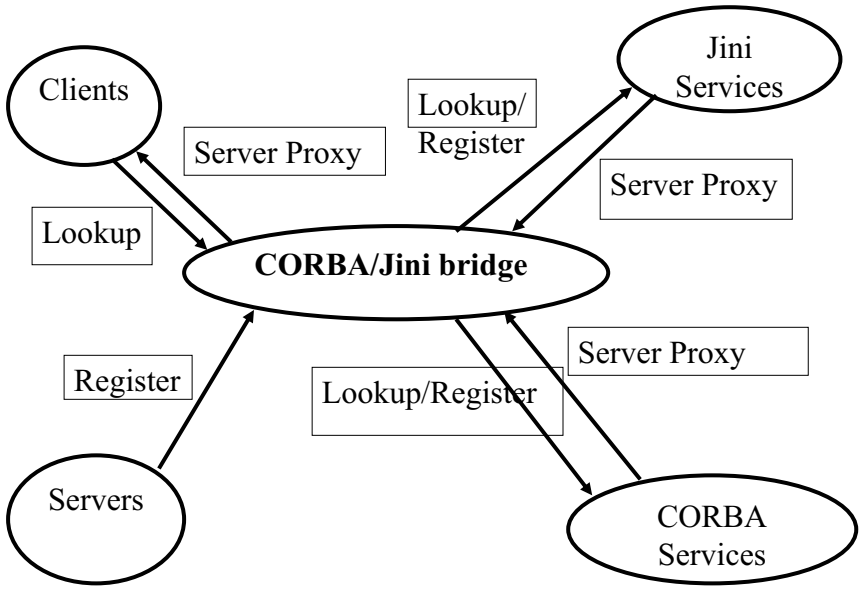


Figure 32: Lookup Service Architecture.

security assurance purposes, the service classification must dominate the resource classification. This holds true for adding methods to services as well. The idea of dominance is defined in Chapter 3 along with what drives the overall resource classification. Note: This version of the security system prototype does not force the entry of a lifetime for each service or method like that accomplished with the resource. Using the lifetime with the resource is adequate for the system proof of concept with rest to lifetimes.

Once the services and methods are registered to a resource, the methods can be logically separated into the appropriate services. Figure 37 shows the successful addition of a method to a service. Note that the method classification is assigned by the resource owner and must dominate the service. In Figure 37, the "getDiagnosis" method carries the classification of (S), secret, but the service classification is (U) or Unclassified.

Figure 38 depicts the query function of the SPC. The query function allows an SPC user to examine what services and methods are available to a specific resource. The GUI is a series of pull-down menus that allow the SPC user to customize a simple query. The query information is found in the database supported by the USR. The database scheme is depicted in Figure 29.

In addition to assigning the correct functionality to a resource by assigning the proper services and methods, the SPC is also used to build user roles (Definition 7). It is in the user role, that the security policy for a distributed system is

Figure 33: Login: Security Policy Client .

realized. In building roles, the policy maker is dictating what a user can and cannot do while executing their role. The principle of "least privilege" is observed when building roles to supports security assurance. The GUI in Figure 39 shows how the SPC is used to create a user role. The SPC is also used to set the classification level of a role (default (U)). For security assurance reasons, a user clearance must dominate the user role classification. In Figure 39, the role "nurse" is created and assigned a classification of Secret, (S). The lifetime of a user roles is assigned to the user role when assigned to a resource, Figure 40.

After creating the user role, it is critical to security assurance that no resource, service, or method be assigned to a role in violation of MACC (Definition 13). This is a design-time check executed by the enforcement framework. Also, as depicted in Figure 40 to Figure 42, a lifetime is assigned to each resource, service, and method, when added to a role. This lifetime assignment is used for assurance checks at design time, when a user is assigned a role and at runtime, when a user tries to invoke a method.

Two of the significant contributions of our security model and enforcement framework are depicted in Figure 43; the Signature Constraint, Definition 11, and the Time Constraint, Definition 12. Both are realized using the SPC while assigning a method to a user role. The Time Constraint is like a lifetime and is assigned in the same way; a start time and an end time are registered using the menu pull-down menus. The Time constraint is always and additional constraint on the availability of a method. A Time Constraint cannot increase method availability beyond that of the existing time constraints. The Signature Constraint is much different than any other constraint discussed thus far. The Signature Constraint uses the method parameters, of the method being assigned, to create Boolean Expressions which must be satisfied at runtime. In our example, Figure 43, the "dept head" role is assigned the "updateCourseCapacity" method with a constraint on course CSE372 of an enrollment of 30 students. This means the maximum course size can be 30 students. This constraint reflects a policy

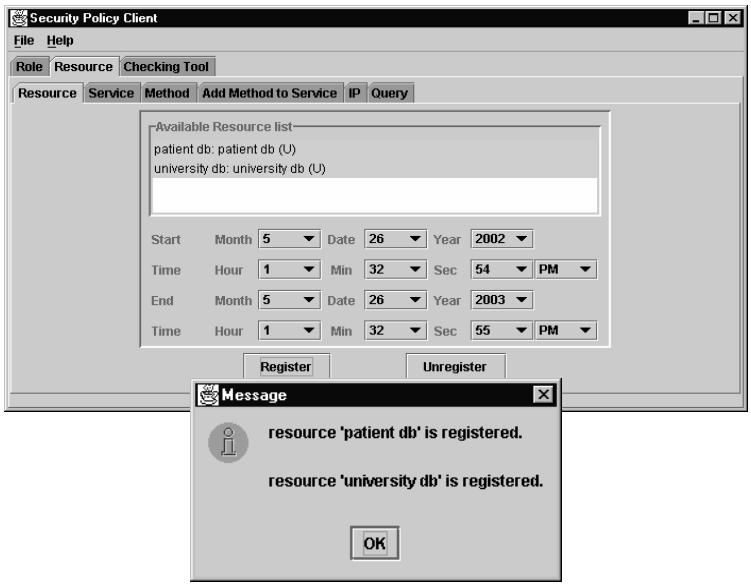


Figure 34: SPC: Register Resources.

that no course should have more than 30 students and even the department head cannot violate this constraint. One method can have many signature constraints, because there can be constraints assigned on each parameter of the method and there can be multiple constraints set on each parameter to form limits. For this reason we make an assumption that there exists a "Signature Constraint Oracle," Table 4, Assumption: XXXIV, that will always return the correct Boolean expression resolution. Constraints on different method parameter are "AND"ed together and constraints on the same parameter are "OR"ed together. Once the the "Add Signature Constraints" option is selected, the SPC will automatically cycle through each parameter and prompt the user for a constraint.

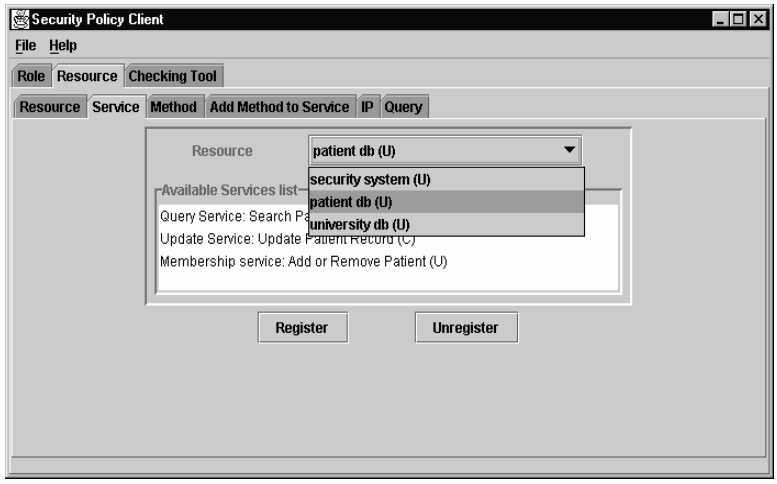


Figure 35: SPC: Add Services to Resource.

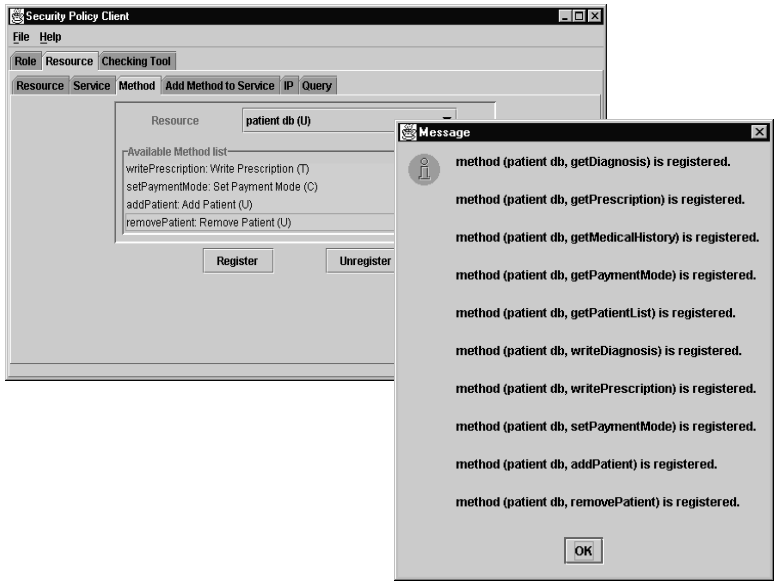


Figure 36: SPC: Add and Confirm Methods to Resource.

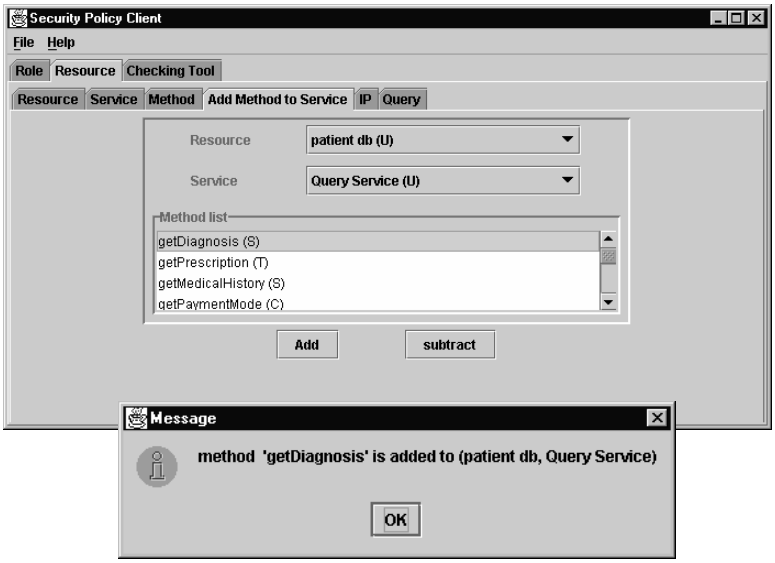


Figure 37: SPC: Add Method to Service.

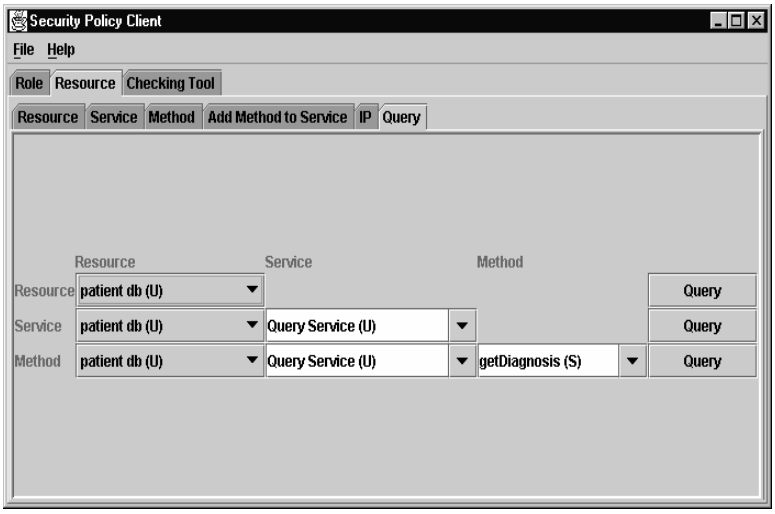


Figure 38: SPC: Resource Query.

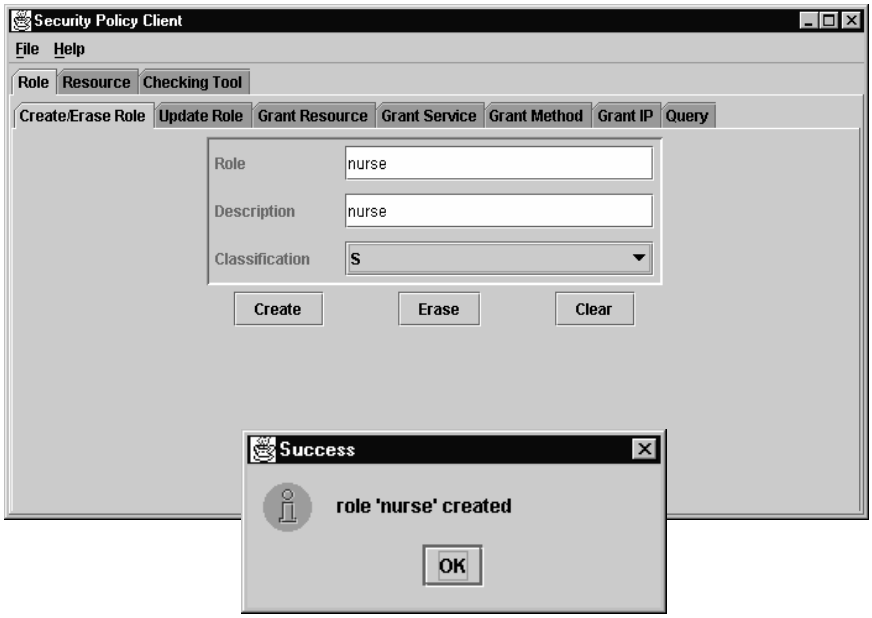


Figure 39: SPC: Create a User Role.

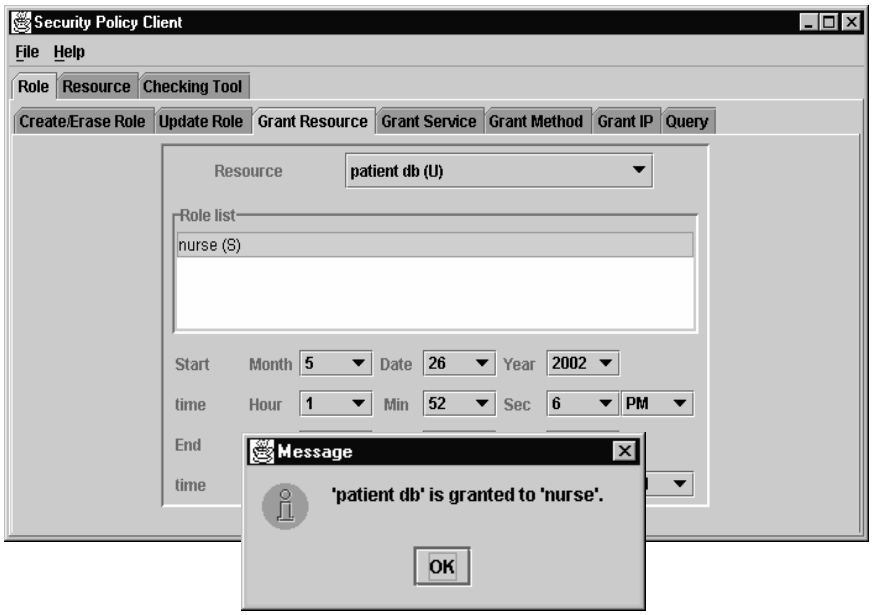


Figure 40: SPC: Grant Resource to User Role.

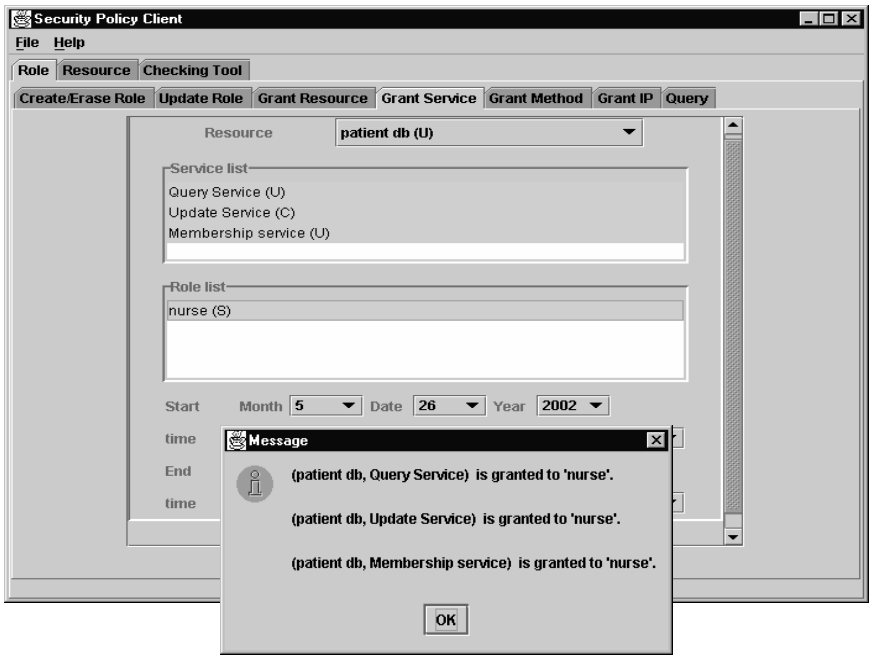
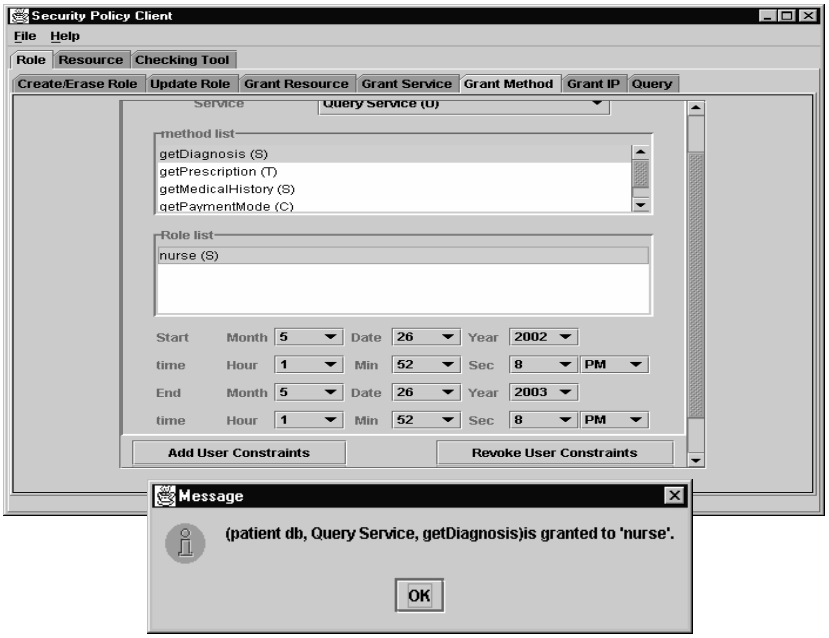


Figure 41: SPC: Grant Service to a User Role.



This GUI uses Rule I to ensure no CLS or LT policy violations.

Figure 42: SPC: Grant Methods to a User Role.

Figure 43: SPC: Create Signature Constraint on Method to User Role.

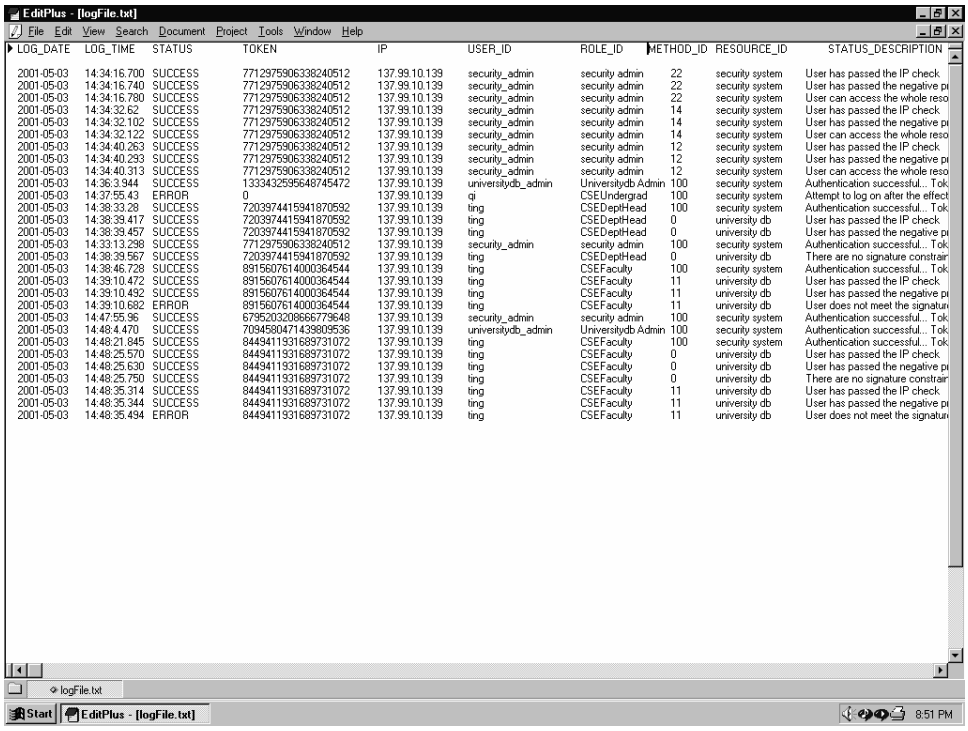


Figure 44: USR: Tracking Service.

In addition to creating and building roles to support the security policy, the security officer can also inspect and monitor security via the tracking capabilities of SPC. A typical security tracking history is shown in Figure 44. By logging invocations the security officer can track status of the attempted access (Success/Error) of the method (method-id) of a resource (resource-id) by the particular user (user-id) playing a specific role (role-id). With this functionality, non-repudiation is achieved, which means a user cannot deny an action and can be held accountable. There is no anonymous activity in this security model prototype, even by the security policy folks. In the event of intended or unintended database corruption, tracking activity also plays a key role in recovery. Database administrators can recover from a point closest to the point of corruption, thus minimizing the impact of an attack. Both non-repudiation and recovery are essential components of good security assurance; they help maintain system integrity and deter malicious user activity.

The Static Analysis Tool (SAT), is utilized by the security officer to analyze source code of Java resources, allowing us to track not only the method on a resource that has been directly authorized to a role, but also the other resources (and services/methods) that are called. Given the directory location of a Java source code, SAT analyzes a class by inspecting all of the method definitions to find any other method called inside the one under inspection. The result, in Figure 45, tracks the information on the method being analyzed, the methods that are invoked by the method, and the user roles assigned to the method.

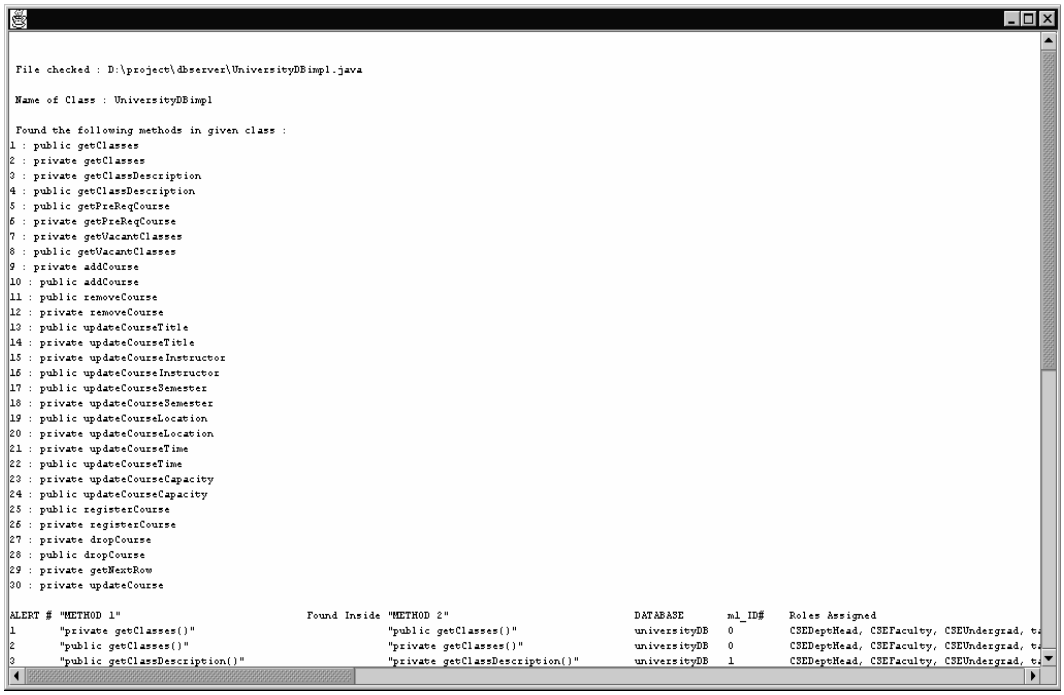


Figure 45: USR: Security Analysis Service.

Methods that are in violation of policy are noted and the security officer can make a decision to include the method in violation into a role (making it valid) or change the role. Role-based system would be complete without the ability to query a role. Figure 46 displays the results of a role query on the nurse role. The display lists the Role, Classification, Resources, Services, and Methods to which the role is assigned, along with the authorized users.

6.2.3 The Security Authorization Client

The Security Authorization Client (SAC) supports authorization of role(s) to users. A user may hold more than one role, but can only act in one role at a time. Playing multiple roles at the same time may lead to conflicts in the event different SCs and/or TCs are defined on the same resource, service, or method. If more broad access is desired, a role with expanded capabilities can be defined to encompass privileges of multiple roles without SC/TC conflicts. The SAC is also a set of tabbed panels, shown in the following figures (Figure 48 to Figure 50). The SAC has the capability to create a new User, which has an identifier, password, a time constraint (begin/end date) that indicates the valid time interval for the user, and a text description. The SAC is separate from the SPC for security assurance reasons. Policy, set by the SPC is normally at

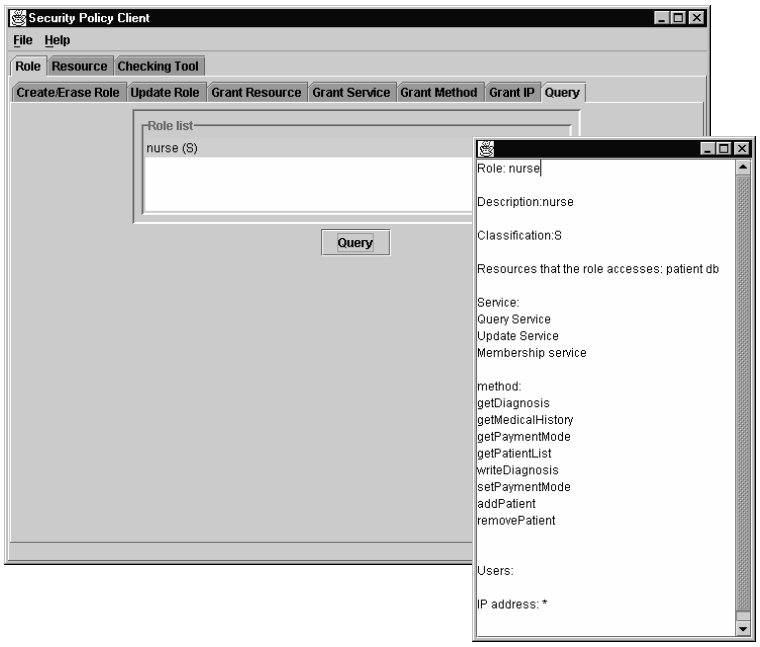


Figure 46: SPC: Query a Role.

a higher authority level than authorizations, so the individual setting the policy should not be the person making authorizations. This could also lead to a conflict of interest, because policy can constrain authorizations and one individual, with both roles, could customize their own authorizations. Separating the SPC and SAC is also in concert with the security assurance concept of least privilege.

An authorized user needs to login to the Security Authorization Client just like the Security Policy Client, (Figure 47. The first SAC figure, Figure 48, describes the creation of a user. In this case the user is assigned a User Id (kim), a clearance (S-Secret), and a default lifetime of one year (5 May 2002 to 5 May 2003). These entries are derived from the definition of a User (Definition 9). The next significant use of the SAC is to grant privileges. This is accomplished in Figure 49 where User, kim, is granted the User Role, nurse. To stay consistent with our security model, Chapter 3, a security check is accomplished to ensure User, kim, has the clearance to dominate the classification of the User Role, nurse. Notice that a lifetime is also given to the user role upon assignment to a user. After assignment, if necessary, the user-role lifetime will be automatically adjusted so the user-role lifetime dominates. The user will not be allowed to access a user role if either the user-role lifetime or the user lifetime has expired. This is consistent with the security model and supports security assurance. The final SAC figure, Figure 50, depicts a user query. Like the SPC, the SAC uses the Unified Security Resource Database to query information. In this case, the User ID, kim, is the subject of the query. The User Id query displays: User

Id, lifetime, clearance, and role, which are necessary for making access control determinations.

Please note that there are other SAC tabs not discussed in this section because they are the subject of previous and future work to be discussed later.

Figure 47: SAC: Login Authentication.

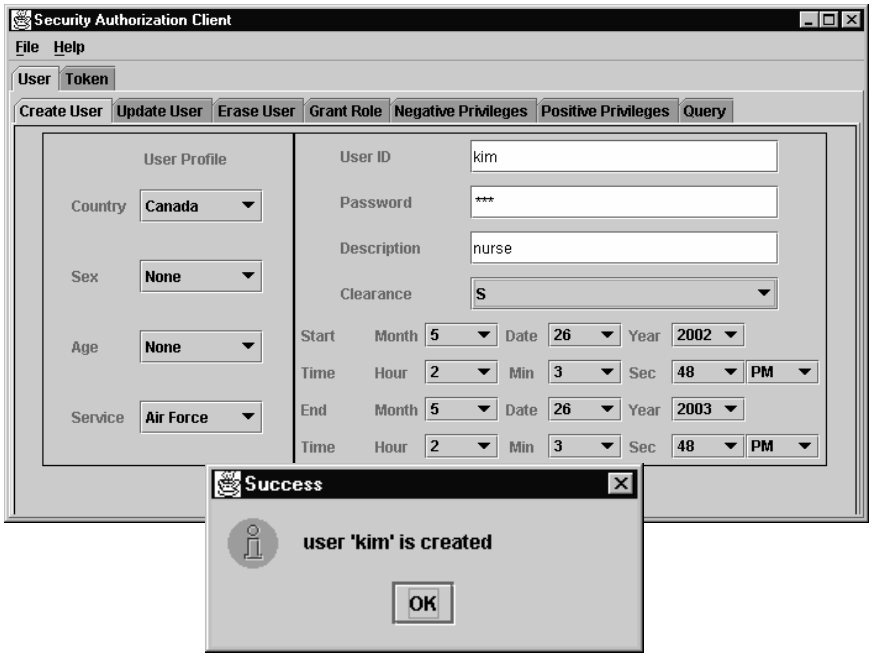
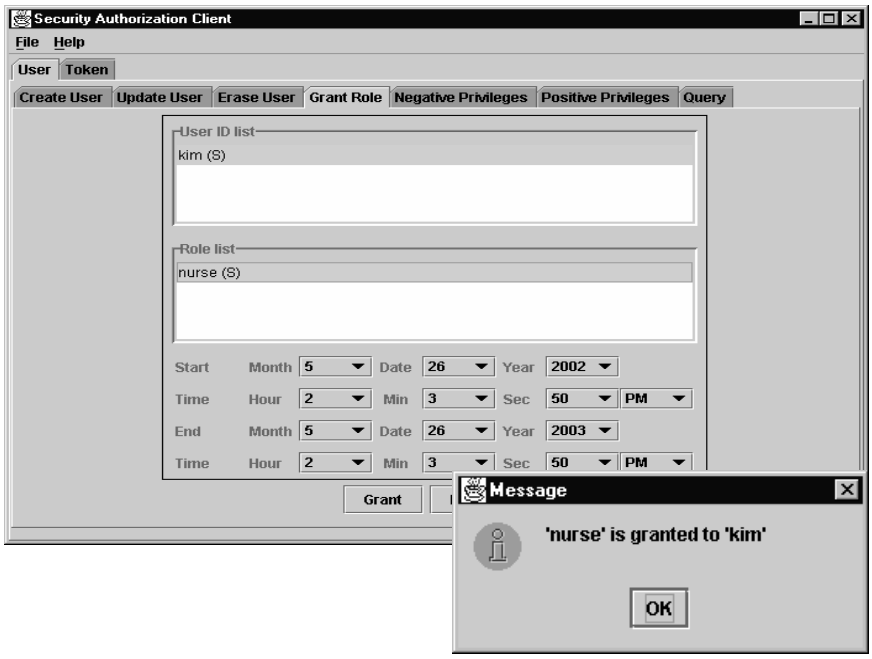


Figure 48: SAC: Create a User.



This GUI uses Rule II to ensure no MAC, TC, or LT violations;
Rule III to ensure valid user authorization;
and Rule IV at runtime to validate constraints.

Figure 49: SAC: Grant Role to User.

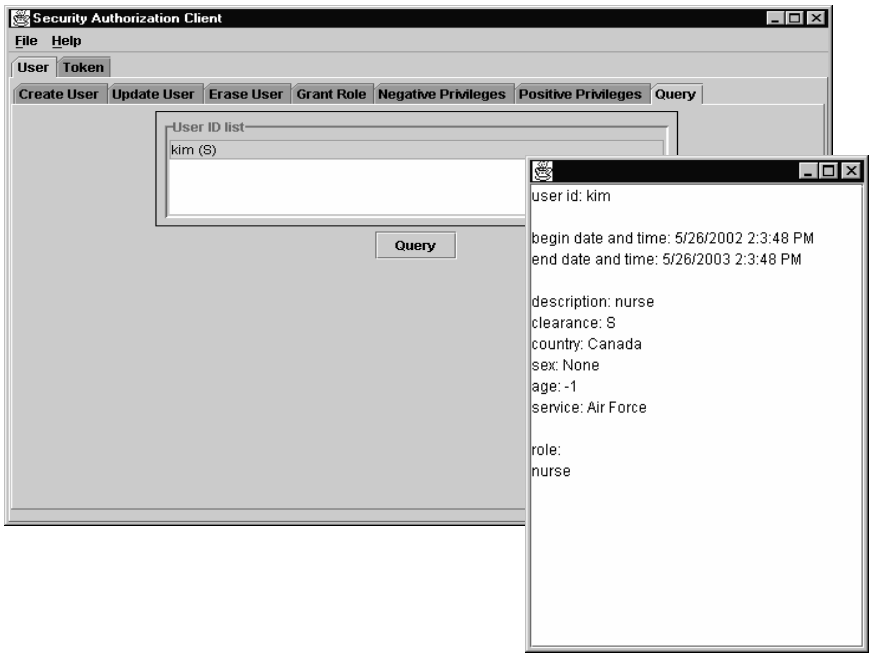


Figure 50: SAC: User Query.

6.2.4 The Security Delegation Client

For consistency in authorization, there must be the ability to define, examine, and control role consistency and delegation authority. Recall that Figure 25 depicts, 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; and the Security Analysis Tool (SAT) to dynamically track all client activity, including logins and method invocations. The SPC, SAC, and SAT all have different responsibilities in defining and assuring role delegation. In fact; role delegation forced a change in both the Security Policy Client and Security Authorization Client, as you will see. In this section, we show how the changes in the SPC and SAC along with Security Delegation Client (SDC) are used to grant, update, and revoke delegations. This is accomplished through a series of figures (Figure 51 to Figure 57), which depicts the different aspects of role delegation.

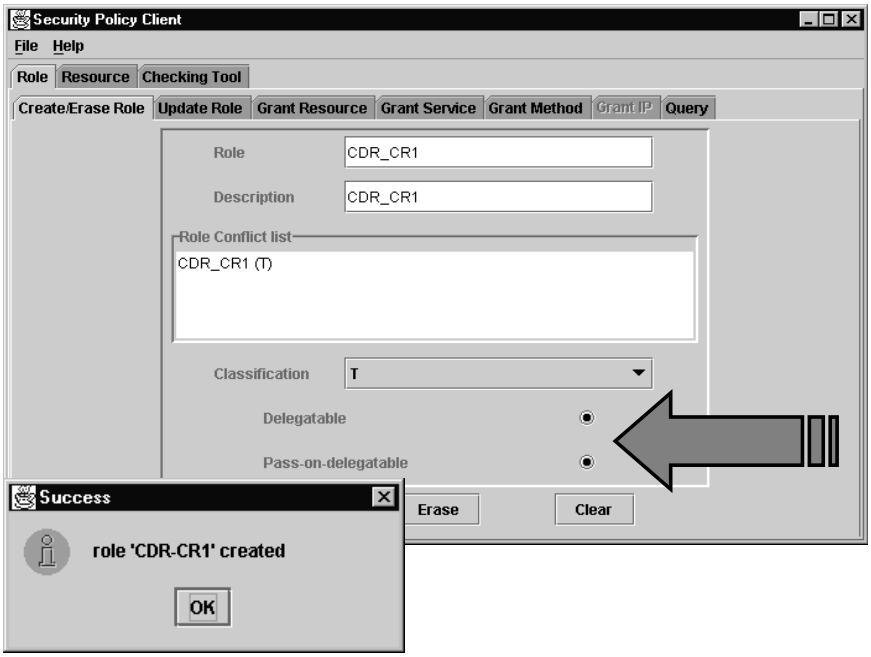
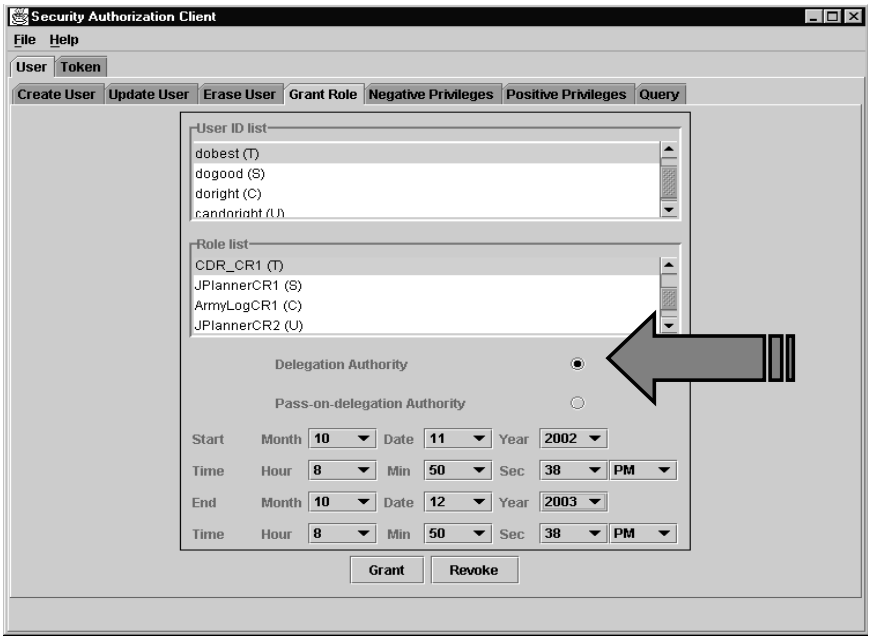


Figure 51: SPC: Create Delegatable Role.

To begin, a user role must be created and designated as delegatable (Definition 19). This is a policy issue and is handled with the policy client, SPC. Figure 51 shows how SPC is used to establish whether a role is delegatable (DUR); note the dark arrow pointing to the radio buttons, indicating delegatable and pass-on delegatable. In our example, CDR-CR1 is a delegatable user role and CDR-CR1 is also a pass-on delegatable user role (Definition 25). This means, the user role can be passed from one delegated user to another, if the original user is authorized and grants that permission at delegation time.



This GUI uses Rules I, II, and III to ensure
no MAC, TC, LT, SC or VUA violations.
This GUI also uses Rules V and VI for assignment of DA and PODA.

Figure 52: SAC: Grant Role with Delegation Authority.

Next, a user must be authorized to delegate, Definition 22. This authorization is accomplished with the authorization client, SAC. In Figure 52, SAC is used to grant delegation authority to the user. Again, note the dark arrow pointing to the radio button. Figure 52, represents a key step in role delegation. A User can be granted Delegation Authority, and a user role can be made delegatable, but delegation still cannot happen until the delegatable user role is granted to a user with delegation authority. Notice also, the SAC is not required to grant delegation or pass-on delegation authority (PODA). In this case, a delegatable user role can be delegated to another authorized user, but the delegation can go no further because there is no pass-on delegation authority.

From the user's point of view, once given authority by the security administrator to delegate a delegatable user role, there must be tools for the user to manage delegation, tools which are also needed by the security administrator. Delegation is accomplished via the Security Delegation Client, SDC. The Security Delegation Client (SDC) is depicted in four figures, Figure 54 to Figure 57, and has three primary delegation functions: Grant, Update, and Revoke.

In order to invoke the SDC, (Figure 53) one must be an authorized user (Definition 17a) of a delegatable role. If this is the case, an OU (Definition 21) will be able to invoke the SDC. The Grant tab GUI, Figure 55, allows the user to:

1. Choose a DU (Definition 22), with the "Delegate to:" pull-down menu. The SDC will only populate the pull-down menu with eligible delegatees. A DU must meet the same MACC requirements as the delegating user and the security assurance rules developed in Chapter 6.

2. The delegatable role to delegate from the "Delegatable Role List", again, only delegatable roles will populate this list according to the DU.

3. Set the delegation lifetime, which will be constrained by the lifetimes of the OU and DU.

4. And finally, the Grant Tab Gui is where the delegating user grants delegation and pass-on delegation authority, PODA (Definition 25). Notice the dark arrow pointing to the checkbox on Figure 55.

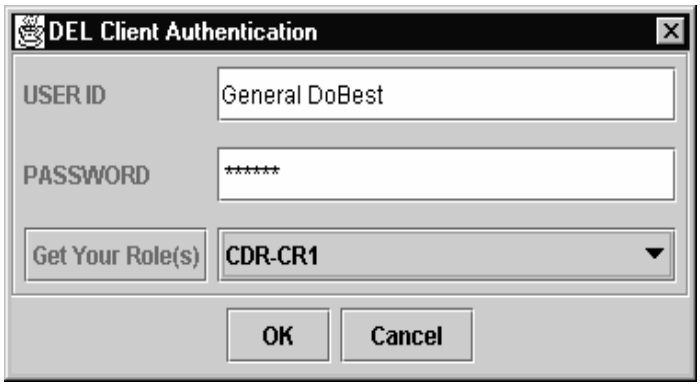
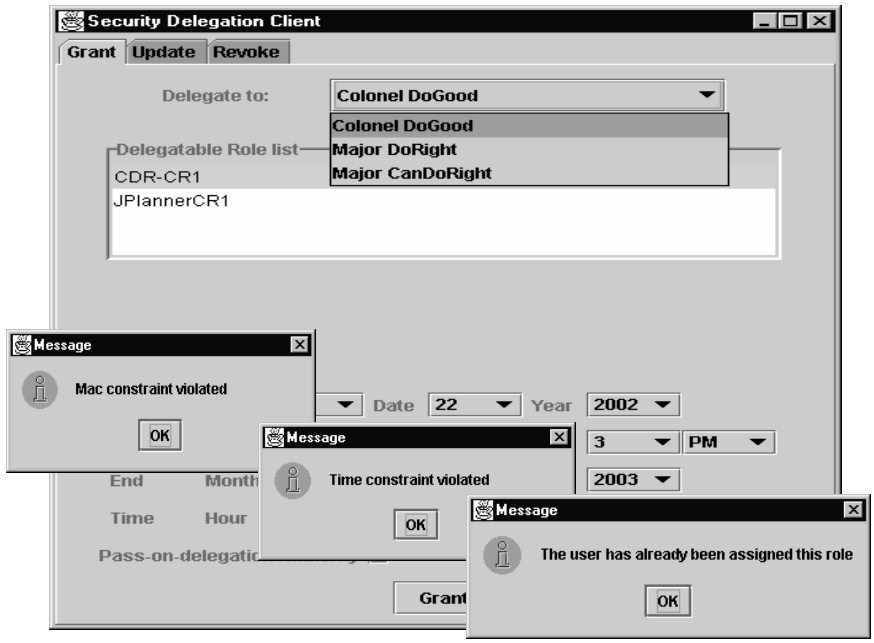


Figure 53: Login: Security Delegation Client.

The Security Delegation Client (SDC) is responsible for interfacing with the other elements of the Unified Security Resource (USR) for security assurance. Figure 54 depicts the typical errors one might encounter while trying to delegate a role. The typical delegation errors are: MACC, Time Constraint, and already assigned. The SDC initiates all of the security assurance checks required for assignment of the role. These checks help maintain consistency with the security model, enforcement framework, and the prototype. The delegation client requires an additional check to make sure the delegated user is not already an authorized user. In that case, there is no need for delegation.



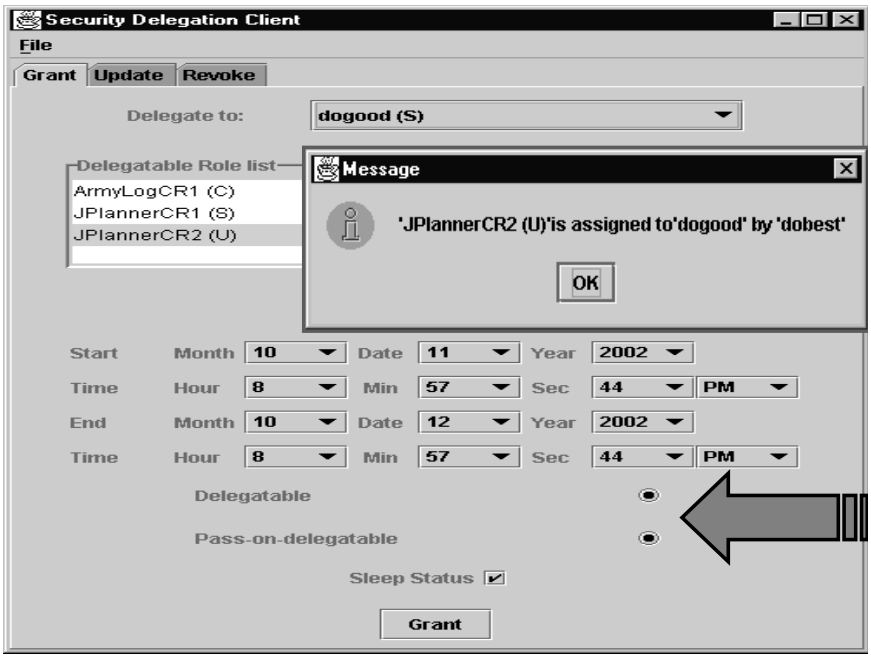
Rules I - IV and VII in action.

Figure 54: SDC: Delegation Errors.

The Update tab allows a user to modify any existing delegation Figure 56, while the Revoke tab enables the user to cancel any delegation Figure 57. It is important to note here, when modifying a delegation, the same security assurance rules (Chapter 5) are check as when making the original delegation, and when a delegation is revoked, the revocation must follow the revocation rules established in Chapter 4. The delegation of a user role by a user is similar to the security officer assigning user roles. This process works in conjunction with the other elements of the Unified Security Resource (see Figure 25), and utilizes the same underlying databases [96]. With the SDC, an organization has the flexibility to give certain users DA and PODA, while still maintaining administrative control and security assurance.

Finally, the design-time assurance checks as given earlier in Chapter 5, are summarized below with respect to the tool (SAC or SDC) that is utilized to perform the check. Again, this shows consistency between our security model, enforcement framework, our prototype.

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 Delegation Consistency: This check is performed by SDC.

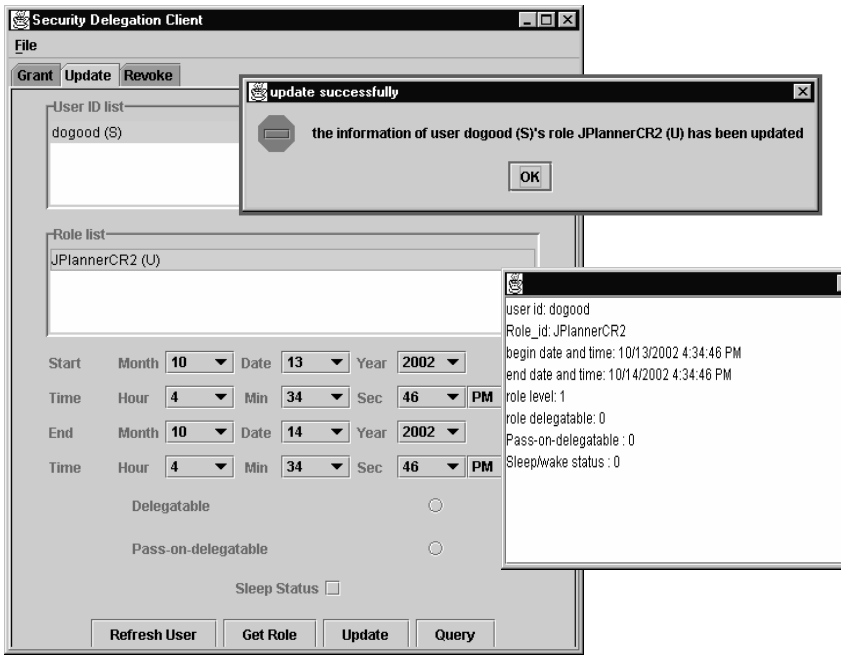


This GUI uses Rule VII for the Delegation.
and this GUI uses Rules V and VI for DA and PODA.

Figure 55: SDC: Successful Delegation.

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 25) 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.



This GUI uses all Rules to validate updated delegation.

Figure 56: SDC: Update GUI.

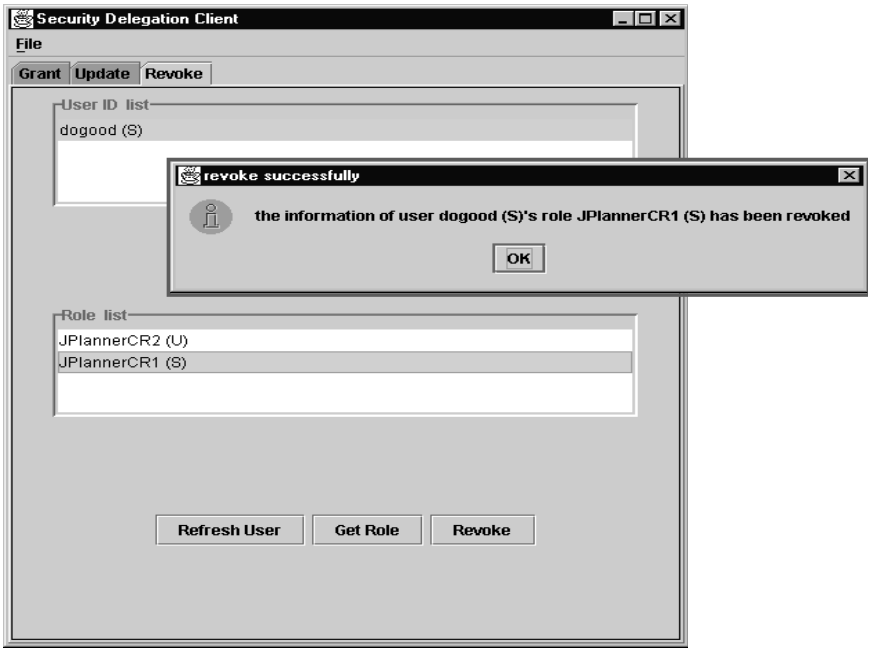


Figure 57: SDC: Revoke Delegation GUI.

6.2.5 Client Applications and Database Resources

The Patient Database resource (PDB) and the University Database resource (UDB) are simple resources built to test our security prototype in a distributed environment. Figure 25 illustrates the distributed environment architecture. We have successfully implemented a Unified Security Resource that can administer more than one security policy to different resources in the same distributed environment. This is an important goal because it shows the potential of this security resource to be used in support of the Dynamic Coalition Problem discussed in Chapter 2 and other distributed environments. This section provides the reader with a set of graphical user interfaces for the PDB. The main purpose of this section is to illustrate how the a security policy can be enforced on a specific resource, and not necessarily how the resource works. We could not simulate the use of the Global Command and Control System (GCCS) used for examples throughout Chapters 2, 3 and 4, but have provided examples of what typical GUIs would look like, if implemented, in Appendix A.

All clients start with a login GUI for authorization and authentication, Figure 58. Upon login, the user must select a role. The PDB Client Authentication GUI will only provide a list of roles authorized to that user, in this case, "jin." It is the security resource that provides that information to the login client. It was the SAC that created the user, jin, and granted the roles, accountant and doctor.

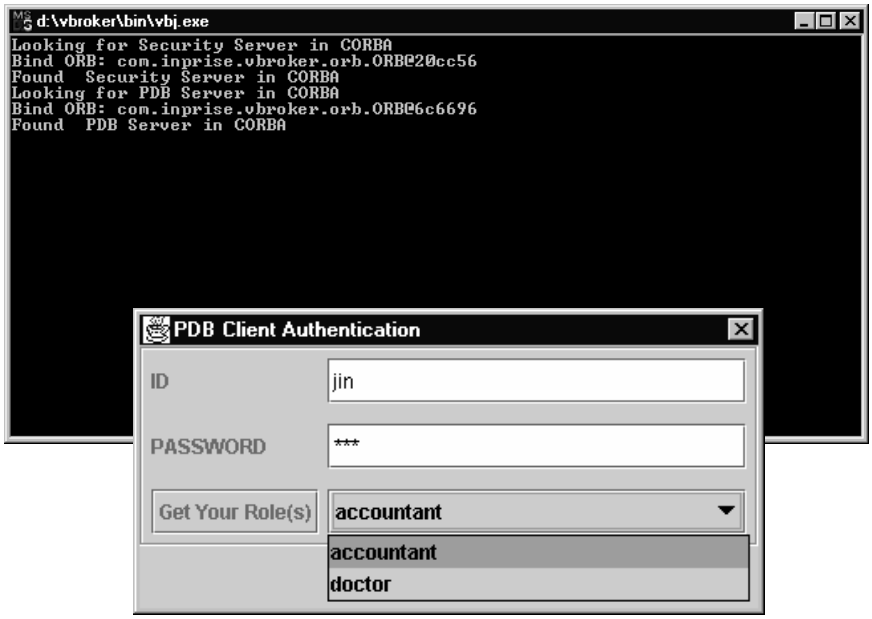


Figure 58: PDB: Client Authentication.

Like the Security Policy Client (SPC) and the Security Authorization Client (SAC) the PDB uses a series of tabbed GUIs. The first PDB GUI, Figure 59 shows the AddRemove Tab. This tab has two sub-tabs that allow the user to add and remove patients. If the user was not authorized to use this function, it would be "grayed out" indicating it is not available. The next tab is the Update Tab, GUI, Figure 60. This tab has three sub-tabs shown in figures 61, 62, and 63. Figure 60 shows the Diagnosis Tab, which is used for updating a patient's diagnosis; Figure 61, shows the Prescription Tab, and Figure 62, shows the Payment Mode. If any function was not available to the user, it would be "grayed out." The final five figures, Figure 63 to Figure 67 depict the five sub-tabs of the Query Tab: Medical History, Diagnosis, Prescription, Payment, and Patient List.

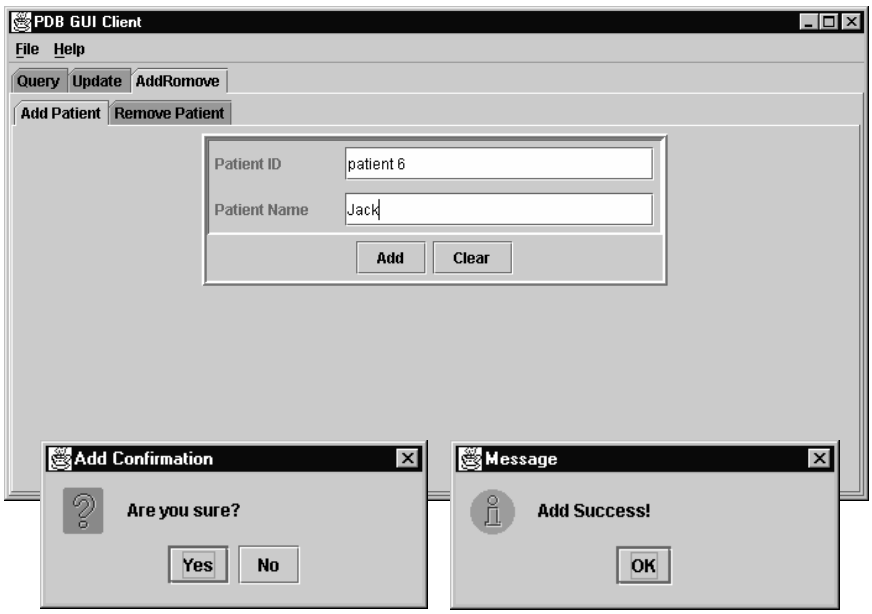


Figure 59: PDB: Add Patient.

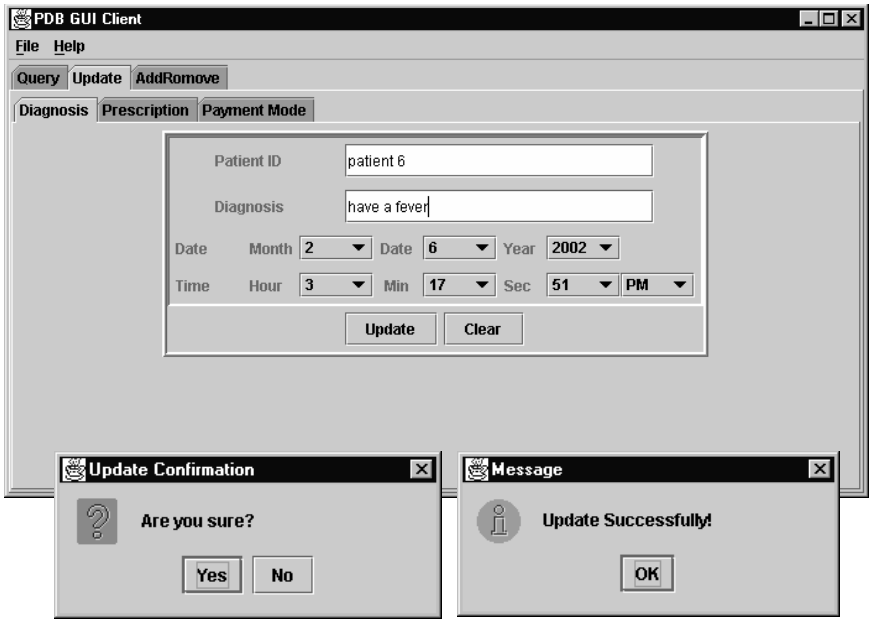


Figure 60: PDB: Update Diagnosis.

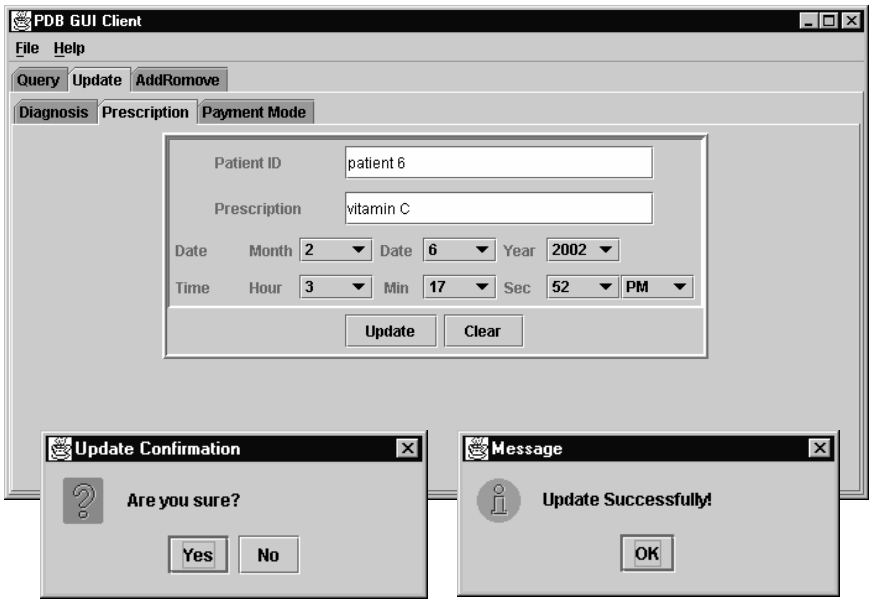


Figure 61: PDB: Update Perscription.

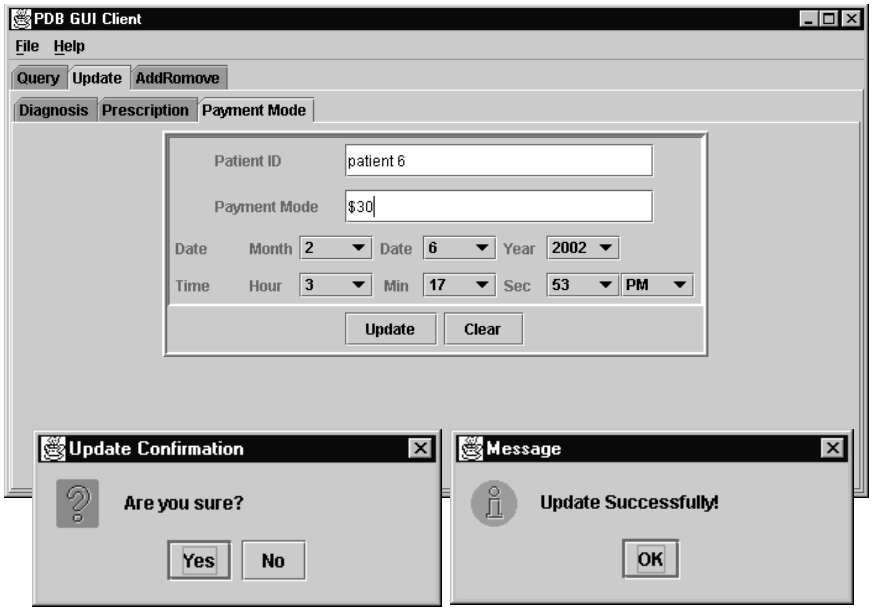


Figure 62: PDB: Update Payment Mode.

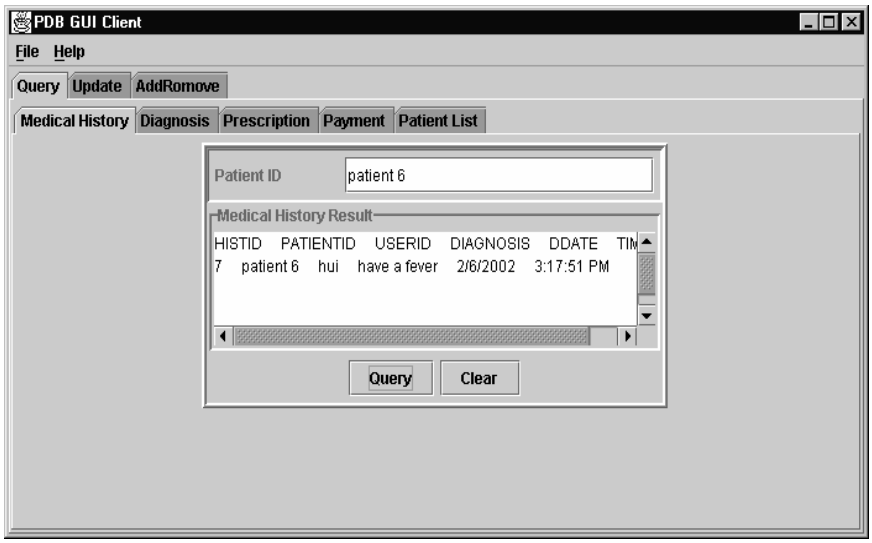


Figure 63: PDB: Query Patient History.

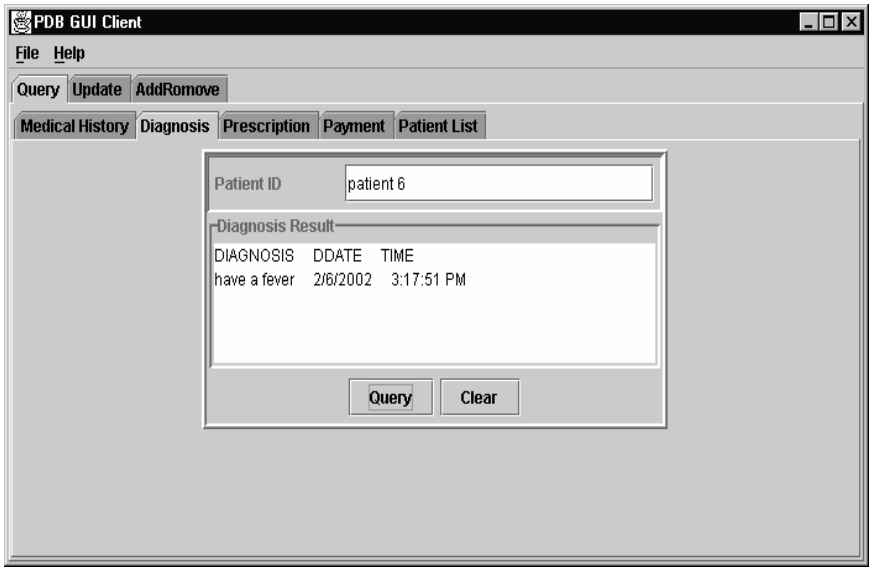


Figure 64: PDB: Query Patient Diagnosis.

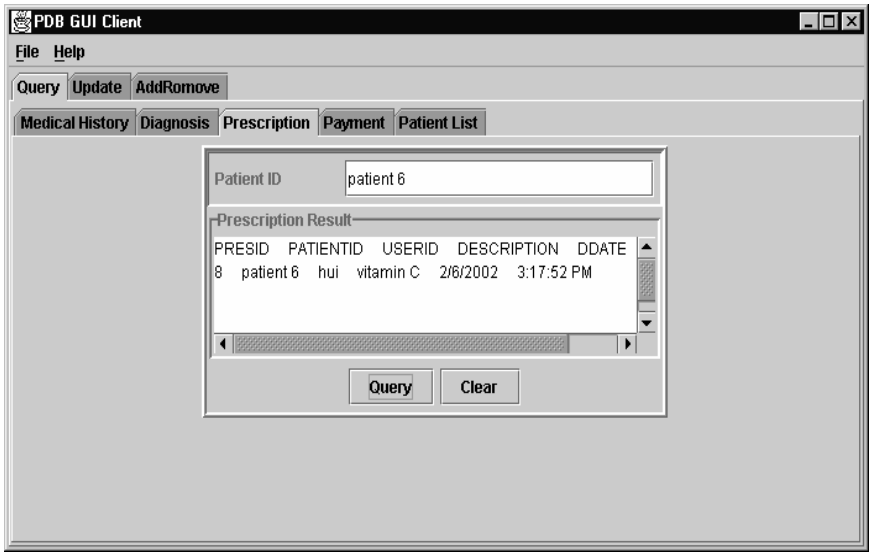


Figure 65: PDB: Query Patient Prescription.

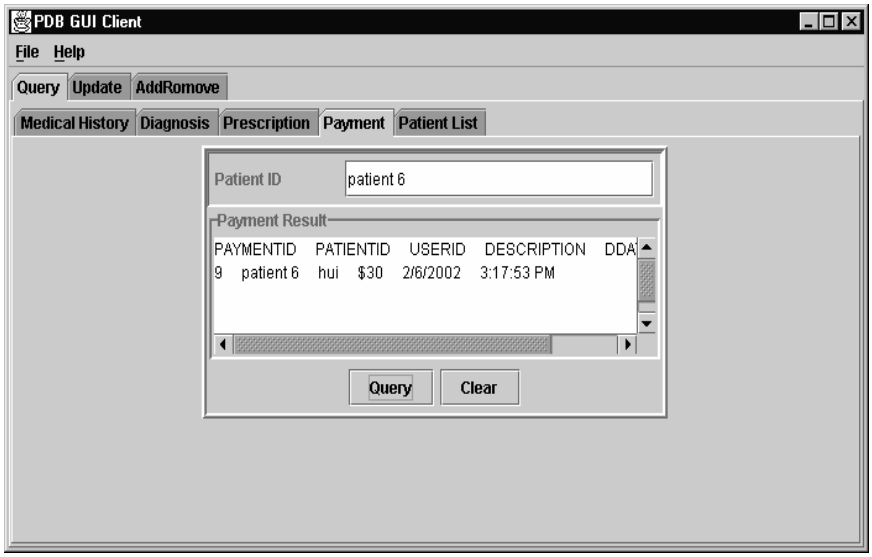


Figure 66: PDB: Query Patient Payment.

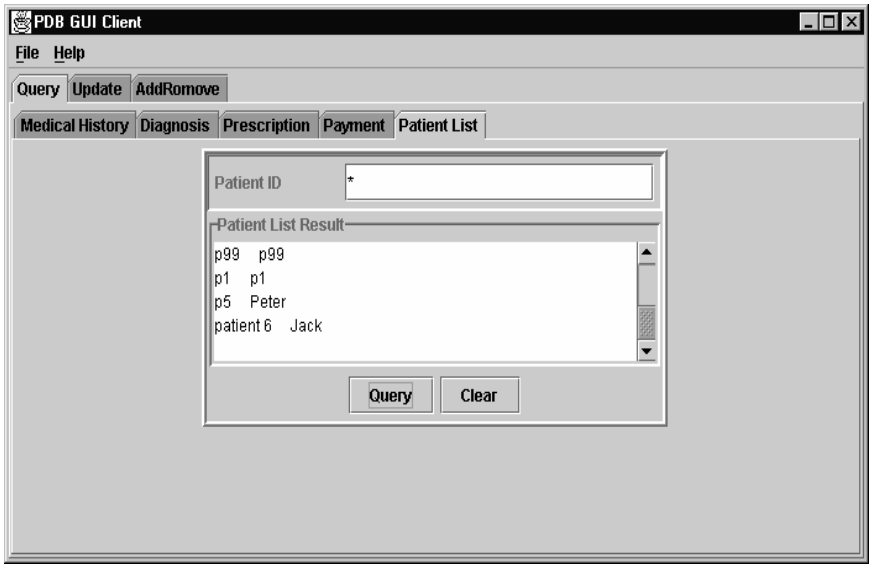


Figure 67: PDB: Query Patient List.

Chapter 7

Research Contributions and Future Research

This dissertation has examined security assurance guarantees for a resource-based, RBAC/DAC/MAC security model and enforcement framework [67, 96, 97]. Specifically: Chapter 2 reviewed the problem definition to include information sharing and security in a distributed environment and a description of the Dynamic Coalition Problem. The Dynamic Coalition Problem addresses concerns facing military and government agencies in an international information sharing environment. Chapter 3 covered the security model we have developed to support RBAC/DAC/MAC security using lookup service middleware. This model with associated proofs, is one of the major contributions to the security and access control field. Chapter 4 extended the RBAC/DAC/MAC security model with DAC, by offering role delegation as a viable tool for users, giving some discretion, but still maintaining the highest levels of security assurance. Chapter 5 examined our security guarantees with respect to available time (when an invocation can occur - Section 5.1), sensitivity levels (MAC) (authorizations of methods to user roles, and user roles to users - Section 5.2), and for the properties of safety and liveness (related to authorization, invocation, and delegation - Section 5.3). The work in Chapter 5 represents a second major contribution in this research, specifically, the series of lemmas and theorems that demonstrate consistency within the context of our RBAC/DAC/MAC security model, provide validation for the important MAC properties of Simple Security and Simple Integrity, and insuring safety (nothing bad can happen) and liveness (all good can happen). Chapter 6 displayed different aspects of the proof-of-concept prototype which we successfully modelled a distributed resource environment. In this prototype, we show the relevance of our work by providing separate resources with different security policies in the same environment using one security resource and different lookup service middleware solutions. The remainder of the chapter will detail my research contributions and future research effort.

7.1 Research Contributions

Security assurance is critical to the success of information sharing in any environment. Our formal RBAC/DAC/MAC security model and enforcement framework with its security assurance proofs is intended to address this critical aspect of information sharing. Security and access control in distributed environments is difficult to achieve and manage. Our research and proof of concept prototype, shows that it is possible to provide a security framework and infrastructure for incorporating security into a distributed setting with flexibility, portability, and platform independence characteristics.

Our security model is a significant improvement over other models and has several unique features. First, this security model unifies RBAC, DAC and MAC. This has not been done because DAC is considered too flexible and MAC too rigid with respect to enforcing security policy. Second, our usage of time-based constraints for temporally controlled access is unique. By defining time- periods of access, *lifetimes* for objects and subjects, we are able to establish a very fine-grained access control in a time sensitive and dynamic environment which has not been realized before. A third feature of this security model is the use of value-based constraints to govern access to methods based on parameter values. Methods that are part of the APIs are normally available to all in an unconstrained manner. Our approach provides fine-grained access to methods, allowing multiple roles, which utilize the same public method, to have different invocation constraints without changing the original code. Finally, we provide a level of DAC in our approach, by supporting role delegation by the user. In our security model the best of role-based, discretionary, and mandatory access controls are unified for hybrid system with proven security assurance value. This model coupled with the concepts of design time and runtime security assurance checks makes for a useful and powerful security assurance tool in a dynamic distributed environment where policies and players change often.

Another significant contribution is the proof of safety and liveness for our unified RBAC/DAC/MAC model. We prove that an authorized user can execute any authorized task, and only those authorized tasks, without violating the established security policy. This is *safety* (nothing bad can happen) and *liveness* (all good things can happen)in its purest form. This is done thought a series of eight safety and liveness proofs which are based on eight formally defined security assurance rules. The *security assurance rules, SARs*, are derived from the security model. These SARs dictate what is required for a user to become a user and for a user to execute (invoke a method) a portion of a user role for both runtime and design time activities. Our model also focuses on the MAC Simple Security and Simple Integrity Properties. This is very important when dealing with objects that can be manipulated by a combination of read or write methods.

The final contribution of this dissertation is the proof of concept prototype. We have realized our unified RBAC/DAC/MAC security model, based on an abstract middleware model, in support of interacting software artifacts and client applications. We leveraged middleware capabilities for our security model and enforcement mechanism and implemented a prototype security system that supports our RBAC/DAC/MAC security model. We have demonstrated a degree of flexibility, portability, and platform independence in our solution approach, through a prototype that utilizes multiple middleware platforms (i.e., JINI, CORBA), databases (Oracle, Access), and operating systems (Linux, NT, Win), in support of applications in health care, a university setting, and military acquisition/logistics.

The relevance of this work can be applied directly to the U.S. Military. With shrinking budgets and force structure, the U.S. Military has become more reliant on solutions that use the Internet to execute necessary tasks and this requires interaction between legacy software systems, COTS, GOTS, and shared databases. We have demonstrated success in this area including the restrictive MACC requirements imposed on government systems. We have demonstrated potential use of a system that can be used in coalitions (DCP) and the GCCS, where there is the need to dynamically federate users and resources while simultaneously maintaining information assurance. We have addressed the inherent security risks incurred as a result of federating participants in a crisis quickly, yet still needing to share information. Since Coalition warfare is the wave of the future with respect to crisis management worldwide, this security model has potential value. We have also developed a solution that is rapidly deployable, easy to use, platform independent, and allows for dynamic policy configuration. Clearly, this work is important and relevant to this situation.

Specifically, we have met the goals of:

1. Unification of Role-based, Discretionary, and Mandatory Access Controls. The objectives here were to develop MAC, time, and method signature constraints to which work in concert to maintain flexibility, but realize a very strict security policy as needed to support Government Orange Book requirements.

2. Establishing a set of security assurance rules that can be proven and used at both run and design time to provide increased assurance. This was accomplished with the objectives of a security framework for both design and dynamic runtime assurance, along with administrative tools to realize a fine-grained security policy.

3. Accomplishing everything in a distributed setting. Our objectives here were to develop a middleware lookup service solution that is flexible, portable, and platform independent, which provides a security solution to legacy systems with minimal programming impact. Our middleware solution seems

viable and the working prototype lends creditability to the idea that a fine-grained security policy can be realized for an existing resource using lookup service middleware and a security resource which itself is distributed.

4. Maintaining a method focus on resource APIs as it is the method that is often used to manipulate objects. The objective here was to use the method signature to control API access based on signature constraints. We were successful in using this approach to build very fine-grained access to public APIs by user role.

7.2 Future Research

There are ongoing efforts in a number of areas. First, for methods, we are researching finer-grained MAC that would allow different method parameters to have different CLSs and constraints. There are arguments that once a query is started, there is no way, other than hardware failure, to prevent a result from being returned to the user. Potentially, a misguided query could generate sensitive information for which the user is not cleared, creating a security violation. We would like to leverage Jini's security platform (sandbox) and execute a return parameter check before results are returned to the user. This would add another level of security assurance by preventing potentially harmful queries from being initiated accidently and the results returned in violation of policy. Second, we are assessing our security model approach using CMU's Systems Security Engineering Capability Maturity Model (SSE-CMM), an accepted ISO standard used in industry. The SSE-CMM was designed to help organizations develop a security program covering all aspects of security from the physical to the digital. Our interest, of course, is in the automation/digital aspect of information security. The SSE-CMM provides a comprehensive list of security considerations and evaluation criteria so an organization can quantify their security posture and make plans to improve. This examination will provide valuable insight to possible flaws and future work to our security model. Third, we are examining the applicability of the core ideas of our security approach in Chapter 3 to XML documents. XML is a protocol for managing information and is fast becoming a standard way of storing data. With XML, we can manage, format and filter data, which may produce security assurance improvements to our model. If we can leverage this standard, we can enhance the portability and usefulness of our model. Next, we have started work on mutual exclusion with respect to role deconfliction and mutual exclusion of methods. Role deconfliction is helpful for separation of duty issues where one user should not be given too much responsibility our conflicting roles. We will look at creating a deconfliction list for a user role when the user role is established (design time) to immediately identify what other roles cannot be combined for a sole user. We will also attempt to do the same checks for methods within roles. Finally, in the longer-term, role hierarchies, and credential-based

access control for resources that do not require specific user identification, are all of interest. We have done work on role hierarchies which makes building roles more efficient, but have not worked that into our current model. Credential-based assess control can have profound impact on security policy because first, credentials will have to be generated then verified (runtime), and second, the organization will not assign individuals to roles, only the credentials necessary to execute that role. This is an interesting twist on access control policy and will raise assurance issues.

Our work in distributed security and access control is ongoing; please see [139] for further information.

Bibliography

[1] PEO C3S HTIO, "NATO Interoperability,Advanced Concept Technology Demonstration, Management Plan," Fort Monmoth, NJ 1999.

[2] G. Ahn and R. Sandhu, "Towards Role-Based Administration In Network Information Services," *Journal Of Network And Computer Applications.*, Vol. 22, 3, Nov. 1999, p. 199-213.

[3] G. Ahn and R. Sandhu, "Role-Based Authorization Constraints Specification", *ACM Transactions on Information and System Security (TISSEC)*, Vol. 3, Issue 4 (November 2000), ACM Press New York, NY, USA, pp. 207 - 226.

[4] K. Alford, et al., "Information Assurance Pedagogy," *Proc. of IEEE Info. Assurance Wksp.*, June 2001.

[5] B. Alpern and F. Schneider, "Defining Liveness," *Information Processing Letters*, Vol. 21, No. 4 Oct. 1985.

[6] K. Arnold, et al., "The JINI Specification," Addison-Wesley, 1999.

[7] R. Awischus, "Role Based Access Control with Security Administration Manager," *Proc. of 2nd ACM Wksp. on RBAC*, Nov. 1997.

[8] J. Baras, V. Gligor, and R. Poovendran, "Integrated Security Services for Dynamic Coalition Management," DARPA ACT Program, March 2001.

[9] J. Barkley, "Implementing Role-Based Access Control Using Object Technology", *Proc. of First ACM Workshop on Role-Based Access Control*, Gaithersburg, MD, November 1995.

[10] J. Barkley, "Comparing Simple Role Based Access Control Models and Access Control Lists," *Proceedings of The 2nd ACM Workshop on Role-Based Access Control*, Virginia, Nov. 1997

[11] E. Barka and R. Sandhu, "Framework for Role-Based Delegation Models," *Proc. of 23rd Natl. Info. Sys. Security Conf.*, Oct. 2000.

[12] P. Barr, "ABCS ITDS", MITRE Corporation presentation, NJ, Oct. 1998.

[13] D. Bell and L. LaPadula, "Secure Computer Systems: Mathematical Foundations Model," M74-244, Mitre Corp., Bedford, MA, 1975.

[14] E. Bertino, et al., "A Temporal Access Control Mechanism For Database Systems", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 1, Feb. 1996.

[15] E. Bertino, E. Ferrari, V. Atluri, "Flexible Model Supporting The Specification And Enforcement Of Role-Based Authorizations In Workflow Management Systems", *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, Maryland, Nov. 1997.

[16] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems", *ACM Transactions on Information Systems Security*, Vol. 2, No. 1, Feb. 1999.

[17] E. Bertino, et al., "TRBAC: A Temporal Role-Based Access Control Model," *Proc. of 5th ACM Wksp. on RBAC*, July 2000.

[18] K. J. Biba, "Integrity Considerations for Secure Computer Systems," TR-3153, Mitre Corp, Bedford, MA, 1977.

[19] S. Boutelle and C. Pizzutelli, "Army Battle Command System," *Army RD & A*, Sept./Oct. 1998.

[20] H. H. Bruggemann, "Rights in an Object-Oriented Environment", in *Database Security, V: Status and Prospects*, C. Landwehr and S. Jajodia (eds.), North-Holland, 1992.

[21] PEO C3S HTIO, "Command, Control, Communications, and Computers Interoperability for Coalition Warfare, Advanced Concept Technology Demonstration, Management Plan," Ver. 1.2, Mar. 1999.

[22] M. Cokus, "XML-MTF, A Military XML Vocabulary," The MITRE Corporation, 2001.

[23] DARPA ITO Sponsored Research, "2000 Project Summary, Flexible Coalition Policies for Secure Information Sharing," Verdian-PSR, 2000.

[24] S. Demurjian, T.C. Ting, and M. Hu, "Role-based access control for object-oriented/C++ systems," *Proceedings of the 1st ACM Workshop on Role-based Access Control*, Maryland, Nov./Dec. 1995.

[25] 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.

[26] S. Demurjian, et al., "Software Architectural Alternatives for User Role-Based Security Policies", in *Database Security, XI: Status and Prospects*, Lin/Qian (eds.), Chapman Hall, 1998.

[27] S. Demurjian, et al., "Software Agents for Role Based Security", in *Database Security, XII: Status and Prospects*, S. Jajodia (ed.), Kluwer, 1999.

[28] S. Demurjian, T.C. Ting, H. Ren, J. Balthazar, C. Phillips, and P. Barr, "A User Role-Based Security Model for a Distributed Environment," *Research Advances in Database and Information Systems Security*, J. Therrien (ed.), Kluwer, 2001.

[29] S. Demurjian, et al., "Concepts and Capabilities of Middleware Security," to appear in *Middleware for Communications*, Q. Mohammed (ed.), John-Wiley, 2004.

[30] D. Denning, et al., "Views for Multilevel Database Security", *Proceedings. of IEEE 1986 Symposium on Security and Privacy*, May 1986.

[31] DevX Enterprise Zone. (2002) Software Engineers Put .NET and Enterprise Java Security to the Test. http://www.devx.com/enterprise/articles/dotnetvsjava/GK0202-1.asp.

[32] Department of Defense Directive 5200.28-STD, "Department of Defense Trusted Computer Systems Evaluation Criteria," December 1985, Authorized by DoD Directive 5200.28, Dec. 1972.

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

[34] Department of Defense Directive 8320.1-M-1, Department of Defense, "Data Standardization Procedures," March 1996. http://jcs.mil/htdocs/teinfo/software/8320.html

[35] Department of Defense Instruction 5200.40, "DOD Information Technology Security Certification and Accreditation Process (DITSCAP)," Dec. 1997.

[36] K. Edwards, Core JINI, Prentice-Hall, 1999.

[37] W. Essamyr, et al., "Using Role-Templates for Handling Recurring Role Structures," *Database Security, XI: Status and Prospects*, Lin/Qian (eds.), Chapman Hall, 1998.

[38] R. Reagan, Executive Order 12356, "National Security Information," The White House, Apr. 1982.

[39] D. Ferraiolo and R. Kuhn. "Role-Based Access Controls," *Proceedings of 15th NIST-NCSC National Computer Security Conference*, Oct. 1992.

[40] D. Ferraiolo, J. Barkley, and D. Kuhn , "A Role-Based Access Control Model and Reference Implementation within a Corporate Internet," *ACM Transactions on Information and System Security*, Vol. 1, No. 2, Feb. 1999.

[41] D. Ferrailo, "The Role Control Center: An Implementation of Role-Based Access Control on Identity-Based Systems," NIST White Paper, 2000.

[42] D. Ferraiolo, "An Argument For The Role-Based Access Control Model," *Proceedings of 6th ACM Symposium on Access Control Models and Technologies*, Virginia, May 2001.

[43] A. Fox and S. Gribble, "Security on the Move: Indirect Authentication Using Kerberos", *ACM MOBICON 96*, Rye, NY, 1996.

[44] E. Freeman, et al., JavaSpaces Principles, Patterns, and Practice, Addison-Wesley, 1999.

[45] S. Gavrila and J. Barkley, "Formal Specification For Role Based Access Control User/Role and Role/Role Relationship Management," *Proc. of the 3rd ACM Wksp. on RBAC*, Oct. 1998.

[46] Joint Operational Support Ctr., http://gccs.disa.mil/gccs/, 1999.

[47] S. Greenwald, "A New Security Policy for Distributed Resource Management and Access Control", *1996 ACM New Security Paradigms Wksp.*, Lake Arrowhead, CA, Sept. 1996.

[48] W. Harrison and H. Ossher, "Subject-Oriented Programming (A Critique of Pure Objects)", *Proc. of 1993 OOPSLA Conf.*, Oct. 1993.

[49] M. Hu, S. Demurjian, T.C. Ting, "Unifying Structural and Security Design and Analysis in the ADAM Object-Oriented Design Environment", *Database Security VII: Status and Prospects*, Edited by J. Bishop, M. Morgenstern, and C. E. Landwehr, North-Holland, 1994.

[50] Y.-K. Hsu and S. Seymour, "An Intranet Security Framework Based on Short-Lived Certificates", IEEE Internet Computing, Vol. 2, No. 2, March-April 1998.

[51] T. Jaeger, and A. Prakash, "Requirements Of Role-Based Access Control For Collaborative Systems," *Proceedings of the 1st ACM Workshop on Role-based Access Control*, Maryland, Nov/Dec. 1995.

[52] T. Jaeger, F. Giraud, N. Islam, and J. Liedtke, "A Role-Based Access Control Model For Protection Domain Derivation And Management," *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, Virginia, Nov. 1997.

[53] T. Jaeger, "On The Increasing Importance Of Constraints." emphProceedings of the 4th ACM Workshop on Role-Based Access Control, Virginia, Oct. 1999.

[54] T. Jaeger and J. Tidswell, "Rebuttal to the NIST RBAC Model Proposal," emphProceedings of the 5th ACM, RBAC 00, Germany, Jul. 2000.
[55] JIEO Handbook 9000, Chapter One, "General Instructions," Joint Information Exchange Operations, Department of Defense Handbook, March 2000.
[56] http://www.sun.com/jini/
[57] http://www.sun.com/jini/whitepapers/architecture.html
[58] Joint Operational Support Center. "Global Command and Control Center," DISA, 1999. http://gccs.disa.mil/gccs/
[59] R. . Kemmerer, "Security Issues in Distributed Software", Proc. of the 6th European Conf. On Software Engineering, held jointly with the 5th ACM SIGSOFT, 1997.
[60] D. Kindred and K. Djahandari, "Adaptive Network Defense, Dynamic Virtual Private Network," Networks Associates Technology, Inc., NAI Labs, 2001.
[61] H. Korth and A. Silberschatz, Database Systems Concepts, "Security and Integrity", Chap. 13, McGraw-Hill, 1986.
[62] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Trans. on Software Engineering*, Vol. 3, No. 2, Mar. 1977.
[63] L. Lamport, "Logical Foundation," in *Distributed systems: Methods and Tools for Specification,* multiple eds., LNCS, Vol. 190, Springer-Verlag, 1985.
[64] B. Lampson, et al., "Authentication in Distributed Systems: Theory and Practice", ACM Trans. On Computer Systems, Vol. 10, No. 4, November 1992.
[65] C. Landwehr, et al., "A Security Model for Military Message Systems", *ACM Trans. on Computer Systems*, Vol. 2, No. 3, Sept. 1984.
[66] S. Levine, "Army Modernization: Digitization and Transformation Overview," briefing at Pentagon, April 2000.
[67] M. Liebrand, et al., "Role Delegation for a Resource-Based Security Model," *Data and Applications Security: Developments and Directions II*, E. Gudes and S. Shenoi (eds.), Kluwer, 2003.
[68] 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 RBAC*, Oct. 1999.
[69] F. Lochovsky and C. Woo, "Role-Based Security in Data Base Management Systems", *Database Security: Status and Prospects*, Landwehr(ed.), North-Holland, 1988.
[70] T. Lunt and D. Hsieh,"The SeaView Secure Database System: A Progress Report", *Proceedings of 1990 European Symposium on Research in Computer Security*, Oct. 1990.
[71] W. Maconachy, et al., "A Model for Information Assurance: An Integrated Approach," *Proc. of IEEE Info. Assurance Wksp.*, June 2001.
[72] J. McCumber, "Information Systems Security: A Comprehensive Model," *Proc. of the 14th National Computer Security Conf.*, NIST, Oct. 1991.
[73] J. McLean, "Security Models," *Enc. of Software Engineering*, Wiley Press, 1994.
[74] Microsoft Corporation, "The Component Object Model (Technical Overview)", Microsoft Press, Redmond, WA, 1995.
[75] Microsoft Corporation (2003a). Microsoft DN, Microsoft .NET Security. http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001369.

[76] M. Millikin, "Distributed Objects: A New Model for the Enterprise", Data Communications on the Web, http://www.data.com, Feb. 1997.
[77] C. Milster, M. Parish, G. Le Fevre, "Taking Digitization to Our Allies, " Army RD&A, Sep-Oct 1998.
[78] M. Morrison, et al., Java Unleashed, second edition, Sams.net Publishing, 1997.
[79] S. Na and S. Cheon, "Role Delegation in Role-Based Access Control," *Proc. of 5th ACM Wksp. on RBAC*, July 2000.
[80] NATO Interoperability from "Advanced Concept Technology Demonstration, Management Plan," PEO C3S HTIO, Fort Monmoth, NJ. 1999.
[81] C. Nueman and T. Ts'o, "An Authorization Service for Computer Networks", Comm. of the ACM, Vol. 32, No. 9, Sept. 94.
[82] Quotation from the National Military Strategy, "C4I For Coalition Warfare, Command and Control Systems Interoperability Program," Army Digitization Office, 1999.
[83] Notargiacomo, L., "Role-Based Access Control in ORACLE7 and Trusted ORACLE7," *Proc. of the 1st ACM Wksp. on RBAC*, November 1995.
[84] M. Nyanchama and S. L. Osborn. "Role-Based Security, Object Oriented Databases & Separation of Duty." *ACM SIGMOD RECORD*, 22(4):45–51, Dec. 1993.
[85] M. Nyanchama and S. Osborn, "Modeling Mandatory Access Control in Role-Based Security Systems", *Proceedings of IFIP Workshop on Database Security*,. Jul. 1995.
[86] Object Management Group, "The Common Object Request Broker: Architecture and Specification - Revision 2.0", Framingham, MA, July 1995.
[87] Object Management Group (2002). Common Object Request Broker Architecture: Core Specification - Version 3.0.2. December. http://www.omg.org.
[88] Open Web Application Security Project (2002) J2EE and .NET Security. http://www.owasp.org/downloads/J2EEandDotNetsecurityByGerMulcahy.pdf.
[89] Oppliger, R. "Internet Security: Firewalls and Beyond", Comm. of the ACM, Vol. 40, No. 5, May 1997.
[90] S. Osborn, "Mandatory Access Control And Role-Based Access Control Revisited," *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, Virginia, Nov. 1997.
[91] S. Osborn, et al., "Configuring Role-Based Access Control to Enforce Mandatory And Discretionary Access Control Policies," *ACM Trans. on Information and System Security*, Vol. 3, No. 2, May 2000.
[92] Open Software Foundation, "OSF DCE Application Development Guide - revision 1.0", OSF, Cambridge, MA, 1994.
[93] H. Ossher, et al., "Subject-Oriented Composition Rules"", *Proc. of 1995 OOPSLA Conf.*, Oct. 1995.
[94] Prepared by PEO C3S HTIO, "Command, Control, Communications, and Computers Interoperability for Coalition Warfare, Advanced Concept Technology Demonstration, Management Plan," Ver. 1.2, March 1999.
[95] W. Peach, "Message Text Formats-A Solution to the Problem of Interoperability," Journal of Battlefield Technology, Vol. 2, March 1999.
[96] C. Phillips, et al., "Security Engineering for Roles and Resources in a Distributed Environment," *Proc. of 3rd ISSEA Conf.*, Mar. 2002.

[97] C. Phillips, et al., "Towards Information Assurance in Dynamic Coalitions," *Proc. of 2002 IEEE Info. Assurance Wksp.*, June 2002.

[98] C. Phillips, et al., "Assurance Guarantees for an RBAC/MAC Security Model," *Proc. of 17th Annual IFIP WG 11.3 Workong Conference on Database and Applications Security 2003.*, Estes Park, CO, August 2003.

[99] F. Rabitti, et al., "A Model of Authorization for Next Generation Database Systems", *ACM Trans. on Database Systems*, Vol. 16, No. 1, March 1991.

[100] American National Standard for Information Technology Role Based Access Control, DRAFT - 4/4/2003, National Institute of Standards and Technology, http://csrc.nist.gov/rbac/rbac-std-ncits.pdf

[101] M. Reiter and S. Stubblebine, "Authentication Metric Analysis and Design", *ACM Trans. On Information and System Security*, Vol. 2, No. 2, May 1999.

[102] J. Richardson and P. Schwarz, "Aspects: Extending Objects to Support Multiple, Independent Roles", *Proc. of 1991 ACM SIGMOD Conf.*, May 1991.

[103] Riordan, R. (2002). Microsoft ADO.NET Step by Step. Microsoft Press.

[104] R. Rivest and B. Lampson, "SDSI - A Simple Distributed Security Infrastructure", MIT and Microsoft Co., http://theory.lcs.mit.edu/ rivest/sdsi10.ps

[105] Roman, E. (1999). Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edi-tion. John-Wiley.

[106] W. Rosenberry, D. Kenney, and G. Fischer, Understanding DCE, O'Reilly & Associates, 1992.

[107] P. Rougeau and Stearns, "The Sybase Secure Databse Server: A Solution to the Multilevel Secure DBMS Problem", *Proc. of 10th Natl. Computer Security Conf.*, Oct. 1987.

[108] P. Samarati, "Access Control: Policies, Models, Architectures, and Mechanisms," FOSAD, Italy, Sept. 2000.

[109] R. Sandhu, "Lattice-Based Access Control Models," *Computer Journal*, Vol. 26, No. 11, Nov. 1993.

[110] R. Sandu and P. Samarati. "Access Control: Principles and Practice." *IEEE Communications Magazine*, Vol. 32, No.9, Sep. 1994.

[111] Sandhu, R., Coyne, E.J., Feinstein, H. L., and Youman, C. E.,. "Role-Based Access Control Models", *IEEE Computer* 29(2): 38-47, IEEE Press, 1996.

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

[113] R. Sandu and Q. Munawer, "The ARBAC99 Model for Administrative Roles," *Proceedings of 15th Annual Computer Security Applications Conference*, Arizona, Dec. 1999.

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

[115] Sceppa, D. (2002). Microsoft ADO.NET (Core Reference). Microsoft Press.

[116] F. Schneider, "Enforcable Security Policies," *ACM Trans. on Information and System Security*, Vol. 3, No. 1, Feb. 2000.

[117] M. Soshi and M. Maekawa, "The Saga Security System: A Security Architecture for Open Distributed Systems", Proc. of 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97), Tunis, Tunisia, 1997.

[118] D. Spooner, "The Impact of Inheritance on Security in Object-Oriented Database Systems", *Database Security, II: Status and Prospects*, Landwehr (ed.), North-Holland, 1989.
[119] S. Spring and D. Gormley, "Information Sharing for Dynamic Coalitions," VPSR Report 2836, Verdian Pacific-Sierra Research, Dec. 2000.
[120] "System Security Engineering Capability Maturity Model, SSE-CMM, Model Description Document", Ver. 2.0, Apr. 1999, 1999 by CMU.
[121] J. Shilling and P. Sweeney, "Three Steps to Views: Extending the Object-Oriented Paradigm", *Proc. of 1989 OOPSLA Conf.*, Oct. 1989. NGDM/NGDT.
[122] P. Stachour and B. Thuraisingham, "Design of LDV: A Multilevel Secure Relational Database Management System", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 2, June 1990.
[123] Sun Microsystems (2003). J2EE Security Model. Java 2 Platform Security. http://java.sun.com/j2se/1.4.1/docs/guide/security/spec/security-spec.doc.html.
[124] V. Swarup, "Trust Appraisal and Secure Routing of Mobile Agents", Proc. of 1997 Workshop on Foundations for Secure Mobile Code (DARPA), March 1997.
[125] R. Tennent, Principals of Programming Languages, Prentice Hall, London, 1981.
[126] S. Tezuka, et al., "Seamless Object Authentication in Different Security Policy Domains", Proc. of 33rd Hawaii Intl. Conf. on System Sciences, January 2000.
[127] M. Thibodeaux, D. Ragsdale, "Ethical Aspects of Information Assurance Education," *Proceedings of 2nd IEEE Information Assurance Workshop*, June 2001.
[128] M.B. Thuraisingham, "Mandatory Security in Object-Oriented Database Systems", *OOPSLA '89 Proceedings*, Oct. 1989.
[129] T.C. Ting, " A Role-Based Data Security Approach", *Database Security: Status and Prospects*, edited by C.E.Landwehr, North-Holland, 1987.
[130] T.C. Ting, "A User-Role Based Data Security Approach," *Database Security: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1988.
[131] T.C. Ting, "Application Information Security Semantics: A Case of Mental Health Delivery," *Database Security, III: Stauts and Prospects*, D. Spooner and C. Landwehr (eds.), North-Holland, 1990.
[132] Valesky, T. (1999). Enterprise JavaBeans: Developing Component-Based Distributed Appli-cations. Addison-Wesley.
[133] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", IEEE Communications Magazine, Vol. 14, No. 2, Feb. 1997.
[134] J. Waldo, "The JINI Architecture for Network-Centric Computing", Communications of the ACM, Vol. 42, No. 7, July 1999.
[135] Walsh, T., Paciorek, N., and Wong, D. "Security and Reliability in Concordia", Proc. of the 31st Hawaii Intl. Conf. on System Sciences (HICSS'98), 1998.
[136] W. Wulf, et al., "A New Model of Security for Distributed Systems", 1996 ACM New Security Paradigms Workshop, Lake Arrowhead, California, Sept. 1996.
[137] Z. Yang and K. Duddy, "CORBA: A Platform for Distributed Object Computing", ACM Operating Systems Review, Vol. 30, No. 2, April 1996.
[138] L. Zhang, et al., "A Rule Based Framework for Role-Based Delegation," *Proc. of 6th ACM SACMAT*, May 2001.
[139] http://www.engr.uconn.edu/ steve/DSEC/dsec.html
******

Appendix A

GUI for GCCS Example

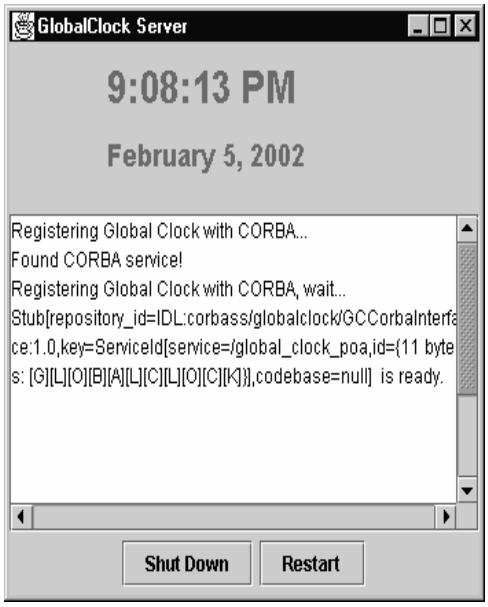


Figure 68: Start Global Clock Server.

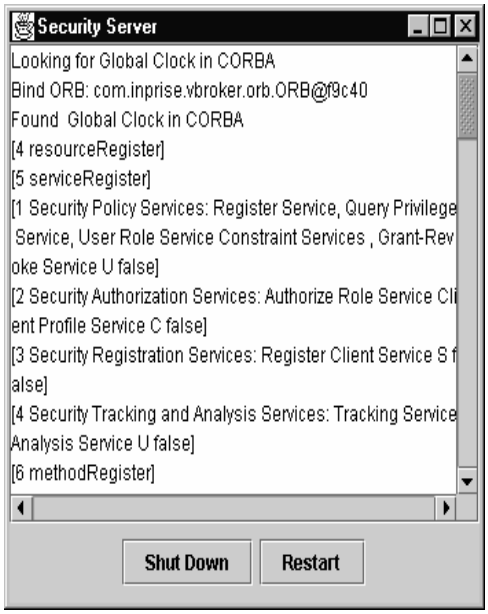


Figure 69: Start Security Server

Figure 70: Login: Policy Client Authentication

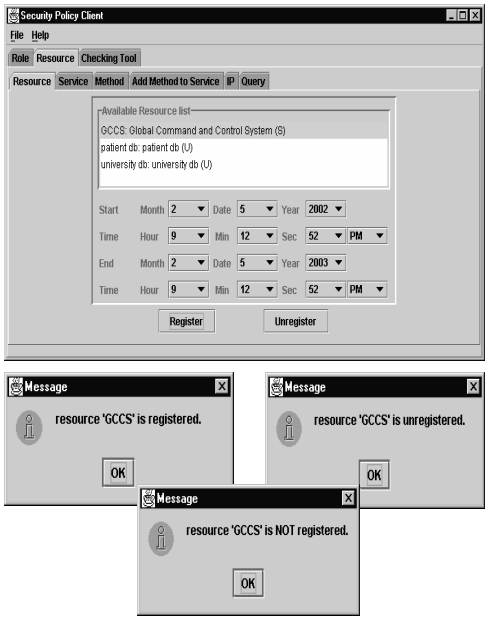


Figure 71: SPC: Register Resource

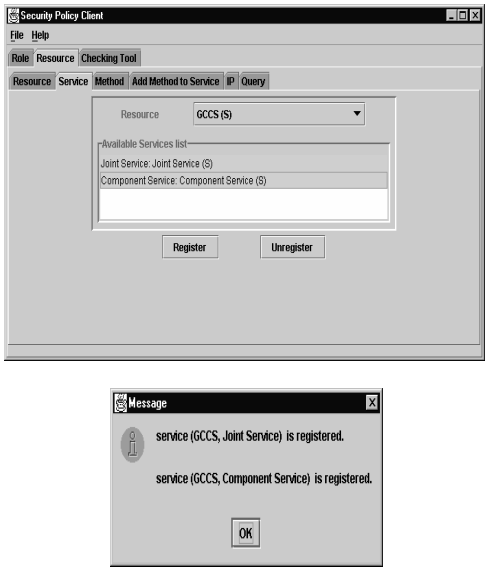


Figure 72: SPC: Register Service

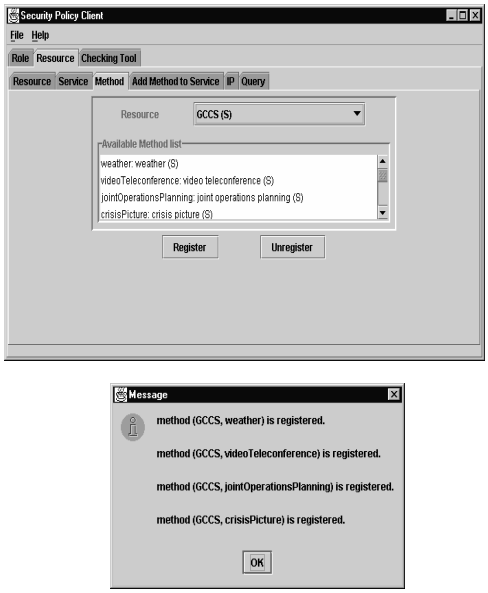


Figure 73: SPC: Register Methods

Figure 74: SPC: Added Method to Service

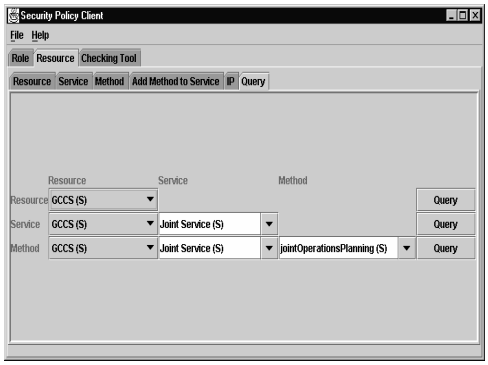


Figure 75: SPC: Resource Query

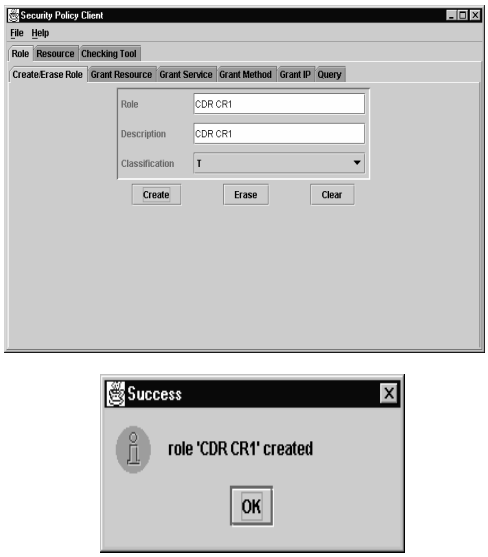


Figure 76: SPC: Create Role

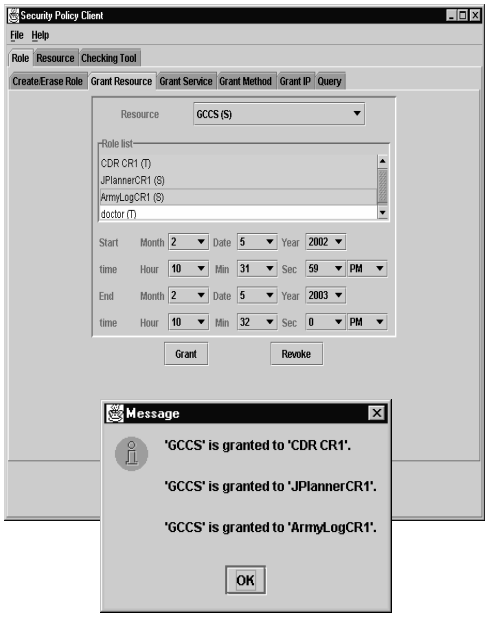


Figure 77: SPC: Add Resource to Role

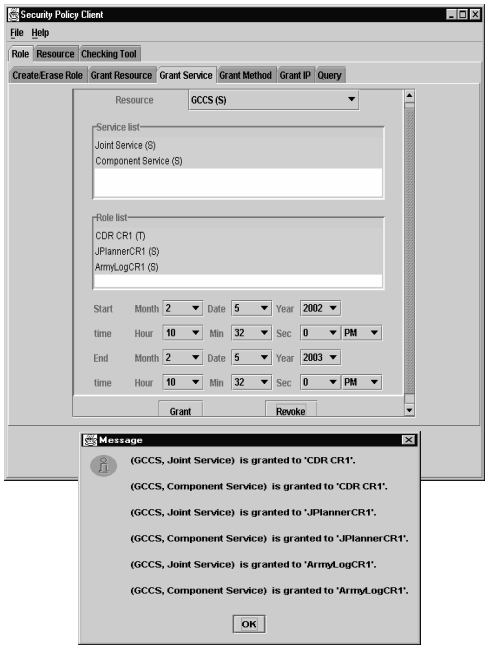


Figure 78: SPC: Add Service to Role

Figure 79: SPC: Add Method to a Role

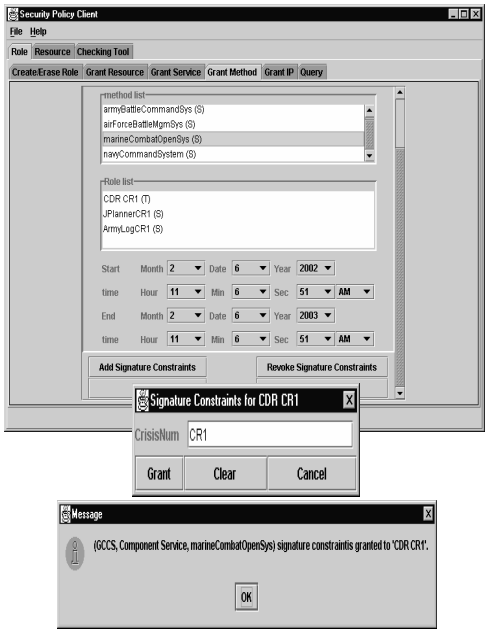


Figure 80: SPC: Add a Signature Constraint

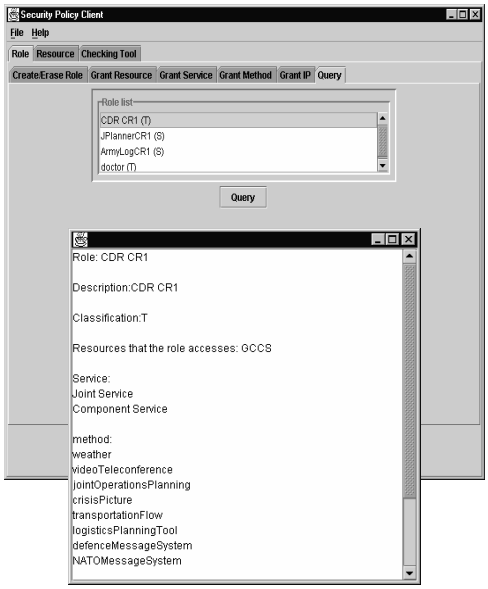


Figure 81: SPC: Query a Role

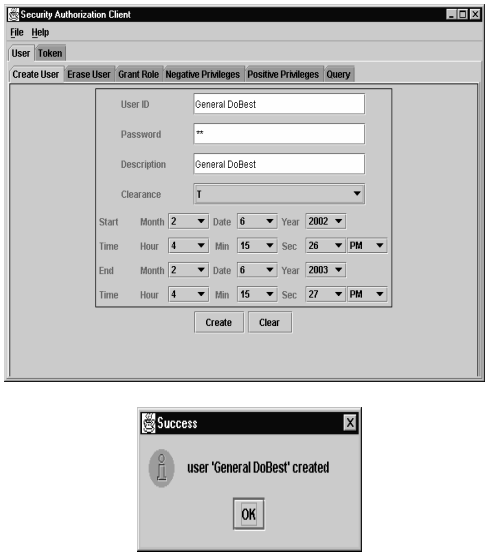


Figure 82: SAC: Create a User

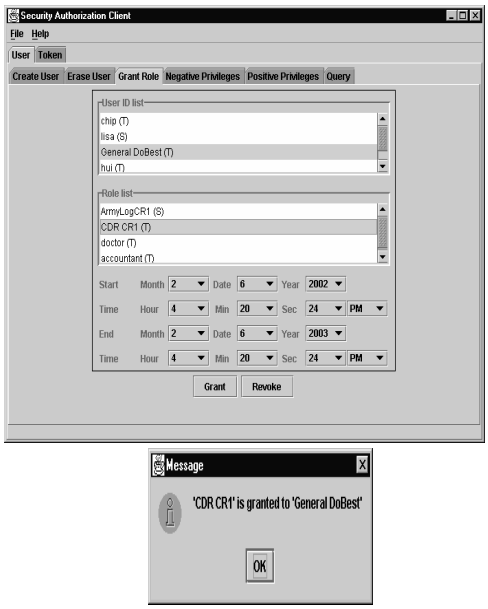


Figure 83: SAC: Grant Role to a User

Appendix B

Security Model Acronyms

Acronyms (Alphabetical Order)

C:	Client	(Definition 18)
CLR:	Clearance	(Definition 2)
CLS:	Classification	(Definition 2)
ct:	current time	(Definition 1)
DA:	Delegation Authority	(Definition 24)
DAM:	Delegation Authority Matrix	(Definition 26)
	$DAM(UR_i, U_j) = \begin{cases} 2 & U_j \text{ has DA and PODA for } UR_i \quad (26.1) \\ 1 & U_j \text{ has only DA for } UR_i \quad (26.2) \\ 0 & U_j \text{ has neither DA nor PODA for } UR_i \quad (26.3) \end{cases}$	
DAPPL:	Distributed Application	(Definition 3)
DU:	Delegated User	(Definition 22)
DUR:	Delegatable User-Role	(Definition 19)
DURV:	Delegatable User-Role Vector	(Definition 20)
	$DURV(UR_i) = \begin{cases} 1 & UR_i \text{ is a DUR} \\ 0 & UR_i \text{ is not a DUR} \end{cases}$	
et:	end time	(Definition 1)
LT:	lifetime	(Definition 1)
M:	Method	(Definition 4)
MACC:	Mandatory Access Control	(Definition 13)
OU:	Original User	(Definition 21)
PODA:	Pass-on Delegation Authority	(Definition 25)
R:	Resource	(Definition 6)
S:	Service	(Definition 5)
SC:	Signature Constraint	(Definition 11)
SLEVEL:	sensitivity level	(Definition 2)
st:	start time	(Definition 1)
TC:	Time Constraint	(Definition 12)
U:	User	(Definition 9)
UA:	User Authorization	(Definition 16)
UAM:	User Authorization Matrix	(Definition 17a)
	$UAM(UR_i, U_j) = \begin{cases} 1 & U_j \text{ is authorized to } UR_i \\ 0 & otherwise \end{cases}$	
UDAM:	User Delegation/Authorization Matrix	(Definition 23)
	$UDAM(UR_i, U_j) = \begin{cases} 2 & U_j \text{ is a DU of } UR_i \quad (23.1) \\ 1 & U_j \text{ is an OU of } UR_i \quad (23.2) \\ 0 & U_j \text{ is not authorized to } UR_i \quad (23.3) \end{cases}$	
UL:	User List	(Definition 10)
UR:	User Role	(Definition 7)
URL:	User-Role List	(Definition 8)
URA:	User-Role Authorization	(Definition 14)
URAM:	User-Role Authorization Matrix	(Definition 15)
	$URAM(UR_i, M_j) = \begin{cases} 1 & UR_i \text{ is authorized to invoke } M_j \\ 0 & otherwise \end{cases}$	
VUA:	Valid User Authorization	(Definition 17a)
VUAL:	Valid User Authorization List	(Definition 17b)
VURA:	Valid User-Role Authorization	(Definition15a)
VURAL:	Valid User-Role Authorization List	(Definition 15b)

Figure 84: Acronyms - Alphbetical Order

Acronyms (Order of Appearance)

LT:	lifetime	(Definition 1)
et:	end time	(Definition 1)
st:	start time	(Definition 1)
ct:	current time	(Definition 1)
SLEVEL:	sensitivity level	(Definition 2)
CLR:	Clearance	(Definition 2)
CLS:	Classification	(Definition 2)
DAPPL:	Distributed Application	(Definition 3)
M:	Method	(Definition 4)
S:	Service	(Definition 5)
R:	Resource	(Definition 6)
UR:	User Role	(Definition 7)
URL:	User-Role List	(Definition 8)
U:	User	(Definition 9)
UL:	User List	(Definition 10)
SC:	Signature Constraint	(Definition 11)
TC:	Time Constraint	(Definition 12)
MACC:	Mandatory Access Control	(Definition 13)
URA:	User-Role Authorization	(Definition 14)
URAM:	User-Role Authorization Matrix	(Definition 15a)

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

VURA:	Valid User-Role Authorization	(Definition 15a)
VURAL:	Valid User-Role Authorization List	(Definition 15b)
UA:	User Authorization	(Definition 16)
UAM:	User Authorization Matrix	(Definition 17a)

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

VUA:	Valid User Authorization	(Definition 17a)
VUAL:	Valid User Authorization List	(Definition 17b)
C:	Client	(Definition 18)
DUR:	Delegatable User-Role	(Definition 19)
DURV:	Delegatable User-Role Vector	(Definition 20)

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

OU:	Original User	(Definition 21)
DU:	Delegated User	(Definition 22)
UDAM:	User Delegation/Authorization Matrix	(Definition 23)

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

DA:	Delegation Authority	(Definition 24)
PODA:	Pass-on Delegation Authority	(Definition 25)
DAM:	Delegation Authority Matrix	(Definition 26)

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

Figure 85: Acronyms - Order of Appearance