Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management

Serban I. Gavrila
VDG Inc.
6009 Brookside Drive
Chevy Chase, MD 20815
gavrila@csmes.ncsl.nist.gov

John F. Barkley
National Institute of Standards and Technology
Gaithersburg, MD 20899
jbarkley@nist.gov

Abstract

Role Based Access Control (RBAC), an access control mechanism, reduces the cost of administering access control policies as well as making the process less error-prone. The Admin Tool developed for the NIST RBAC Model manages user/role and role/role relationships stored in the RBAC Database. This paper presents a formal specification of the RBAC Database and Admin Tool operations. Consistency requirements for the RBAC Database are defined as a set of properties. Alternative properties, substantially simpler to verify in an implementation, are shown to be equivalent. In addition, the paper defines the semantics of Admin Tool operations, and shows that, given a consistent RBAC Database and an operation which meets specified conditions, the RBAC Database remains consistent after the operation is performed.

1 Introduction

Role Based Access Control (RBAC) is an access control mechanism that reduces the cost of administering access control policies, as well as making the process less error-prone. The NIST RBAC Model [1] supports complex access control policies while also permitting efficient implementation.

The Admin Tool developed for the NIST RBAC Model manages user/role and role/role relationships. These relationships are stored in the RBAC Database. In order to maintain the integrity of the information in the RBAC Database, a set of properties defining data consistency was developed. The properties initially developed can be simplified and reduced in number. The equivalent properties are verified before the Admin Tool permits any RBAC Database operation to be performed. This results in a more efficient implementation of the Admin Tool.

This paper describes the Admin Tool developed for the NIST RBAC Model. It presents a formal specification of the initial set of consistency properties for the RBAC Database consistency and the simplified set. These two sets of consistency properties are shown to be equivalent.

In addition, the paper presents a formal specification of RBAC Database operations. It is shown that, given a consistent RBAC Database, database operations which meet given conditions maintain database consistency.

The Admin Tool described in this paper is part of three implementations of the NIST RBAC Model: one for the World Wide Web (RBAC/Web) [2], one for use in relational database environments, where the RBAC Database is implemented by tables in a commercial DBMS, and one for Windows NT.[1] The Windows NT implementation does not support Dynamic Separation of Duties.

Section 1 of the paper is this Introduction. Section 2 describes the NIST RBAC Model and Admin Tool. Section 3 presents the formal specification of RBAC Database consistency properties, operations, and conditions under which database operations preserve database consistency. Section 4 summarizes.

2 NIST RBAC Model and Admin Tool

The NIST RBAC Model is a Sandhu $RBAC_3$ Model [3]. It extends the basic RBAC Model by adding role hierarchies, role cardinality, and conflict of interest relationships. The Role hierarchy is a partial ordering on the set of roles. If one role inherits another and a user is authorized for the inheriting role, then that user is also authorized for the inherited role. Role cardinality is a role attribute which restricts the number of users for which a role may be authorized.

---

[1] Because of the nature of this report, it is necessary to mention vendors and commercial products. The presence or absence of a particular trade name product does not imply criticism or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available.

There are two types of conflict of interest relationships: Static Separation of Duties (ssd) and Dynamic Separation of Duties (dsd). If two roles have an ssd relationship, then no user may be authorized for both roles. If two roles have a dsd relationship, then a user may be authorized for both roles, but that user may not have both roles active at the same time (in the same session or different sessions).

The Admin Tool manages user authorization for roles, role hierarchies, ssd, dsd, and role cardinality. In order to reduce errors in administration, the Admin Tool makes use of the concept of role *assignment*. A role gets assigned to a user explicitly through the Admin Tool. A role is *authorized* for a user if that role is assigned to the user or is inherited by a role which has been assigned to the user. This concept of role assignment helps an administrator maintain awareness of the role hierarchies which describe an organization. The Admin Tool does not permit a role, which is inherited by a role already assigned to a user, to be assigned to that user. This design decision is reflected in rule P3 of Section 3.4.

Another design decision is required to address the problem illustrated by the following example. Suppose role r_1 and role r_2 have a dsd relationship, r_1 is authorized for user u, and r_1 inherits r_2. When establishing u's active role set for a session, the following apparent contradiction results. Role r_2 belongs in u's active role set because r_1 inherits r_2, but r_2 cannot be in u's active role set because r_1 and r_2 have a dsd relationship.

In order to address this problem, the Admin Tool does not permit a role pair to simultaneously have both a hierarchical and a dsd relationship. Thus, the apparent contradiction in active role set contents can never occur. This design decision is based on the desire to have all role relationships specified in the RBAC Database hold at all times and in all situations. The goal is to simplify the task of administration. Administrators are not required to be aware of situation-sensitive rules. They are able to know that what is reflected in the RBAC Database holds throughout the administration, session establishment, and enforcement of role relationships and access. This design decision is reflected in rules P15, P16, and P17 of Section 3.4.

It is recognized that alternative approaches may be equally valid depending on implementation requirements. One such alternative approach is described by Sandhu [4].

Figure 1 shows the Admin Tool's graphical display for a hypothetical policy in a bank. In the bank there are roles such as *branch_manager*, *teller*, and *account_holder*. The display shows:

- The bank's role hierarchy. For example, the financial advisor is a special kind of account representative authorized to market non-insured bank products. The financial advisor, account representative, branch manager, internal auditor, and teller are all bank employees.

- The number of users to which a role is authorized and the cardinality for each role. For example, no users are currently authorized for the role *invited_guest* and an unlimited number of users may be authorized for that role.

- The conflict of interest relationships for a selected role *teller*. The red (in a color display) pentagon indicates that *teller* and *internal_auditor* have a ssd relationship, and the blue (in a color display) rectangles indicate that the roles *financial_advisor*, *account_rep*, and *account_holder* each have a dsd relationship with *teller*.

Figure 2 shows the main display of the Admin Tool. With this display, user/role and role/role relationships are managed. Using the left panel of the display, users are created, deleted, and their role assignments managed and displayed. The left panel shows that user *ko* has role assignments *account_holder* and *teller* indicating that *ko* is employed as a teller and has an account in the bank where employed. User *ko* may also be assigned the roles *account_rep*, *branch_manager*, *financial_advisor*, and *invited_guest*. User *ko* may not be assigned the roles *employee*, *internal_auditor*, *role_admin*, and *visitor* for the reasons indicated.
Using the right panel of the display in Fig. 2, roles are created and deleted, and role hierarchies, role cardinality, and conflict of interest relationships between roles (ssd, dsd) are defined. Mirroring the graphical display of Fig. 1, the right panel shows that the role *teller* is currently selected. The role *teller* can now be deleted, or its cardinality, position in a hierarchy, or conflict of interest relationships with other roles modified.

3 RBAC Database Consistency

3.1 Basic Sets and Functions

USERS: is the set of users.

ROLES: is the set of roles.

OPERATIONS = {*addUser*, *rmUser*, *addRole*, *rmRole*, *addAssignment*, *rmAssignment*, *addInheritance*, *rmInheritance*, *addSsd*, *rmSsd*, *addDsd*, *rmDsd*, *setCardinality*, *addActiveRoles*, *rmActiveRoles*}. This set contains administrative operations, such as add a user, remove a user, etc., as well as operations to add/remove active roles, which may be initiated by users.

assigned_roles: $USERS \rightarrow 2^{ROLES}$. *assigned_roles*($u$) denotes the set of roles assigned to user u.

active_roles: $USERS \rightarrow 2^{ROLES}$. *active_roles*($u$) denotes the set of active (currently assumed) roles of user u in his sessions.

inherits $\subseteq ROLES \times ROLES$ is the inheritance relation between roles. If $(r_1, r_2) \in inherits$, we write also $r_1 \rightarrow r_2$. We denote the transitive closure, respectively transitive and reflexive closure of the $\rightarrow$ (*inherits*) relation by $\rightarrow^+$, respectively $\rightarrow^*$.

ssd $\subseteq ROLES \times ROLES$ is the static separation of duties relation between roles.

dsd $\subseteq ROLES \times ROLES$ is the dynamic separation of duties relation between roles.

cardinality: $ROLES \rightarrow \mathbf{N} \cup \{\infty\}$. *cardinality*($r$) denotes the cardinality of role r, i.e., the maximum number of users authorized for that role.[2]

3.2 Derived Functions

authorized_roles: $USERS \rightarrow 2^{ROLES}$ returns the roles authorized for a given user. We say that a role r is authorized for a user u if either r is assigned to u, or r is inherited by another role that is assigned to u. For example, if John is assigned the teller role, and teller inherits ("is a") employee, then both the teller and employee roles are authorized for John. Formally:

$\forall r \in ROLES, \forall u \in USERS, r \in authorized\_roles(u) \Leftrightarrow$
$\quad \exists p \in ROLES$ such that $p \in assigned\_roles(u) \wedge p \rightarrow^* r$.

authorized_users: $ROLES \rightarrow 2^{USERS}$ returns the users authorized for a given role. This is simply a convenience function; a user is authorized for a role if the role is authorized for that user. Formally:

$\forall r \in ROLES, \forall u \in USERS, u \in authorized\_users(r) \Leftrightarrow$
$\quad r \in authorized\_roles(u)$.

[2] In the context of the Admin Tool implementation, the symbol "∞" means that the number of roles which can be authorized to users by means of the Admin Tool is unlimited, i.e., will not be checked by the Tool. The other integer values are, of course, limited by the implementation's environment, i.e., the size of the integer used to contain the value of a role's cardinality.

3.3 States and Transitions

A state is a tuple
(*USERS*, *ROLES*, *assigned_roles*, *active_roles*, *inherits*, *ssd*, *dsd*, *cardinality*).

We denote the set of states by *STATES*.

The state transitions are triggered by administrators performing administrative operations and by users assuming or dropping roles during their RBAC sessions. Each operation needs one or more arguments, that we leave unspecified for the time being, but we denote their set by *ARGS*. The transition function is a partial function:

$$\delta: STATES \times OPERATIONS \times 2^{ARGS} \rightarrow STATES,$$

such that $\delta(s, op, args) = s'$ if and only if the RBAC Database goes from state s to state s' by performing operation op with arguments $args$ on the sets and functions defined above.

3.4 RBAC Database Consistency Rules

During system operation, we require each database state to satisfy the following properties:

P1. The number of authorized users for any role does not exceed the cardinality of that role. Formally:
$\quad \forall r \in ROLES, |authorized\_users(r)| \leq cardinality(r)$.

P2. No role inherits (directly or indirectly) itself. Formally:
$\quad \forall r \in ROLES, \neg(r \rightarrow^+ r)$.

P3. Any two roles assigned to same user do not inherit (directly or indirectly) one another. Formally:
$\forall u \in USERS, \forall r_1, r_2 \in ROLES, r_1, r_2 \in assigned\_roles(u) \Rightarrow$
$\quad \neg(r_1 \rightarrow^+ r_2)$.

P4. Any two roles authorized for same user are not in static separation of duties. Formally:
$\forall u \in USERS, \forall r_1, r_2 \in ROLES, r_1, r_2 \in authorized\_roles(u) \Rightarrow$
$\quad (r_1, r_2) \notin ssd$.

P5. There is no role in static separation of duties with itself. Formally:
$\quad \forall r \in ROLES \Rightarrow (r, r) \notin ssd$.

P6. The static separation of duties relation is symmetric. Formally:
$\quad \forall r_1, r_2 \in ROLES, (r_1, r_2) \in ssd \Rightarrow (r_2, r_1) \in ssd$.

P7. Any two roles in static separation of duties do not inherit (directly or indirectly) one another. Formally:

$\forall r_1, r_2 \in ROLES, r_1 \rightarrow^+ r_2 \Rightarrow (r_1, r_2) \notin ssd.$

P8. There is no role inheriting (directly or indirectly) two roles in static separation of duties. Formally:

$\forall r, r_1, r_2 \in ROLES, r \rightarrow^+ r_1, r \rightarrow^+ r_2 \Rightarrow (r_1, r_2) \notin ssd.$

P9. If a role inherits (directly or indirectly) another role and that role is in static separation of duties with a third role, then the inheriting role is in static separation of duties with the third one. Formally:

$\forall r, r_1, r_2 \in ROLES, r \rightarrow^+ r_1, (r_1, r_2) \in ssd \Rightarrow (r, r_2) \in ssd.$

P10. The active role set of any user is a subset of his or her authorized roles. Formally:

$\forall u \in USERS, active\_roles(u) \subseteq authorized\_roles(u).$

P11. Any two roles in dynamic separation of duties do not belong both to the active role set of any user. Formally:

$\forall u \in USERS, \forall r_1, r_2 \in ROLES, r_1, r_2 \in active\_roles(u) \Rightarrow (r_1, r_2) \notin dsd.$

P12. The dynamic separation of duties and static separation of duties relations are disjoint. Formally:

$\forall r_1, r_2 \in ROLES, (r_1, r_2) \in dsd \Rightarrow (r_1, r_2) \notin ssd.$

P13. There is no role in dynamic separation of duties with itself. Formally:

$\forall r \in ROLES \Rightarrow (r, r) \notin dsd.$

P14. The dynamic separation of duties relation is symmetric. Formally:

$\forall r_1, r_2 \in ROLES, (r_1, r_2) \in dsd \Rightarrow (r_2, r_1) \in dsd.$

P15. Any two roles in dynamic separation of duties do not inherit (directly or indirectly) one another. Formally:

$\forall r_1, r_2 \in ROLES, r_1 \rightarrow^+ r_2 \Rightarrow (r_1, r_2) \notin dsd.$

P16. There is no role inheriting (directly or indirectly) two roles in dynamic separation of duties. Formally:

$\forall r, r_1, r_2 \in ROLES, r \rightarrow^+ r_1, r \rightarrow^+ r_2 \Rightarrow (r_1, r_2) \notin dsd.$

P17. If a role inherits (directly or indirectly) another role and that role is in dynamic separation of duties with a third role, then the inheriting role is in dynamic separation of duties with the third one. Formally:

$\forall r, r_1, r_2 \in ROLES, r \rightarrow^+ r_1, (r_1, r_2) \in dsd \Rightarrow (r, r_2) \in dsd.$

Definition 1. We say that the RBAC Database is consistent in a state if and only if properties P1-P17 hold in that state.□

We successively try to substitute simpler but equivalent properties for some of the consistency properties defined above. First we show that property P9 can be substituted by a similar one that only involves direct inheritance, and is defined below.

P18. $\forall r, r_1, r_2 \in ROLES, r \rightarrow r_1, (r_1, r_2) \in ssd \Rightarrow (r, r_2) \in ssd.$

Theorem 1. P9 ⇔ P18.

Proof. P18 is a particular case of P9, hence P9 ⇒ P18. Now assume that P18 holds. We show the property $r \rightarrow^+ r_1 \wedge (r_1, r_2) \in ssd \Rightarrow (r, r_2) \in ssd$ by induction on the number of steps in the inheritance $r \rightarrow^+ r_1$. For one step (direct inheritance), $(r, r_2) \in ssd$ follows directly from P18. Assume the property holds for any role r_1 and any number of steps $\leq n$ (where $n \geq 1$), and let $r \rightarrow^+ r_1$ in n+1 steps and $(r_1, r_2) \in ssd$. There exists a role r' such that $r \rightarrow r'$ and $r' \rightarrow^+ r_1$ in n steps. Then $(r', r_2) \in ssd$ by the induction hypothesis, and $(r, r_2) \in ssd$ by P18.□

Property P17 can be substituted by a similar one that only involves direct inheritance:

P19. $\forall r, r_1, r_2 \in ROLES, r \rightarrow r_1, (r_1, r_2) \in dsd \Rightarrow (r, r_2) \in dsd.$

Theorem 2. P17 ⇔ P19.

Proof. P19 is a particular case of P17, hence P17 ⇒ P19. Now assume that P19 holds. We show the property $r \rightarrow^+ r_1 \wedge (r_1, r_2) \in dsd \Rightarrow (r, r_2) \in dsd$ by induction on the number of steps in the inheritance $r \rightarrow^+ r_1$. For one step (direct inheritance), $(r, r_2) \in dsd$ follows directly from P19. Assume the property holds for any role r_1 and any number of steps $\leq n$ (where $n \geq 1$), and let $r \rightarrow^+ r_1$ in n+1 steps and $(r_1, r_2) \in dsd$. There exists a role r' such that $r \rightarrow r'$ and $r' \rightarrow^+ r_1$ in n steps. Then $(r', r_2) \in dsd$ by the induction hypothesis, and $(r, r_2) \in dsd$ by P19.□

The following two theorems show that not all properties P1-P17 are independent. Consequently, some of them can be omitted from the consistency requirements.

Theorem 3. P5∧P6∧P9 ⇒ P7∧P8.

Proof. Assume that P5, P6, P9 hold, and let us prove P7. Assume that $r_1 \rightarrow^+ r_2$, and, by way of contradiction, that $(r_1, r_2) \in ssd$. P6 implies that $(r_2, r_1) \in ssd$. P9 applied to $r_1 \rightarrow^+ r_2$ and $(r_2, r_1) \in ssd$ results in $(r_1, r_1) \in ssd$, which contradicts P5.

Let us prove P8. Assume that $r \rightarrow^+ r_1, r \rightarrow^+ r_2$, and, by way of contradiction, that $(r_1, r_2) \in ssd$. P9 applied to $r \rightarrow^+ r_1$ and $(r_1, r_2) \in ssd$ gives $(r, r_2) \in ssd$, or $(r_2, r) \in ssd$ by P6. P9 applied again to $r \rightarrow^+ r_2$ and $(r_2, r) \in ssd$ gives $(r, r) \in ssd$, which contradicts P5.□

Theorem 4. P13$\wedge$P14$\wedge$P17 $\Rightarrow$ P15$\wedge$P16.

Proof. Assume that P13, P14, P17 hold, and let us prove P15. Assume that $r_1 \rightarrow^+ r_2$, and, by way of contradiction, that $(r_1, r_2) \in dsd$. P14 implies that $(r_2, r_1) \in dsd$. P17 applied to $r_1 \rightarrow^+ r_2$ and $(r_2, r_1) \in dsd$ gives $(r_1, r_1) \in dsd$, which contradicts P13.

Let us prove P16. Assume that $r \rightarrow^+ r_1$, $r \rightarrow^+ r_2$, and, by way of contradiction, that $(r_1, r_2) \in dsd$. P17 applied to $r \rightarrow^+ r_1$ and $(r_1, r_2) \in dsd$ gives $(r, r_2) \in dsd$, or $(r_2, r) \in dsd$ by P14. P17 applied again to $r \rightarrow^+ r_2$ and $(r_2, r) \in dsd$ gives $(r, r) \in dsd$, which contradicts P13.□

In conditions already satisfied in a consistent state, property P4 can be relaxed to P20, defined below, which forbids only roles assigned to same user to be in static separation of duties.

P20.$\forall u \in USERS, \forall r_1, r_2 \in ROLES$, $r_1, r_2 \in assigned\_roles(u)$
$\Rightarrow (r_1, r_2) \notin ssd$.

Theorem 5. In any state such that P6$\wedge$P9 holds, P4 $\Leftrightarrow$ P20.

Proof. Assume that P6$\wedge$P9 holds. P20 is a particular case of P4, hence P4$\Rightarrow$P20. Assume that P20 holds, and let us prove P4. Let $r_1, r_2 \in authorized\_roles(u)$, and assume by way of contradiction that $(r_1, r_2) \in ssd$. There exist roles p_1, $p_2 \in assigned\_roles(u)$, such that $p_1 \rightarrow^* r_1$, $p_2 \rightarrow^* r_2$.

If $p_1 = r_1$, then $(p_1, r_2) \in ssd$. If $p_1 \neq r_1$, then $p_1 \rightarrow^+ r_1$, $(r_1, r_2) \in ssd$, and P9 implies $(p_1, r_2) \in ssd$. Anyway, $(r_2, p_1) \in ssd$ by P6.
If $p_2 = r_2$, then $(p_2, p_1) \in ssd$. If $p_2 \neq r_2$, then $p_2 \rightarrow^+ r_2$, $(r_2, p_1) \in ssd$, and P9 implies $(p_2, p_1) \in ssd$. But p_2, p_1 belong to $assigned\_roles(u)$, and by P20 $(p_2, p_1) \notin ssd$, contradiction.□

Now we can establish a set of consistency conditions, equivalent to, but fewer and simpler to verify than the original set.

Theorem 6. The RBAC Database is consistent if and only if the following properties hold:

P1. $\forall r \in ROLES$, $|authorized\_users(r)| \leq cardinality(r)$.
P2. $\forall r \in ROLES$, $\neg(r \rightarrow^+ r)$.
P3.$\forall u \in USERS, \forall r_1, r_2 \in ROLES$, $r_1, r_2 \in assigned\_roles(u)$
$\Rightarrow \neg(r_1 \rightarrow^+ r_2)$.
P20.$\forall u \in USERS, \forall r_1, r_2 \in ROLES$, $r_1, r_2 \in assigned\_roles(u)$
$\Rightarrow (r_1, r_2) \notin ssd$.
P5. $\forall r \in ROLES \Rightarrow (r, r) \notin ssd$.
P6. $\forall r_1, r_2 \in ROLES$, $(r_1, r_2) \in ssd \Rightarrow (r_2, r_1) \in ssd$.
P18. $\forall r, r_1, r_2 \in ROLES$, $r \rightarrow r_1$, $(r_1, r_2) \in ssd \Rightarrow (r, r_2) \in ssd$.
P10. $\forall u \in USERS$, $active\_roles(u) \subseteq authorized\_roles(u)$.
P11.$\forall u \in USERS$, $\forall r_1, r_2 \in ROLES$, $r_1, r_2 \in active\_roles(u)$
$\Rightarrow (r_1, r_2) \notin dsd$.
P12. $\forall r_1, r_2 \in ROLES$, $(r_1, r_2) \in dsd \Rightarrow (r_1, r_2) \notin ssd$.
P13. $\forall r \in ROLES \Rightarrow (r, r) \notin dsd$.
P14. $\forall r_1, r_2 \in ROLES$, $(r_1, r_2) \in dsd \Rightarrow (r_2, r_1) \in dsd$.
P19. $\forall r, r_1, r_2 \in ROLES$, $r \rightarrow r_1$, $(r_1, r_2) \in dsd \Rightarrow (r, r_2) \in dsd$.

Proof. The proof is a simple exercise of predicate calculus using the results of Theorems 1-5.□

3.5 Operations

This section shows under what conditions each operation in *OPERATIONS* preserves the database consistency. Specifically, we show that if the database is in a consistent state, and a certain set of conditions is satisfied by (the arguments of) an operation, then the database remains in a consistent state after that operation is performed.

For each operation in *OPERATIONS*, we specify its arguments, semantics, and consistency preserving conditions. In the semantics specification of a database operation, a primed variable denotes its value after that operation has been performed.

adduser
Arguments:
    user
Semantics:
    $USERS' = USERS \cup \{user\}$
    $active\_roles' = active\_roles \cup \{user \mapsto \emptyset\}$
    $assigned\_roles' = assigned\_roles \cup \{user \mapsto \emptyset\}$
Conditions:
    C11: $user \notin USERS$

The new user is added to the *USERS* data set; its active and assigned roles are set to empty.

rmUser
Arguments:
    user
Semantics:
    $USERS' = USERS \setminus \{user\}$
    $active\_roles' = active\_roles \setminus \{user \mapsto active\_roles(user)\}$
    $assigned\_roles' = assigned\_roles \setminus \{user \mapsto assigned\_roles(user)\}$
Conditions:
    C21. $user \in USERS$
    C22. $assigned\_roles(user) = \emptyset$

The user is deleted from the *USERS* data set; its entries in the active roles and assigned roles are removed. Note that condition C22 corroborated with P10 implies that *user* has no active roles.

addrole
Arguments:
  role
Semantics:
  $ROLES' = ROLES \cup \{role\}$
  $cardinality' = cardinality \cup \{role \mapsto \infty\}$
Conditions:
  C31: $role \notin ROLES$

The new role is added to the *ROLES* data set. By default, the roles receives an infinite cardinality.

rmRole
Arguments:
  role
Semantics:
  $ROLES' = ROLES \setminus \{role\}$
  $cardinality' = cardinality \setminus \{role \mapsto cardinality(role)\}$
Conditions:
  C41. $role \in ROLES$
  C42. $\forall u \in USERS, role \notin assigned\_roles(u)$
  C43. $\forall p \in ROLES, \neg(p \rightarrow role) \wedge \neg(role \rightarrow p)$
  C44. $\forall p \in ROLES, (p, role) \notin ssd$
  C45. $\forall p \in ROLES, (p, role) \notin dsd$

The role is removed from the *ROLES* data set; its entry in the cardinality is also removed. The role may be removed only if it is not assigned to any user (condition C42), it is not part of a role hierarchy (C43), and it is not in separation of duties relationships with other roles (C44, C45). Note that conditions C42, C43, corroborated with P10, imply that *role* is not active for any user.

addAssignment
Arguments:
  user, *role*
Semantics:
  $assigned\_roles' = (assigned\_roles \setminus \{user \mapsto assigned\_roles(user)\}) \cup \{user \mapsto assigned\_roles(user) \cup \{role\}\}$
Conditions:
  C51. $user \in USERS$
  C52. $role \in ROLES$
  C53. $role \notin authorized\_roles(user)$
  C54. $\forall p \in ROLES, role \rightarrow^{+} p \Rightarrow p \notin assigned\_roles(user)$
  C55. $\forall p \in ROLES, p \in assigned\_roles(user) \Rightarrow (p, role) \notin ssd$
  C56. $\forall p \in ROLES, role \rightarrow^{*} p \Rightarrow |authorized\_users(p)| < cardinality(p)$

The entry for *user* in *assigned_roles* is updated to reflect the new assignment. The role may not be already authorized for that user, or inherit another role assigned to that user, or be in static separation of duties with another role assigned to that user. Because the authorization propagates along the *inherits* relation, *role* and all roles inherited by it must satisfy the requirement related to the cardinality (C56).

rmAssignment
Arguments:
  user, *role*
Semantics:
  $assigned\_roles' = (assigned\_roles \setminus \{user \mapsto assigned\_roles(user)\}) \cup \{user \mapsto assigned\_roles(user) \setminus \{role\}\}$
Conditions:
  C61. $user \in USERS$
  C62. $role \in ROLES$
  C63. $role \in assigned\_roles(user)$
  C64. $\forall r \in ROLES, r \in active\_roles(user) \wedge role \rightarrow^{*} r \Rightarrow \exists p \in ROLES: p \neq role \wedge p \rightarrow^{+} r \wedge p \in assigned\_roles(user)$

The entry for *user* in *assigned_roles* is updated. Condition C64 forbids a role *r* to remain active for *user* after deassignment if *r* is no more authorized for *user*.

addInheritance
Arguments:
  $role_1$, $role_2$
Semantics:
  $inherits' = inherit \cup \{(role_1, role_2)\}$
Conditions:
  C71. $role_1, role_2 \in ROLES$
  C72. $role_1 \neq role_2$
  C73. $\neg(role_1 \rightarrow^{+} role_2) \wedge \neg(role_2 \rightarrow^{+} role_1)$
  C74. $\forall u \in USERS, \forall r \in ROLES, role_2 \rightarrow^{*} r, role_1 \in authorized\_roles(u) \Rightarrow r \notin assigned\_roles(u)$
  C75. $\forall r \in ROLES, (r, role_2) \in ssd \Rightarrow (r, role_1) \in ssd$
  C76. $\forall r \in ROLES, (r, role_2) \in dsd \Rightarrow (r, role_1) \in dsd$
  C77. $\forall r \in ROLES, role_2 \rightarrow^{*} r \Rightarrow |authorized\_users(role_1) \cup authorized\_users(r)| \leq cardinality(r)$

The pair $(role_1, role_2)$ is added to the *inherits* relation. The roles may not already be part of the hierarchy. Establishing the new inheritance must not result in a user being authorized to $role_1$ and assigned to another role inherited by $role_2$ (C74). The inherited role $role_2$ may not be in separation of duties with another role without the inheriting role $role_1$ being in separation of duty with that role. The new inheritance may increase the number of authorized users for some roles; condition C77 takes care of that.

rmInheritance

Arguments:

$role_1, role_2$

Semantics:

$inherits' = inherits \setminus \{(role_1, role_2)\}$

Conditions:

C81. $role_1, role_2 \in ROLES$

C82. $role_1 \rightarrow role_2$

C83. $\forall u \in USERS, \forall r \in ROLES,$
$u \in authorized\_users(role_1) \wedge$
$r \in active\_roles(u) \wedge$
$role_2 \rightarrow^* r \Rightarrow$
$\exists p \in ROLES: p \in assigned\_roles(u) \wedge$
$(p \rightarrow^* r$ without using $role_1 \rightarrow role_2)$

The inheritance $role_1 \rightarrow role_2$ is removed. Condition C83 forbids a role r to remain active for a user u after removing the inheritance $role_1 \rightarrow role_2$, if r is no more authorized for u.

addSsd

Arguments:

$role_1, role_2$

Semantics:

$ssd' = ssd \cup \{(role_1, role_2), (role_2, role_1)\}$

Conditions:

C91. $role_1, role_2 \in ROLES$

C92. $role_1 \neq role_2$

C93. $(role_1, role_2) \notin ssd$

C94. $(role_1, role_2) \notin dsd$

C95. $\forall r \in ROLES, r \rightarrow role_1 \Rightarrow (r, role_2) \in ssd$

C96. $\forall r \in ROLES, r \rightarrow role_2 \Rightarrow (r, role_1) \in ssd$

C97. $\forall u \in USERS, \{role_1, role_2\} \not\subseteq assigned\_roles(u)$

The *ssd* relation is updated. The two roles may not be in either separation of duties relation, and any role inheriting one of the arguments must be in ssd with the other. The arguments may not be both assigned to same user.

rmSsd

Arguments:

$role_1, role_2$

Semantics:

$ssd' = ssd \setminus \{(role_1, role_2), (role_2, role_1)\}$

Conditions:

C101. $role_1, role_2 \in ROLES$

C102. $(role_1, role_2) \in ssd$

C103. $\forall r \in ROLES, role_1 \rightarrow r \Rightarrow (r, role_2) \notin ssd$

C104. $\forall r \in ROLES, role_2 \rightarrow r \Rightarrow (r, role_1) \notin ssd$

The *ssd* relation is updated. After deleting the ssd relation between the two roles, no one of them may remain in ssd with a role inherited by the other.

addDsd

Arguments:

$role_1, role_2$

Semantics:

$dsd' = dsd \cup \{(role_1, role_2), (role_2, role_1)\}$

Conditions:

C111. $role_1, role_2 \in ROLES$

C112. $role_1 \neq role_2$

C113. $(role_1, role_2) \notin ssd$

C114. $(role_1, role_2) \notin dsd$

C115. $\forall r \in ROLES, r \rightarrow role_1 \Rightarrow (r, role_2) \in dsd$

C116. $\forall r \in ROLES, r \rightarrow role_2 \Rightarrow (r, role_1) \in dsd$

C117. $\forall u \in USERS, \{role_1, role_2\} \not\subseteq active\_roles(u)$

The *dsd* relation is updated. The two roles may not be in either separation of duties relation, and any role inheriting one of the arguments must be in dsd with the other. The arguments may not be both active for same user in same state.

rmDsd

Arguments:

$role_1, role_2$

Semantics:

$dsd' = dsd \setminus \{(role_1, role_2), (role_2, role_1)\}$

Conditions:

C121. $role_1, role_2 \in ROLES$

C122. $(role_1, role_2) \in dsd$

C123. $\forall r \in ROLES, role_1 \rightarrow r \Rightarrow (r, role_2) \notin dsd$

C124. $\forall r \in ROLES, role_2 \rightarrow r \Rightarrow (r, role_1) \notin dsd$

The *dsd* relation is updated. After deleting the dsd relation between the two roles, no one of them may remain in dsd with a role inherited by the other.

setCardinality

Arguments:

$role, c$

Semantics:

$cardinality' = (cardinality \setminus$
$\{role \mapsto cardinality(role)\}) \cup \{role \mapsto c\}$

Conditions:

C131. $c \in \mathbf{N} \cup \{\infty\}$

C132. $role \in ROLES$

C133. $|authorized\_users(role)| \leq c$

The cardinality for *role* is updated. The new cardinality must be numeric or infinite, and not smaller than the number of users currently authorized for *role*.

addActiveRoles
Arguments:
user, *roleset*
Semantics:
$active\_roles' = (active\_roles \setminus \{user \mapsto active\_roles(user)\}) \cup \{user \mapsto active\_roles(u) \cup \text{roleset}\}$
Conditions:
C141. $user \in USERS$
C142. $roleset \subseteq authorized\_roles(user)$
C143. $\forall r_1, r_2 \in roleset \cup active\_roles(u), (r_1, r_2) \notin dsd$

The entry for *user* in *active_roles* is updated. The added active roles must be authorized for *user*. The new set of active roles of *user* may not contain roles in dsd.

rmActiveRoles
Arguments:
user, *roleset*
Semantics:
$active\_roles' = (active\_roles \setminus \{user \mapsto active\_roles(user)\}) \cup \{user \mapsto active\_roles(u) \setminus \text{roleset}\}$
Conditions:
C151. $user \in USERS$
C152. $roleset \subseteq active\_roles(user)$

The entry for *user* in *active_roles* is updated.

Theorem 7. Let *op* be one of the operations defined above and *args* its arguments. If *s* is a consistent state and *args* satisfies the conditions specified for operation *op*, then $s' = \delta(s, op, args)$ is a consistent state.

Proof. We present the proof for the *addAssignment* operation. The proofs for other operations are similar.

Assume that the arguments *user*, *role* of the *addAssignment* operation satisfy the conditions C51-C56. Let us prove that, after executing *addAssignment* for *user* and *role*, the state *s*' is still consistent. We will show that the consistency conditions provided by Theorem 6 hold in state *s*'.

P1. There are two cases. If role *r* is such that $\neg(role \rightarrow^* r)$, then our operation does not modify *authorized_users*(*r*). If $role \rightarrow^* r$, condition C56 ensures that $|authorized\_users(r)| < cardinality(r)$ in state *s*. Assigning *user* to *role* may increase $|authorized\_users(r)|$ by at most 1, hence P1 holds in *s*'.

P3. Let $r_1, r_2 \in assigned\_roles'(u)$ in state *s*', where r_1, r_2 are roles and *u* is a user, and let us show that $\neg(r_1 \rightarrow^+ r_2)$.
If $u \neq user$, or $r_1, r_2 \neq role$, then $r_1, r_2 \in assigned\_roles(u)$ in state *s*, and, consequently, $\neg(r_1 \rightarrow^+ r_2)$, which is preserved by the transition from state *s* to state *s*', because the *inherits* relation does not change.

If $u = user$ and $r_1 = role$, assume by way of contradiction that $role \rightarrow^+ r_2$ in state *s*'. Then $role \rightarrow^+ r_2$ and $r_2 \in assigned\_roles(user)$ must have held also in state *s*, and contradict C54.

If $u = user$ and $r_2 = role$, assume by way of contradiction that $r_2 \rightarrow^+ role$. Then $r_2 \rightarrow^+ role$ and $r_2 \in assigned\_roles(user)$ must have held also in state *s*, and imply that $role \in authorized\_roles(\text{user})$ in state *s*, which contradicts C53.

P20. Let $r_1, r_2 \in assigned\_roles'(\text{u})$ in state *s*', where r_1, r_2 are roles and *u* is a user, and let us show that $(r_1, r_2) \notin ssd'$.

If $u \neq user$, then $r_1, r_2 \in assigned\_roles(u)$ also in state *s*, hence $(r_1, r_2) \notin ssd$, which is preserved by *addAssignment*.

If $u = user$, and, for example, $r_1 = role$, then $r_2 \in assigned\_roles(user)$ also in state *s*, and C55 implies that $(role, r_2) \notin ssd$, hence $(role, r_2) \notin ssd'$.

P2, P5, P6, P18, P10, P11, P12, P13, P14, P19 do not depend in any way on *addAssignment*; thus, if they hold in state *s*, then they hold also in state *s*'.□

4 Conclusions

Formal specification of the consistency requirements of the RBAC Database leads to the development of an equivalent reduced set of consistency properties and results in a more efficient Admin Tool implementation. Showing that specific preconditions for each RBAC Database operation assure preservation of RBAC Database consistency (Theorem 7) increases the efficiency of the Admin Tool. This proof alleviates the need for performing a full database consistency check at each operation. Theorem 7 shows that a single consistency check at the beginning of an administrative session and a precondition check for each operation are sufficient to ensure database integrity.

Careful implementation of the Admin Tool following the formal specification of RBAC Database consistency checks and operation preconditions results in a higher assurance Admin Tool. In fact, in proving Theorem 7, it was realized that some preconditions were initially omitted from the implementation.

Further work on performance analysis of the algorithms used to implement Admin Tool's consistency checks would be useful. In addition, it would be desirable to discover a *minimal* set of RBAC Database consistency properties.

References

[1] D. Ferraiolo, J. Cugini, and D. R. Kuhn. Role Based Access Control: Features and Motivations. In *Annual Computer Security Applications Conference*, IEEE Computer Society Press, 1995.

[2] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, and D.R. Kuhn. Role Based Access Control for the World Wide Web. In *20th National Information System Security Conference*, NIST/NSA, 1997.

[3] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role Based Access Control Models. *IEEE Computer 29* (2), February 1996.

[4] R. Sandhu. Role Activation Hierarchies. Proceedings of the Third ACM Workshop on Role-Based Access Control, October 1998.

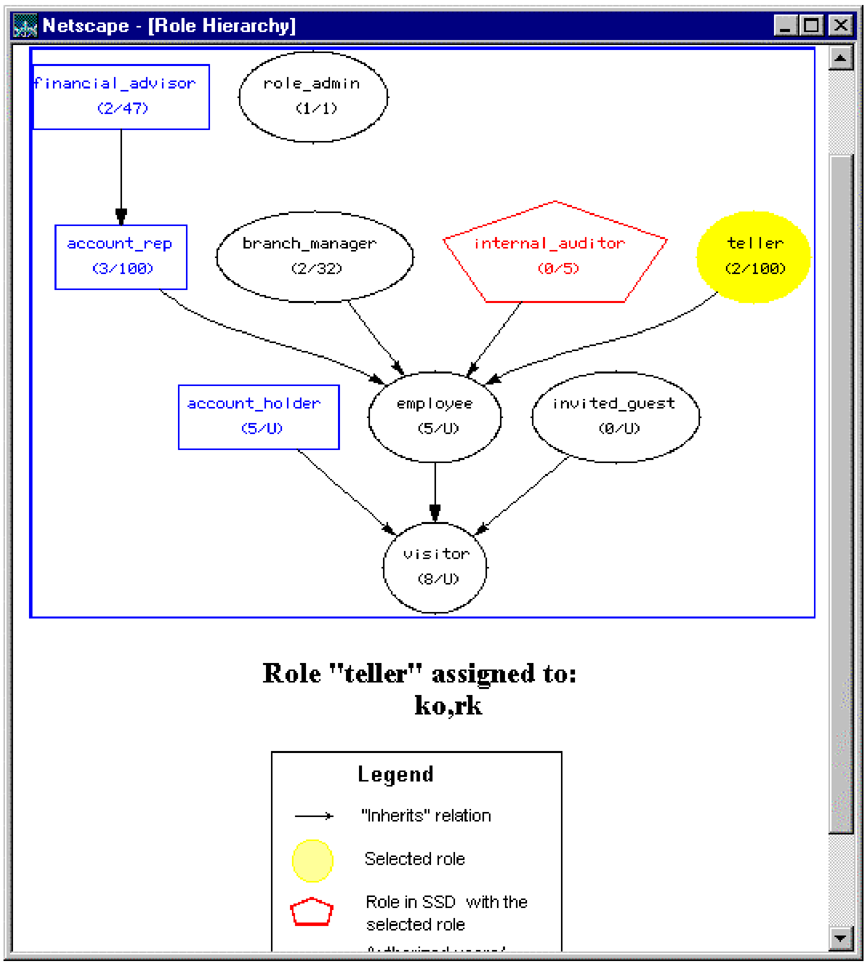

Figure 1. Admin Tool: Graphical Display

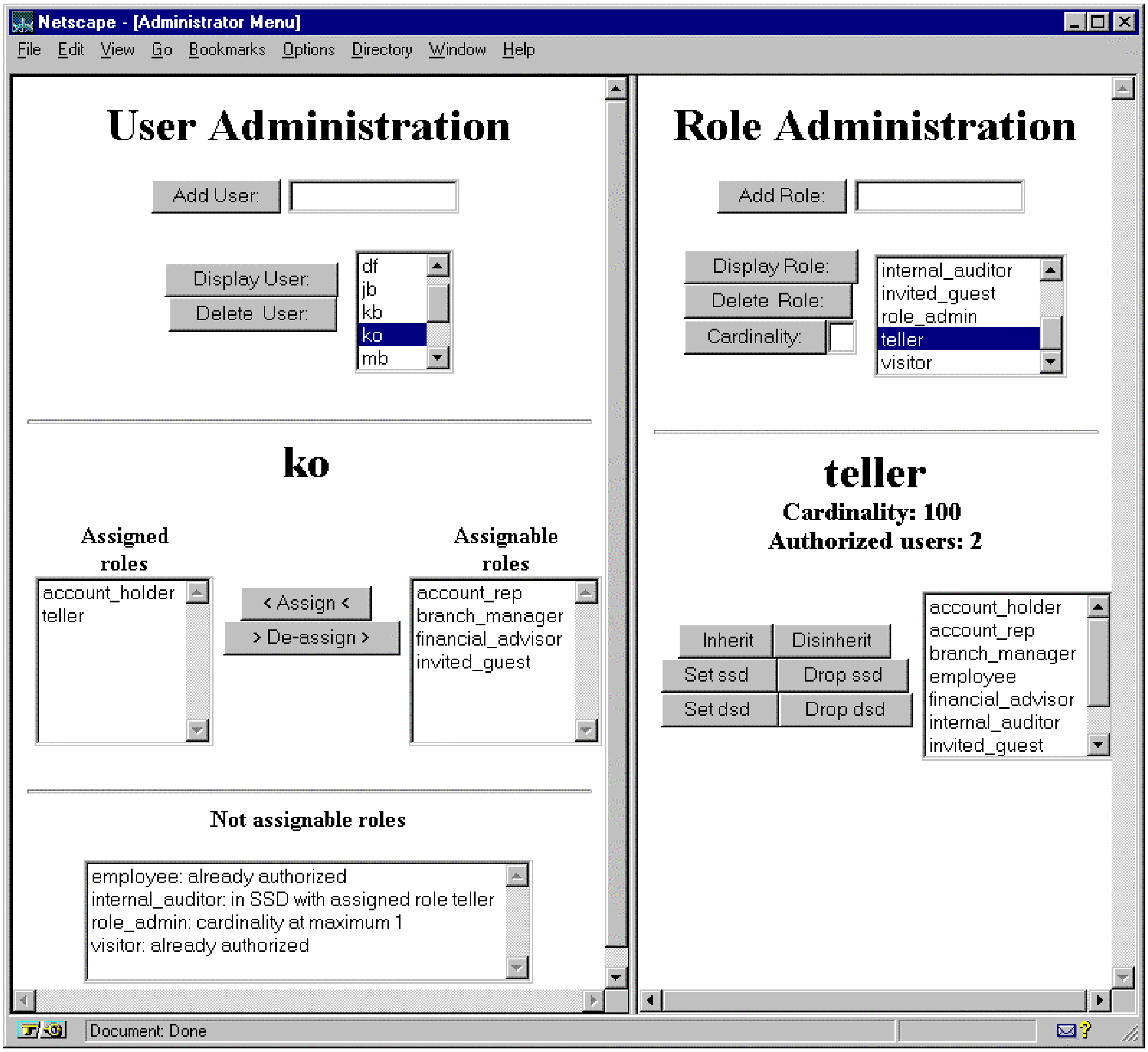

Figure 2. Admin Tool: Main Display