Towards Information Assurance for Dynamic Coalitions

Charles E. Phillips, Jr., Steven A. Demurjian, and T.C. Ting

Abstract - Distributed applications for day-to-day operations of corporations and government agencies are composed of inter-operating legacy, COTS, databases, clients, servers, etc. One critical challenge, therein, is information assurance and security, to insure that users are accessing only information to which they have been authorized. In the past three years, we have designed and prototyped a security model and enforcement framework that controls access to the APIs of software artifacts which operate in a distributed environment running middleware (CORBA, JINI, etc.). The effort has concentrated on discretionary access control, to limit, on a role-by-role basis, which users (clients) can access which parts of artifact APIs, constrained by time and data values. In this paper, we extend this work by: incorporating mandatory access control to limit access of users based on security clearance level; and, examining the degree that information assurance can be attained via the characteristics and tools of our model/framework. To facilitate our discussion, we utilize examples involving dynamic coalitions.

I. Introduction

Information assurance represents the confidence level of the security capabilities that are embodied and integrated in distributed environments, to enforce the security policy for all users, thereby insuring that sensitive information is protected from access and misuse, which is critical in both DoD [1] and multi-national settings [2]. Mandates for addressing information sharing for coalition forces [3] are augmented by directives that require multilevel security by DoD [4] and NATO [5]. Information sharing and assurance is essential for international coalitions of non-government agencies and their military counterparts to be successful. Of particular attention for this paper, are the assurance requirements in government and industry systems that utilize classified/sensitive information that must be controlled from use and misuse.

Research and development for classification-based security [6, 7, 8, 9, 10, 11, 12, 13] leverages the Bell and Lapadula Model (BLM) [14] to support mandatory access control (MAC) to classify and tag data. Alternatively, discretionary access control (DAC) [15, 16, 17] emphasizes the responsibilities of end users when establishing access rights. DAC can provide discretion to allow the APIs of software artifacts to be customizable and restricted on an individual-by-individual basis against a distributed application. Role-based access control (RBAC) [18] can augment DAC and MAC, allowing user roles to be defined that promote discretion against software artifacts (DAC – which portions of APIs can be invoked based on the responsibilities of a user role) while limiting access to classified software artifacts (MAC – only access portions of APIs based on the security level of a user). Our current research has designed a comprehensive security model for a distributed setting which grants and/or denies the invocation of methods (APIs of software artifacts) by roles, based on allowable data values and valid time intervals, realized in a distributed enforcement framework [19, 20], expanding our prior RBAC/DAC work in an object-oriented setting [21]. The first focus of this paper is to extend the security model and enforcement framework by including MAC, allowing a security officer to assign clearance levels to methods, to control, at runtime, the methods to be invoked based on the security level of a user.

In order for the security model and enforcement framework to be useful, it must be capable of capturing the intricate details of a security policy, which are application dependent and vary widely from application to application. Moreover, a security policy can be impacted by the security approach (MAC and/or DAC) and the underlying software/computational model (database vs. application program vs. component API). Once defined, the ultimate responsibility for a security policy is on the shoulders of the organization’s management personnel and security officer. In order to have these critical policy makers take full advantage of security capabilities and design techniques, associated tools are critical to allow software engineers and security officers to accurately and precisely specify functional and security requirements. The second focus of the paper involves a detailed examination of the characteristics and tools of our security model and enforcement framework towards supporting information assurance with respect to the security policy for a distributed application.

In both foci, we employ, as a test bed, the Dynamic Coalition Problem (DCP), an alliance of governmental, military, civilian, and international agencies, formed with the primary concern of the most effective way to solve the crisis. DCP has inherent security, resource, and information sharing risks that occur as a result of the coalition forming quickly, while needing information and resource sharing for crisis resolution [22]. DCP is ideal for examining information assurance in our unified RBAC/MAC security model/enforcement framework [19].

The remainder of this paper has four sections. In Section II, we examine the dynamic coalition problem (DCP) with respect to security policy. In Section III, we present a unified RBAC/MAC model for information assurance. In Section IV, we discuss the ability of the security enforcement framework to support information assurance. Conclusions and future work are reviewed in Section V.

Authors are with the Computer Science and Engineering Dept., Univ. of Connecticut, Storrs, CT, 06269-3155 [charlesp, steve, ting] @engr.uconn.edu, (860)486-4818

ISBN 555555/$10.00 © 2002 IEEE
II. Dynamic Coalition Problem and Information Assurance with Respect to Access Policy

Information sharing in response to a crisis (e.g., natural disaster, humanitarian relief, international incident, war, combat operations other than war, etc.) is of critical importance, and often necessitates the formation of a dynamic coalition. Information assurance in dynamic coalitions is a complex task with many considerations arising during its formation, management, and lifetime, including: federate groups of users quickly and dynamically in response to a crisis; bring together resources/artifacts without modification for usage in support of the crisis; dynamically realize and manage a security policy during simultaneous crises; identify users by their roles to finely tune their access in support of a crisis; authorize, authenticate, and enforce a scalable security policy to respond to changing coalition needs; provide a distributed security solution that is portable, extensible, and redundant for survivability; and offer robust security policy definition, management, and introspection capabilities to track and monitor system behavior and activities of users [19]. The Dynamic Coalition Problem (DCP) also encompasses other wide-ranging issues (i.e., interoperability, sharing, security, extensibility, scalability, etc.) that must be undertaken to form, maintain, and eventually disperse the coalition. The remainder of this section discusses civilian agency and military involvement in coalitions (Section II.A), the limitations of the Global Command and Control System (GCCS) (Section II.B), and access control policies for sharing and assurance in DCP (Section II.C).

A. Civilian and Military Organizations

There are many civilian organizations that contribute to the successful resolution of a crisis. These organizations can be governmental (agencies, embassies, and bureaus), non-governmental (NGOs), and private (PVOs), for handling crises such as: humanitarian relief, nation building, disaster relief, diplomatic problems, refugee situations, etc. International NGOs and PVOs usually stay involved in a crisis until the country is self-sufficient and the country's private organizations can take over responsibility. Figure 1 depicts a general network architecture of the participants (NGOs, PVOs, and military) that take responsibility at different times, need to share information and resources, and must ensure smooth transitions through each phase of a crisis [23].

Military forces are often used in crisis situations. Information or capabilities sharing is difficult within U.S. military services (Army, Navy, Air Force, Marines, or Coast Guard), and the problems are exacerbated in a quickly formed coalition [24]. As coalitions become more complex, the risk to information assurance increases, which includes risk to classified intelligence information. Security mechanisms need to work in joint and combined environments, where 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. The information assurance problem involves classifying information, encrypting data paths, interoperability, controlling multinational access to resources, and adapting to different generations of technology. The current inability to effectively bring international users and their assets (resources) efficiently and securely together in a crisis is unfortunate, since the actual infrastructure (e.g., localized networks and information resources) can be easily and quickly linked.

B. Limitations of GCCS

The U.S. Global Command and Control System (GCCS) is the automation tool that provides a local U.S. commander with operational awareness of the situation (crisis) in near real-time through integrated sets of services. GCCS provides information-processing support to planning, mobility, sustainment, and messaging, by bringing together 20 separate automated systems in over 625 locations worldwide [25] in a private network. Unfortunately, GCCS does not satisfy all of the needs of a dynamic coalition. GCCS needs to be part of or protected by a distributed security system that can make it a coalition asset, while respecting both coalition and U.S. security policies. 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 for GCCS and other command and control systems to be acceptable for coalition use, several information sharing and security issues need to be addressed, as presented in earlier work [19, 20]. First, dynamic user role administration can be invaluable for coalition partners to share information and resources, providing a mechanism to restrict access based on the least privilege principle. Second, time controllable access to information placed on users, user roles and resources can facilitate proper use by authorized users at strict time intervals. Third, constraints on resources that focus on allowable values can protect sensitive information and allow coalition partners to participate in the crisis. Finally,
C. DAC, RBAC and MAC for DCP

Successful information assurance in coalitions will require a detailed security policy that defines access control (what operations are performed on what resource, by whom) and information flow [26]. Authorization, authentication, and enforcement mechanisms, will all be an integral part of any coalition. DAC, RBAC, and MAC offer many of the capabilities needed by coalitions, particularly as it relates to assurance. DAC provides a means of restricting access 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 [27]. For DCP, DAC must be carefully administrated, managed, and controlled to maintain information assurance and to limit the ability to pass on access restrictions or disseminate sensitive information by changing ownership. 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 [22].

RBAC, a realization of DAC, regulates a user’s access to certain resources based on a user role, the collection of permissions the user needs to accomplish his/her responsibilities. By controlling access via roles and permissions, a security policy can be realized and information assurance achieved. RBAC has been touted for its ability to support non-traditional security applications, where flexibility of usage is crucial [18, 28, 29]. Not only does RBAC provide flexibility, it is the best for supporting least privilege, the ability to access only that information which is necessary to accomplish one’s tasks [30]. This is a key concern to the military and coalitions in achieving information assurance. There are different RBAC approaches that allow for fine-grained role definition [21], which leads to improved granularity on access controls and assurance. A temporal RBAC approach [31] is relevant to DCP due to its rapidly changing needs and environment.

The Bell and LaPadula Model (BLM) for mandatory access control (MAC) [14] is considered a classic security model and is the basis of the Orange Book, which dictates security policy standards for the U.S. Government [27]. The two key concepts of MAC are: clearance (CLR), the authorized sensitivity level to which an individual user or client can access information, usually associated with a “need to know” requirement; and, classification (CLS), the sensitivity level given to information or objects based on the security policy. CLR and CLS have values that are based on the impact of national security by the exposure of the information: unclassified (U) or no impact; confidential (C), expected to cause some damage; secret (S), expected to cause serious damage; and top secret (T) expected to cause exceptionally grave damage [32]. These security concepts apply to U.S. information. Different countries not only have different security requirements, but also apply different labels, making translation between CLR/CLS levels a problem for dynamic coalitions. Furthermore, it will be a tedious and difficult task to carefully define the CLR/CLS levels for coalition partners, particularly since coalitions will include past adversaries. In order to achieve assurance in a crisis, security systems need to allow for the normalization of CLRs/CLSs and for adherence to individual country assurance requirements.

III. A SECURITY MODEL FOR INFORMATION ASSURANCE

MAC requires users (clients) to hold security clearances and all information objects (or methods) to carry classifications, as realized in systems like, GCCS. BLM is a relationship between hierarchical CLR and CLS, where a user can always read at the CLS level or below but never read a higher CLS. In the same vein, the same user can write to a higher level than the given security level (write up, read down). Collectively, from lowest to highest, there is a CLR/CLS hierarchy: null < U < C < S < T [32]. In order to fully support the security policy requirements of GCCS and other DCPs, a unified solution is needed that complements flexible RBAC (where flexible means the best for realization of a dynamic security policy [28]) with MAC (which can impose strict limits for select and critical security situations). While we concentrated on RBAC security in our earlier models [20], in this section we extend our model to support MAC. Our approach labels all users (clients) with CLRs and all user roles and methods of APIs with CLSs, and incorporates MAC rules at both definitional and runtime levels to dynamically enforce MAC, thereby increasing assurance. Our research is in sharp contrast to MAC for object-oriented systems [13], which has focused on instances (objects) rather than on the methods. In this section, we unify the security model previously introduced [20] by incorporating MAC extensions.

A. Core Security Model Definitions

This section introduces the core definitions for the security model formalized in [20] without MAC, with Definition 3a added and Definition 5 extended to support MAC.

Definition 1: A distributed application has \( m \) uniquely identifiable software/system resources (e.g., a legacy, COTS, DB, etc.), defined as: \( R = \{ R \mid i = 1 \ldots m \} \).

Definition 2: Each resource \( R_i, i = 1 \ldots m \), has \( n_i \) uniquely identifiable services, defined as: \( S = \{ S_j \mid j = 1 \ldots n_i \} \).
Definition 3: Each service \( S_j, j = 1 \ldots n_j \), of resource \( R_i, i = 1 \ldots m \), has \( q_{ij} \) uniquely identifiable methods, defined as: \( M_j = \{ M_{jk} | k = 1 \ldots q_{ij} \} \).

Definition 3a: Classification levels are: each \( \{ M_{jk} | k = 1 \ldots q_{ij} \} \) is chosen from the set \{U, C, S, T\}, where U is the default (no MAC), \( S_{ij}^{CLS} = \min \{ M_{jk}^{CLS} | k = 1 \ldots q_{ij} \} \) and \( R_{ij}^{CLS} = \min \{ S_{ij}^{CLS} | j = 1 \ldots n_j \} \) for \( i = 1 \ldots m \), where min is "least" secure.

Definition 4: Each method \( M_{ijk} \), for some \( i = 1 \ldots m, j = 1 \ldots n_j, k = 1 \ldots q_{ij} \) of service \( S_j \) of resource \( R_i \) is unique, and has a signature consisting of: the method name, \( N_{ijk} \) (may be overloaded), a parameter list, \( P_{ijk} \), of parameter name and parameter type (which may be null), and a single value return type, \( RT_{ijk} \) (null, basic, or object).

Definition 5: A user role, UR, is a uniquely named entity \( UR^{URName} \) representing a specific set of responsibilities against an application. A UR is assigned a CLS, \( UR^{CLS} \in \{ U, C, S, T \} \), where U is the default (no MAC). In summary, \( UR = [UR^{URName}, UR^{CLS}] \).

Representative user roles for Crisis 1 would be Commander <CDR_CR1, Top Secret>, Joint Planner <JPlannerCR1, Secret> and Army Logistics Officer <ArmyLogCR1, Secret>. From a privilege perspective, URs will be granted access to resources, services, and methods which have classification levels that are at or below the role’s classification level. For the two GCCS services in Figure 2, CDR_CR1 may be granted access to all of the services or JPlannerCR1 to all methods from either service that have levels either S, C, or U.

B. Constraint Definitions for Assurance

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

Definition 6: A signature constraint, SC, is a boolean expression defined on the signature of method \( M_{ijk} \) for some \( i = 1 \ldots m, j = 1 \ldots n_j, k = 1 \ldots q_{ij} \), of a service \( S_j \) of a resource \( R_i \), to limit the allowable values on the parameters, \( P_{ijk} \), and the return type, \( RT_{ijk} \). The boolean expression has the form: (return-type constraint) and (parameters constraint) where either or both constraints can be null. The return-type constraint limits the return value of the method call, \( RT_{ijk} \). The parameters constraint is a boolean expression with operators: AND, OR, and NOT.

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

Second, a time constraint limits the execution based on when a user (playing a role) can execute a method.

Definition 7: A time constraint, TC, is defined to represent a discrete period of time (i.e., days or time period in GMT) when a method can be invoked. The TC can involve relational expressions combined using: AND, OR. The TC can also have the values: always (unconstrained by time - the default value) and never (prohibited at any time). We define: \( TC = \{ e | e = "never" \ or \ e = "always" \ or \ e = \text{boolean expression} \} \).

In Figure 3, the JPlannerCR1 has a TC on ArmyBattleCommandSys and MarineCombatOpsSys \( (10\text{dec}00 < \text{date} < 16\text{feb}01) \). 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: \((\text{CrisisNum} = \text{CR1})\) and TC: \( (10\text{dec}00 < \text{date} < 16\text{feb}01) \).

Joint Service with Methods: a.k.a.

<table>
<thead>
<tr>
<th>Role</th>
<th>a.k.a.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S)Weather (Token)</td>
<td>METOC</td>
</tr>
<tr>
<td>(S)VideoTeleconference (Token, fromOrg, toOrg)</td>
<td>TLF</td>
</tr>
<tr>
<td>(S)JointOperationsPlanning (Token, CrisisNum)</td>
<td>JOPIES</td>
</tr>
<tr>
<td>(S)CrisisPicture (Token, CrisisNum, Grid1, Grid2)</td>
<td>COP</td>
</tr>
<tr>
<td>(S)TransportationFlow (Token)</td>
<td>JFAST</td>
</tr>
<tr>
<td>(S)LogisticsPlanningTool (Token, CrisisNum)</td>
<td>LOGSAFE</td>
</tr>
<tr>
<td>(S)DefenceMessageSystem (Token)</td>
<td>DMS</td>
</tr>
<tr>
<td>(T)NATO_MessageSystem (Token)</td>
<td>CRONOS</td>
</tr>
</tbody>
</table>

Component Service with Methods:

<table>
<thead>
<tr>
<th>Role</th>
<th>a.k.a.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S)ArmyBattleCommandSys (Token, CrisisNum)</td>
<td>ABCS</td>
</tr>
<tr>
<td>(S)AirForceBattleManagementSys (Token, CrisisNum)</td>
<td>TBMCS</td>
</tr>
<tr>
<td>(S)MarineCombatOpsSys (Token, CrisisNum)</td>
<td>TCO</td>
</tr>
<tr>
<td>(S)NavyCommandSystem (Token, CrisisNum)</td>
<td>JMCIS</td>
</tr>
</tbody>
</table>

Note: Access Classification Precedes Each Entry.

Figure 2. A GCCS Resource with Two Services.
The third type of constraint is for MAC to support CLR and CLS. As discussed in Section III.A, the CLS level is assigned to individual methods that comprise each service of a resource (see Figure 2). By enforcing the relationship among CLR and CLS at design and runtime, it is possible to realize the key aspect of MAC and BLM.

**Definition 8:** A mandatory access control constraint, MACC, is the fixed expression CLR ⩾ CLS that defines the valid relationship between CLR and CLS, for assigning a method to a role (design time) and for invoking a method by a user playing a role (run time).

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. The first step by the security officer using MAC is to establish a CLS for each method of each service for the resource, using U, C, S, or T (see Figure 2). Then, the CLS of the resource is inferred from the minimum service CLS, which is inferred from the minimum method CLS, automatically at design time using the tools for managing the security policy (see Section IV). 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 security privileges are constantly changing, this run-time check is required; 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.

**C. Privilege Definition and Authentication**

Once classifications have been established, the privileges for a user role can be defined, as indicated below:

**Definition 9:** Assume that a distributed application consists of resources, services, and methods: 

\[ \{ R_i, S_j, M_{ik} \mid i = 1 \ldots m, j = 1 \ldots n, k = 1 \ldots q_y \} \]  

A security privilege tuple, SPT, is defined as the resource, service, and/or method (with optional SC) that has been authorized (granted) to a UR with an optional TC: 

\[ \text{SPT} = \{ UR, TC, R_i, S_j, M_{ik}, SC_{ik} \} \]  

A security privilege tuple set, \( \rho \), contains all tuples for UR: 

\[ \text{UR}_\rho \text{URName} = \{ \{ UR, TC, R_i, S_j, M_{ik}, SC_{ik} \} \} \text{ for all } i, j, \text{ and } k. \]

Under this notation, a security officer can grant a UR access to all of a resource's services and their methods via the tuple \([ UR, TC, R_i, *, [*] \) Alternatively, a UR can be granted access to all methods of a service, by the tuple \([ UR, TC, R_i, S_j, [*] \). A user cannot be assigned an entire resource (or service) if the CLR does not match or exceed the CLS of all services (or methods).

The next step authorizes a UR to an end user, leading to:

**Definition 10:** A user, U, is uniquely identified (\( U_{UserId} \)) and authorized to play one or more URs in an application. Each user is assigned a CLR \( U^{CLR} \in \{ U, C, S, T \} \) chosen from the set \{U, C, S, T\}, where unclassified is the default (no MAC). In summary, \( U = \{ U_{UserId}, U^{CLR}, U^{UR} \} \), where \( U^{UR} \) represents the UR (see Definition 5) currently being played by the user \( U_{UserId} \).

**Definition 11:** A client, C, represents an authorized user U, and is uniquely identified for each session via a system generated token composed of: \([ \text{User-Id}, \text{UR_Name}, \text{IP-Address}, \text{Token-Creation-Time} \].

To illustrate the concepts, in Figure 3, there are URs, CDR_CR1, JPlannerCR1, and ArmyLogCR1, and their respective privilege tuple sets defined against GCCS. Figure 3 also contains users (General DoBest, Colonel DoGood, and Major DoRight) with CLR levels authorized to play different roles and a sample client token.

**IV. INFORMATION ASSURANCE ENFORCEMENT**

The second focus of the paper details the ability of our security model and enforcement framework to attain a degree of information assurance, insuring consistency of user roles, CLR/CLS levels, and end-user authorizations, so that their creation, modification, and deletion will always maintain the required RBAC/MAC policy. The key issue is to provide a set of techniques and tools that allow RBAC/MAC policies to be analyzed and assured at all times during design, development, and maintenance of distributed applications. In general, RBAC/MAC policies are application dependent and data security requirements vary widely from application to application. For example, Special Forces troop movement data must be both protected from use, while being almost instantaneously available to platoon leaders in the event that the forces are entering the same geographic area. In addition, up-to-date operational planning data must be readable to supply personnel to coordinate troop movement and sustainment. In this case, the security policies aren't life-critical, but are important from an operational perspective.

In order for an organization's management personnel and security officer to take full advantage of RBAC/MAC, there must be administrative and management tools to accurately and precisely specify an application's security requirements. These tools should automatically alert the security officer when potential conflicts occur during the creation or modification of roles, classification levels, and end-user authorizations, thereby heading off possible inconsistencies, and provide on-demand analyses, allowing officers to gauge their realized software and/or security requirements against their specifications. The overriding intent is to finish with a distributed application with a strong confidence in the attainment of the RBAC/MAC policy. In Section IV.A, we overview the
enforcement framework for the security model presented in Section III. Then, we investigate assurance for roles, CLR/CLS levels, and end-user authorizations, respectively, in Sections IV.B, IV.C, and IV.D.

A. Overview of Enforcement Framework

The processing required by a client joining the distributed environment and attempting to access resources is given in Figure 4. In Steps 1 to 4, the client is authenticated. In Step 5, the client selects a role to play for the session. In Steps 6 to 8, a token is generated and assigned to the user for the session via a name and password verification of the Unified Security Resource (USR). USR is a set of middleware security resources (JINI and CORBA) that manage all MAC and RBAC meta-data for users, user roles, and resources [20]. The user chooses a role and registers via the RegisterClient method, which requests a global time from the global clock resource and returns a token via CreateToken. In Step 9, the client discovers the desired method from the lookup service (JINI or CORBA), and attempts to invoke the method with its parameters and the Token. In Step 10, the resource uses the hasClientRight method (Step 11) to check whether the user/client meets all of the MACC, time, and signature constraints required to invoke the method (Step 12).

![Figure 4. Client Interactions and Service Invocations.](image)

In summary, in order to obtain access to a service or method, there are a series of checks as shown in Figure 5:

Client Authentication with Token Check on the validity of the token and IP Constraint Check to verify if the user is logged on from a permitted location; and Client Authorization with a MACC Check to verify that the user/client has the required CLR to access the method, a TC Check to verify if the access is within the allowable time period, and a SC Check to verify that the invocation satisfies the signature constraint. Collectively, all of the checks illustrated in Figures 4 and 5 provide run-time assurance of the RBAC/MAC policy as a client (with a UR and CLR) invokes methods (with CLS) of resources.

B. Consistency for User Roles

When a security officer is creating and modifying URs for a distributed application, consistency is critical to maintaining the RBAC policy. This is a time-oriented issue; changes to the policy are needed when evolution and extensibility are the norm. Regardless of the changes that are made, there must be assurance that the privileges of each UR are adequate to satisfy its functions, while not exceeding the capabilities of the UR, to insure that misuse and corruption do not occur. In addition, since URs are often interdependent on one another, it may be necessary to examine their interactions to insure that privileges aren't passed inadvertently from role to role.

There are many different scenarios of evolution that must be handled. In the base case, a security officer may create new roles for a group of potential users or may create specific roles that are targeted for a particular end-user for a special assignment under a special circumstance. Each newly created or modified role must be internally consistent so that no conflicts occur within the role itself. For the distributed case, when privileges are assigned to each role, this assignment implicitly grants access privileges to the role holder (end-user). Such an assignment process utilizes the least-privilege principle. This policy is intentionally conservative, requiring that the RBAC policy be validated by the software officer, security officer, or both. Our approach separates security responsibilities between policy and authorizations. Policy is set by the organization or resource and guides the authorizations. Figure 6 depicts the Security Policy Client (SPC), which manages URs by granting/revoking privileges (TCs, methods, SCs) and setting a CLS level.

To complement the least-privilege principle, URs often must satisfy mutual exclusion conditions. Here, there must be a careful balance between permitting access to certain methods while prohibiting access to other methods. Mutual exclusion is a strong RBAC concern, and may be dictated by rules, regulations, or law. For example, an individual overseeing the activities of a military crisis may be explicitly prohibited by the military command hierarchy from making certain changes. This strong mutual exclusion is observed by the military chain of command, and mandated by its regulations. A role may be built with a specific user in mind, but policy governs
the privileges of the UR. Figures 7 and 8 demonstrate the assignment of resources and methods to roles. Note that the security system will not allow assignment of a method to a role that has a lesser CLS level than the method (Figure 8); a design-time assurance check.

U.S. government use of classified information is dictated by executive orders from the President of the United States and DoD. In classified or non-classified multilevel security systems, the CLR of the user and the CLS of the information objects must be clearly defined in order for security policy to be established and enforced. In a coalition, inconsistent CLS levels need to be normalized so that coalition information assurance can be established based on required security policy. Figure 9 depicts the Security Authorization Client (SAC) which is utilized to assign CLRs and authorize roles to end users. Like the SPC in Figure 8, this tool provides design-time assurance checks at to verify consistency of CLS levels.

D. Consistency in End-User Authorization

For consistency in authorization, there must be the ability to inspect the intra and inter-role consistency. Figure 9 is an example of information available via SAC (see Figure 8) for the security officer to review user authorizations. In addition to these capabilities, Figure 10 is an example of one output of the Security Analysis Tool (SAT) that dynamically tracks all client activity, including logons and method invocations. This assurance tool will not prevent an unwanted action, but can alert a security officer of the violation to hold a user accountable, or assist in recovery.

C. Consistency of Classification Levels

A multi-level security system has two or more CLS levels, with not all users authorized to access all data [27], and requires additional logic to recognize and enforce the security policy. In this situation, the consistency of these CLS levels is very important. Some organizations may consider grades, salary, or social security numbers as sensitive and limit access to that type of information.

Figure 6. Policy Client: Creating Role with Classification.

Figure 7. Policy Client: Assigning a Resource to Role.

Figure 8. Policy Client: Assignment of Method to Role.

Figure 9. Authorization Client: Create User and Clearance.

Figure 10. Security Access History from SAC.
V. CONCLUSIONS AND FUTURE WORK

In this paper, we have examined the unification of MAC and RBAC into a security model and enforcement framework for distributed applications towards the support of information assurance. To facilitate the discussion, in Section II, we introduced the Dynamic Coalition Problem and its risks to information assurance that arise in a crisis. In Section III, we presented a unified constraint-based model which extended our prior work on RBAC [20] to include MAC to ensure that the CLR of users playing roles met or exceeded CLS of the resources, services, and methods being utilized. The RBAC/MAC security model is supported by an enforcement framework (Section IV) with security administrative tools (design time) and a security infrastructure (run time) for information assurance of user roles, classifications, and end-user authorizations. Overall, we believe this work provides a strong foundation towards information assurance of RBAC/MAC in a distributed setting.

Ongoing research efforts are in a number of different areas. First, there is a project involving the ability of users to delegate roles to other users, which is an important policy/assurance issue, since a user can be given authority to delegate roles without intervention. Second, there is ongoing research in extending the model with user constraints, which in turn leads to a third area, namely, role deconfliction, which involves both consistency constraints and mutual exclusion. Long-term we are interested in investigating issues involving the performance of our assurance approach. Finally, the prototyping of the enforcement framework is ongoing (see http://www.engr.uconn.edu/~steve/DSEC/dsec.html).

REFERENCES