NISTIR 6192

A REVISED MODEL FOR ROLE-BASED ACCESS CONTROL

W. A. Jansen

July 9, 1998

Table of Contents

1. Introduction

Role Based Access Control (RBAC) is a term used to describe security mechanisms that mediate users' access to computational resources based on role constructs. A role defines a set of allowable activities for users authorized its use. It can be thought of as a job title or position within an organization, which represents the authority needed to conduct the associated duties. For many types of organizations, RBAC provides a more intuitive and effective way to represent and manage authorization information than other forms of access control.

A general model for RBAC, including essential features and the rationale for them, is specified in [1]. Since publication of the general model, a number of notational problems and inconsistencies have been noted, ranging in degree from trivial to serious. This report reviews the original RBAC model as defined in [1], corrects notational problems, and formulates a revised model to address noted discrepancies. The aim is to improve understanding of implications within the original model and to provide a firm baseline for subsequent activities involving the use or implementation of the model.

Properties of the revised RBAC model are organized along two themes: static properties and dynamic properties. Static properties deal mainly with constraints on role membership, while dynamic properties deal with constraints on role activation [4]. With this perspective, several new properties are derived from those identified in the original model, and other new properties are proposed, taking into account aspects found in a number of related models [2, 3, 4, 8]. A list of minimum recommended properties, requisite for a system or product to be considered as a true instantiation of the revised model, is also proposed. Very little of the rationale given in [1] is repeated here; therefore, the reader is assumed to be familiar with its contents.

2. Model Elements

The main components of the original model are User, Subject, Role, Function, Operation, and Object. The complete set of relationships among those components was not specified, however. Figure 1(a) gives a representation derived from the available information, whereby a single headed arrow represents a one-to-many, binary relationship between model components, and a double headed arrow represents a many-to-many, binary relationship.

One drawback of the original conceptual model is that it associates elements of Operation directly with those of Role, but independently of Object. In practice, elements of Operation and Object are closely coupled together when associated with an element of Role. This three-way association illustrated in Figure 1 (b) is more accurately represented as a ternary relation (i.e., many-to-many-to-many relationship) in the revised model.

Another drawback in the original model is the use of the Function component to define operational separation of duty (i.e., operations that are mutually exclusive of one another). There, the Function component identified a set of operations associated with a critical business function, whereby no single user was allowed to perform all elements in the set. Rather than treating Function as a component of the model, operational separation of duty can be defined instead as a constraint on Permission, similar to the treatment given other separation of duty properties in the original model. This approach has the benefit of simplifying the basic model, while unifying the form of the model specification. In summary, the components of the revised model are User, Subject, Role, Operation, and Object, as defined below.

u : User
User = the set of people, both trusted (e.g., administrators) and untrusted, who use the system.

x, y : Subject
Subject = the set of active entities of the system, operating within roles on behalf of individual users.

i, j, k : Role
Role = the set of named duties or job functions within an organization.

op : Operation
Operation = the set of access modes or types permitted on objects of the system.

o : Object
Object = the set of passive entities within the system, protected from unauthorized use.

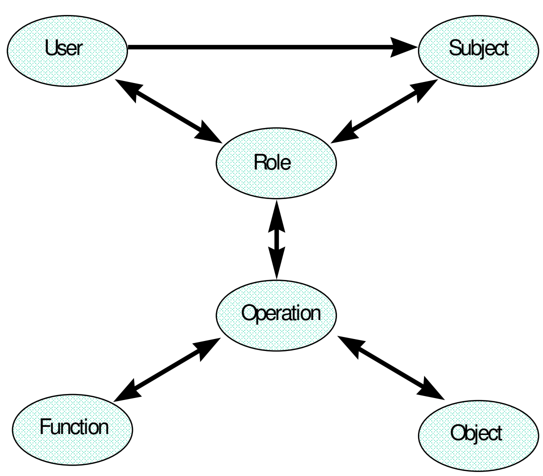

Figure 1(a): Original Relationships

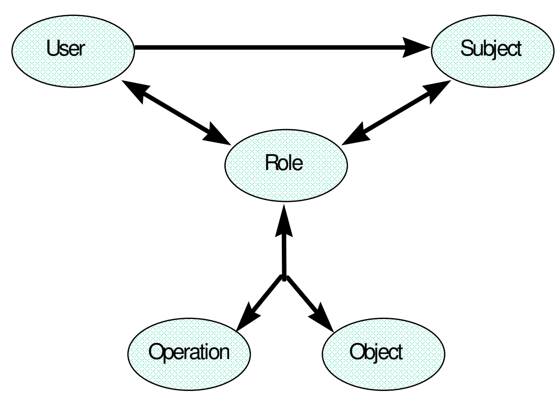

Figure 1(b): Revised Relationships

3. Mappings & Relations

The mappings from the original model are shown Figure 2. They further refine the general relationships among the components of the model given in Figure 1(a), and are used to express properties of the model. As clearly seen in Figure 2, some redundant mappings exist between some components, namely User and Role, and Subject and Role. While a particular mapping may be useful in expressing a specific property, only a subset of the mappings is needed to express all the properties of the model. Moreover, redundant mappings increase the difficulty of expressing properties unambiguously, which has led to inconsistencies. The reformulation given in this paper eliminates the authorized-roles and role-members functions, and expresses the original set of properties with the remaining functions.

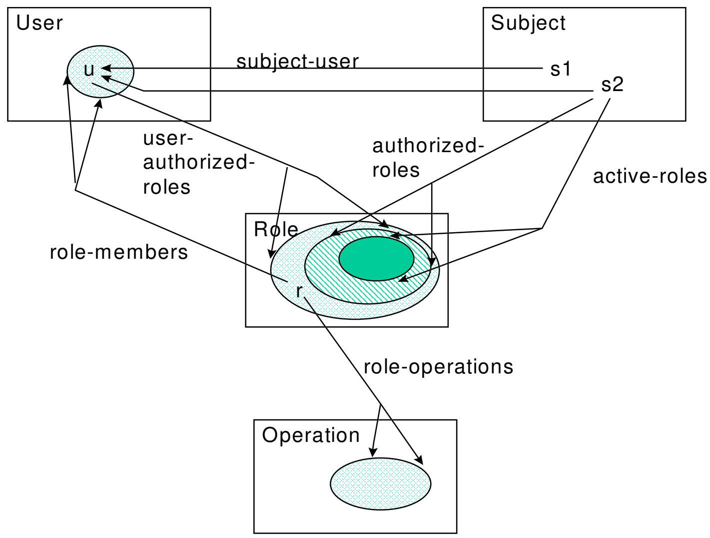

Figure 2: Original Mappings

Eliminating authorized-roles results in a significant simplification of the property that the active-roles of a subject must be a subset of its authorized-roles, which in turn must be a subset of the user-authorized-roles for the user associated with the subject (i.e., active-roles[s] $\subseteq$ authorized-roles[s] $\subseteq$ user-authorized-roles[subject-user[s]]). This property is represented in Figure 2 by the nested ellipses within the rectangle for the Role component. The new property, that the active-roles of a subject must be a subset of the user-authorized-roles for the user associated with the subject (i.e., active-roles[s] $\subseteq$ user-authorized-roles[subject-user[s]]), while simpler, maintains the critical feature of the original (i.e., a subject is constrained by the authorization of its user).

Eliminating the role-member function doesn't have as dramatic an effect on the model as with authorized-roles, but, in conjunction with the above changes, minimizes the number of mappings utilized. The overall result, shown in Figure 3, can be compared with Figure 2. Other choices for the minimal set of mappings may be selected. For example, user-authorized-roles could be eliminated in lieu of role-member. The set selected, however, closely models the actions of an administrator in assigning permissions to roles and roles to users. The selected mappings are intended to be illustrative only. While they are used to define model properties precisely, the mappings should not be considered a restriction on an actual implementation, since other equally effective alternatives exist, including those employing redundant mappings.

Note the new model component, Permission, in Figure 3. For notational and conceptual purposes, the ternary relationship between Role, Operation, and Object is refined into a pair of binary relations: one between operations and objects, referred to as Permission; the other between

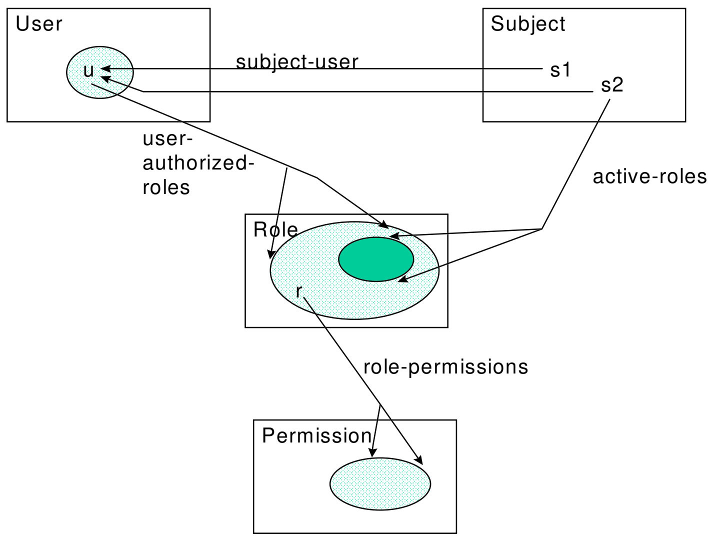

Figure 3: Revised Mappings

Role and Permission. The reformulation is shown in Figure 4, where Permission is used to designate a set of Operation/Object pairs associated with Role elements. The use of Permission conforms with the notion of privilege or permission found in present day information systems [2, 8]. The reformulation allows a permission to represent a broad range of access controls ranging from basic read/write/execute rights on files to more extensive administrator rights on operating systems. In Figure 3, the role-permissions function has replaced role-operations of the original model for consistency with the new component, Permission.

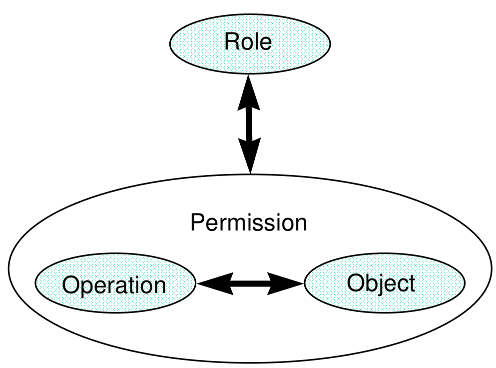

Figure 4: Decomposition of Ternary Relationship

The refined set of mappings for the basic model is given below, where "$\uparrow$" is used to represent the power set of the exponent, and "$\wp$" used to represent a subset of the indicated set.

user-authorized-roles: User $\rightarrow 2\uparrow$Role
user-authorized-roles[u] = the set of roles authorized for user u.

Permission: $\wp(\text{Operation} \times \text{Object})$
p, q : Permission
permission = a set of ordered operation/object pairs, <op,o>, where op is an operation that can be applied to object o.

role-permissions: Role $\rightarrow$ $2 \uparrow$Permission
role-permissions[i] = the set of permissions authorized for role i.

subject-user: Subject $\rightarrow$ User
subject-user[x] = the user u associated with the subject x.

active-roles: Subject $\rightarrow$ $2 \uparrow$Role
active-roles[x] = the set of roles in which a subject x is active.

4. Role Hierarchy

To facilitate administration of access control privileges and constraints, a role may be defined in terms of one or more other roles, with additional characteristics added to distinguish the new role further. A role defined this way is said to contain the roles that comprise its baseline, since it automatically takes on or inherits their collective characteristics as the basis for the new role being defined. Containment is similar to inheritance in object-oriented systems, whereby the properties and constraints of a containing role are inclusive of the properties and constraints of any contained role. Containment is also recursive; one role can contain other roles, which contain others, etc., as illustrated in Figure 5(a). By definition, a role cannot contain itself.

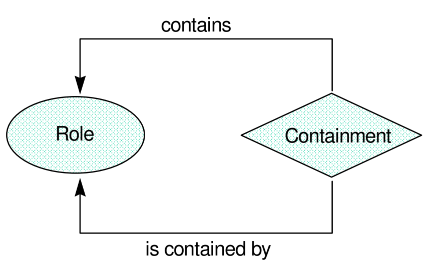

Figure 5(a): Containment Relation

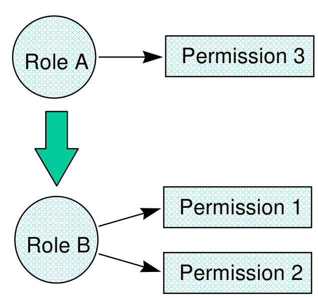

Figure 5(b): Inheritance View of Containment

Besides facilitating role administration, containment permits the substitution of role instances. For example, if role A contains role B, then instances of role A are treated as instances of role B for the purpose of access control. In Figure 5(b), users active within instances of Role A have the same capabilities as if they were active within instances of Role B, namely the access allowed through Permission 1 and Permission 2. In addition, users active within Role A also possess an additional capability, access allowed by Permission 3.

Containment can be characterized through the notion of effective roles. The effective roles of any given role include that role plus the set of roles contained by that role. For any role, the effective role set represents the capabilities afforded a user authorized the role. In the example above, the effective roles for Role A are Role A and Role B. At times it is also useful to consider the effective roles associated with a given user (or subject), which is the set of roles authorized (active) for that user (subject) plus all roles contained by any authorized (active) role. For example, assume there is a role C in addition to the roles defined above, and a user is authorized for both Roles A and C. The effective roles for that user would be Roles A, B, C, and any roles contained by Role C. Similarly, the effective permissions can be defined for a role or a user, as the set of permissions authorized each member of the effective role set respectively, for that role or that user. In the example of Figure 5(b), the effective permissions for Role A are Permission 1, Permission 2, and Permission 3.

Role Hierarchy is formally defined in terms of the following mappings and relations:

> Contains: $\wp(\text{Role}\times\text{Role})$
> contains = the set of ordered role pairs $\langle i,j \rangle$ having a containment relation, written as $i \succeq j$, where role i is said to contain role j, or alternatively, role j is said to be contained by i.
>
> effective-roles: Subject $\rightarrow 2\uparrow$Role
> effective-roles[x] = the union of the set of active roles for a subject, x, together with the set of roles contained by each active role; i.e., $\{j \ni j \in \text{active-roles}[x] \vee i \in \text{active-roles}[x] \wedge i \succeq j\}$.
>
> effective-permissions: Role $\rightarrow 2\uparrow$Permission
> effective-permissions[i] = the set of permissions authorized each of the effective roles for role i; i.e., $\forall j\ \{p \ni p \in \text{role-permissions}[j] \wedge (j \succeq i \vee j = i)\}$.

Role Hierarchy *(rule 1 redefined):* The containment relation defines an irreflexive and transitive relation on Roles, forming a quasi ordering of the elements in the set. The containment relation can also be shown to be antisymmetric. The quasi ordering of Roles is referred to as a role hierarchy.

> $\forall i\ \neg(i \succeq i)$ (irreflexive)
> $\forall i \forall k\ i \succeq j \wedge j \succeq k \rightarrow i \succeq k$ (transitive)

5. Static Properties

Static properties refer to properties of the model that do not involve either the Subject component or mappings from Subject to other basic components (vis., subject-user and active-roles). As their name implies, static properties apply early, at role authorization time, and are upheld throughout role activation. Hence, they are the most fundamental constraints and relationships expressed in the model, and also the strongest. Static properties include cardinality, separation of duty, and operational separation of duty. For each static property defined, a reference is given to a rule number, which corresponds to that appearing in the original model description [1]. Static properties are defined in terms of the mappings and relations below.

membership-limit: Role $\rightarrow$ N
membership-limit[i] = the maximum number of users that may be authorized a role; the default value is the total number of system users.

authorized-members: Role $\rightarrow$ N
authorized-members[i] = the number of users authorized either a given role or a role that contains the given role; i.e.; $|\{u \ni \exists j\ ((j \geq i \lor j = i) \land j \in \text{user-authorized-roles}[u])\}|$, where the cardinality of a set is expressed by a pair of bars "| |" delimiting the defined set.

SSD: $\wp(\text{Role} \times \text{Role})$
SSD = the symmetric set of role pairs <i,j> involved in a Static Separation of Duty (SSD) relationship (i.e., where i and j are mutually exclusive of one another for authorization to the same user due to an inherent conflict of interest); for a symmetric set <i,j> is a member iff <j,i> is also a member.

Mutex-authorization: $\wp(\text{Role} \times \text{Role})$
mutex-authorization = the symmetric set of role pairs <i,j> mutually exclusive of one another for authorization to the same user.

Mutex-permission: $\wp(\text{Permission} \times \text{Permission})$
mutex-permission = the symmetric set of permission pairs <p,q> mutually exclusive of one another for authorization to an individual role or to the set of roles authorized any user.

SOSD: $\wp(\text{Role} \times \text{Role})$
SOSD = the symmetric set of role pairs <i,j> involved in a Static Operational Separation of Duty (SOSD) relationship, with respect to the permissions in Mutex-permission; i.e., $\forall i \forall j \forall p \forall q$ SOSD = $\{ <i,j> \ni p \in \text{role-permissions}[i] \land q \in \text{role-permissions}[j] \land <p,q> \in \text{Mutex-permission}\}$.

Cardinality *(rule 3):* The number of users authorized a role at any one time cannot exceed the capacity (i.e., membership limits) of the role. For example, a role with a capacity of one would be used exclusively by the single user who is assigned to it. In terms of the mappings defined, the cardinality of the set of users who are authorized the same role must be less than or equal to the membership limit of that role.

$\forall i$ authorized-members $\leq$ membership-limit[i]

Cardinality Inheritance *(new rule):* Cardinality constraints are inherited by containing roles. A containing role must be assigned a membership limit less than or equal to that of any contained role.

$\forall i \forall j\ i \succeq j \rightarrow \text{membership-limit}[i] \leq \text{membership-limit}[j]$

Static Separation of Duty *(rule 2):* In many organizations, responsibilities are split among multiple roles as a countermeasure to fraud, misappropriation of assets, and other conflict of interest based policy deviations that require the collusion of two or more individuals. In commercial transactions, for example, one role may have the capability to input transaction requests, while another the capability to approve them. A group of roles may be designated through the Static Separation of Duty (SSD) property as mutually exclusive of one another with regard to role authorization. That is, to avoid a possible conflict of interest, a user may be authorized to only one of the distinct roles so designated. SSD involving multiple roles is expressed pairwise, using the SSD relation. If for example i, j, and k are such roles, then <i,j>, <j,i>, <i,k>, <k,i>, <j,k>, <k,j> are members of SSD. This representation of SSD differs syntactically but not semantically from the original model, which uses a function to return a mutually exclusive set of roles for a given role.

$\forall i \forall j \forall u\ i \in \text{user-authorized-roles}[u] \wedge j \in \text{user-authorized-roles}[u] \rightarrow \langle i,j \rangle \notin SSD$

or alternatively

$\forall i \forall j \forall u\ \langle i,j \rangle \in SSD \rightarrow (i \in \text{user-authorized-roles}[u] \rightarrow j \notin \text{user-authorized-roles}[u])$

SSD Safety *(new rule):* The Static Separation of Duty property, when applied, asserts there is a conflict of interest among the capabilities authorized a set of roles. Besides explicit SSD relationships, there may also be implicit SSD relationships regarding other roles possessing comparable capabilities. This property ensures that implicit SSD relationships, which can be inferred from explicit SSD relationships, are upheld. If a user is authorized a role, which has an SSD relationship with another role whose effective permissions are a subset of a third role's, then the user cannot be authorized the third role.

$\forall u \forall i \forall j \forall p\ i \in \text{user-authorized-roles}[u] \wedge \langle i,j \rangle \in SSD \wedge \exists k(p \in \text{effective-permissions}[j] \rightarrow p \in \text{effective-permissions}[k]) \rightarrow k \notin \text{user-authorized-roles}[u]$

SSD Hierarchical Consistency *(new rule):* An SSD relationship cannot exist between roles that have a containment relation between them or are contained by another role in common. The rationale behind this property is that, by definition, an instance of a containing role is treated the same as an instance of any contained role (i.e., the effective roles of the containing role include the contained role); therefore, the conflict of interest asserted by an SSD relationship cannot exist without contradicting the capability intended by the containment relation.

$\forall i \forall j\ (i \succeq j \vee \exists k\ (k \succeq i \wedge k \succeq j) \rightarrow \langle i,j \rangle \notin SSD)$

SSD Inheritance *(new rule):* SSD relationships are inherited by containing roles. If one role contains another role that has an SSD relationship with a third role, then the containing role also has an SSD relationship with the third role. This property must hold since a contradiction occurs if the effective roles for the containing role includes, in addition to the contained role, the third role. Through SSD Inheritance, distinct hierarchies of roles, isolated with regard to role authorization, are asserted from SSD relationships among a basic set of roles.

$$\forall i \forall j \forall k \ i \succeq j \wedge \langle j,k \rangle \in SSD \rightarrow \langle i,k \rangle \in SSD$$

Static Mutual Exclusion *(new rule):* Static Mutual Exclusion (SME) is somewhat similar to SSD insofar as one group of roles may be completely segregated from another with regard to role authorization. However, the motivation behind mutual exclusion is not as a countermeasure to collusion, but rather as an administrative aid to codifying organizational policy. SME involving multiple roles is expressed pairwise, using Mutex-authorization. Because SME is primarily an administrative feature, and not one based on conflict of interest or other incompatibility among the capabilities authorized a set of roles, there is no need for any associated Safety or Hierarchical Consistency properties as with SSD. Omission of the latter property is significant, since, unlike SSD, it allows constraints to exist within as well as among hierarchies.

For example, in a hospital system a staff member role may be inherited by an intern role, which in turn may be inherited by physician role. If a doctor were assigned to both an intern and a physician role through either accident or intention, there would be no inherent conflict of interest, since a containment relationship exists between them. However, to avoid accounting inconsistencies, the hospital policy may require that a doctor on staff be classified as either a physician or an intern, but not both. It would be incorrect to use SSD to represent this policy, and also contradictory because of the containment relation. However, the policy can be correctly represented by designating the physician and intern roles mutually exclusive of one another through the SME property. SME can be considered as simply an additional type of constraint that can be applied among any set of roles, regardless of their fundamental capabilities.

$$\forall i \forall j \forall u \ i \in \text{user-authorized-roles}[u] \wedge j \in \text{user-authorized-roles}[u] \rightarrow \langle i,j \rangle \notin \text{Mutex-authorization}$$

or alternatively

$$\forall i \forall j \forall u \ \langle i,j \rangle \in \text{Mutex-authorization} \rightarrow (i \in \text{user-authorized-roles}[u] \rightarrow j \notin \text{user-authorized-roles}[u])$$

SME Inheritance *(new rule):* SME relationships are inherited by containing roles, if those relationships occur with roles outside the chain of containing roles or lower within the chain. If one role contains another role that has an SME relationship with a third role, then the containing role also has an SME relationship with the third role, provided that the third role does not contain either role and is not contained directly by the first role (i.e., only indirectly through the second role). As with SSD Inheritance, this property requires that a specific type of constraint be upheld among any containing roles. However, the way in which the constraint is upheld within a chain of containing roles is different in some cases. For example, for the physician$\succeq$intern$\succeq$staff member containment chain from the example above, a surgeon role is defined in terms of physician (i.e., surgeon$\succeq$physician). The designated SME relationship between physician and intern, through SME inheritance, would require an SME relationship to also exist between surgeon and intern, but not surgeon and physician.

$$\forall i \forall j \forall k\ i \succeq j \wedge \neg(k \succeq j) \wedge (j \succeq k \vee \neg(i \succeq k)) \wedge \langle j,k \rangle \in \text{Mutex-authorization} \rightarrow \langle i,k \rangle \in \text{Mutex-authorization}$$

Static Operational Separation of Duty *(rule 8 restated):* The rationale behind SOSD is that business tasks are composed of a number of operations, only a subset of which a single user may perform. SOSD is enforced by using permissions to represent allowable subsets of operations on objects involved in business tasks, and designating a group of permissions as mutually exclusive of one another with respect to the roles authorized any single user. Mutually exclusive permissions ensure that no single user may be authorized one or more roles having permissions involved in an SOSD relationship. One side effect of this property is that no role may be authorized more than one permission from a group of permissions designated mutually exclusive of one another. SOSD is expressed among multiple permissions through pairwise specification of members in a mutual exclusion set, Mutex-permission, which in turn determines the membership of the SOSD relation.

$$\forall i \forall j \forall u \forall p \forall q\ \ i \in \text{user-authorized-roles}[u] \wedge j \in \text{user-authorized-roles}[u] \wedge p \in \text{role-permissions}[i] \wedge q \in \text{role-permissions}[j] \rightarrow \langle p,q \rangle \notin \text{Mutex-permission}$$

or alternatively

$$\forall i \forall j \forall u\ \ i \in \text{user-authorized-roles}[u] \wedge j \in \text{user-authorized-roles}[u] \rightarrow \langle i,j \rangle \notin \text{SOSD}$$

or alternatively

$$\forall i \forall j \forall u \forall p \forall q\ \ p \in \text{role-permissions}[i] \wedge q \in \text{role-permissions}[j] \wedge \langle p,q \rangle \in \text{Mutex-permission} \rightarrow (i \in \text{user-authorized-roles}[u] \rightarrow j \notin \text{user-authorized-roles}[u])$$

or alternatively

$$\forall i \forall j \forall u\ \langle i,j \rangle \in \text{SOSD} \rightarrow (i \in \text{user-authorized-roles}[u] \rightarrow j \notin \text{user-authorized-roles}[u])$$

SOSD Hierarchical Consistency *(new rule):* An SOSD relationship cannot exist between roles that have a containment relation between them or are contained by another role in common.

$$\forall i \forall j\ (i \succeq j \vee \exists k\ (k \succeq i \wedge k \succeq j) \rightarrow \langle i,j \rangle \notin \text{SOSD})$$

Static Operational Separation of Duty Inheritance *(new rule):* SOSD relationships are inherited by containing roles. If one role contains another role that has an SOSD relationship with a third role, then the containing role also has an SOSD relationship with the third role.

$$\forall i \forall j \forall k\ i \succeq j \wedge \langle j,k \rangle \in \text{SOSD} \rightarrow \langle i,k \rangle \in \text{SOSD}$$

6. Dynamic Properties

Dynamic properties are used in conjunction with static properties to maintain additional constraints and relationships on the activities that can occur when a role is active (i.e., a subject is active in an authorized role on behalf of a user). Dynamic properties refer to properties of the model that involve either the Subject component or mappings from Subject to other basic components (i.e., subject-user and active-roles). For every static property, a corresponding dynamic property may be defined. Dynamic properties are in a sense weaker than their static property counterparts, since they come into play at role activation time rather than at role authentication time. Weaker doesn't necessarily mean undesirable. Instead, they offer an additional degree of flexibility desirable in many contexts as either a substitute for, or complement to, static properties. Dynamic properties include role activation, cardinality, separation of duty, and operational separation of duty, and utilize the mappings and relations below.

exec: Subject×Operation×Object → {True, False}
exec[x,op,o] = True iff subject x can perform an operation op on object o; otherwise, False.

active-membership-limit: Role → N
active-membership-limit[i] = the maximum number of users that may be active in a role.

active-members: Role → N
active-members[i] = the number of users active either in a given role or in a role that contains the given role; i.e.; $|\{u \ni \forall x \exists j\ ((j \geq i \vee j = i) \wedge j \in \text{active-roles}[x] \wedge u = \text{active-user}[x])\}|$

DSD: $\wp$(Role×Role)
DSD = the symmetric set of role pairs <i,j> involved in a Dynamic Separation of Duty (DSD) relationship (i.e., where i and j mutually exclusive of one another for activation by the same user, due to an inherent conflict of interest).

Mutex-activation: $\wp$(Role×Role)
mutex-activation = the symmetric set of role pairs <i,j> mutually exclusive of one another for activation by the same user.

Mutex-perm: $\wp$(Permission×Permission)
mutex-perm = the symmetric set of permission pairs <p,q> mutually exclusive of one another for activation by the same user, simultaneously within different roles.

DOSD: $\wp$(Role×Role)
DOSD = the symmetric set of role pairs <i,j>, involved in a Dynamic Operational Separation of Duty (DOSD) relationship with respect to the permissions in Mutex-perm; i.e., $\forall i \forall j \forall p \forall q$ DOSD = $\{\langle i,j\rangle \ni p \in \text{role-permissions}[i] \wedge q \in \text{role-permissions}[j] \wedge \langle p,q\rangle \in \text{Mutex-perm}\}$.

Role Authorization *(rule 4, subsumes assumption 1 in original model, Consistent Subject):* A subject cannot be active in a role that is not authorized for its associated user. Note, this rule should be renamed Role Activation for accuracy, since it only comes into play after authorization.

$$\forall x \forall i \; i \in \text{active-roles}[x] \rightarrow i \in \text{user-authorized-roles}[\text{subject-user}[x]]$$

Operation Authorization *(rule 7 subsumes rule 5, Role Execution):* A subject can perform an operation on an object if, and only if, the subject is acting within an effective role authorized that permission.

$$\forall x \forall op \forall o \; \text{exec}[x,op,o] \equiv \exists i \; (i \in \text{effective-roles}[x] \wedge p \in \text{role-permissions}[i] \wedge \langle op,o \rangle \in p)$$

Dynamic Cardinality *(new rule):* The number of users active in a role at any one time cannot exceed the dynamic capacity (i.e., active-membership-limit) of the role. This rule, though more difficult to implement than Static Cardinality, seems to be much more desirable, since the role capacity is maintained at activation time as opposed to authorization time. For example, a role with a dynamic capacity of one would allow at most a single role instance to be active at any time, ensuring consecutive use of the role's capabilities by any assigned users.

$$\forall i \; \text{active-members}[i] \leq \text{active-membership-limit}[i]$$

Dynamic Cardinality Inheritance *(new rule):* Cardinality constraints are inherited by containing roles. If one role contains another role, then the containing role must have an active membership limit less than or equal to that of the contained role.

$$\forall i \forall j \; i \succeq j \rightarrow \text{active-membership-limit}[i] \leq \text{active-membership-limit}[j]$$

Dynamic Separation of Duty *(rule 6):* A group of roles may be designated as mutually exclusive of one another with regard to role activation, ensuring that at any one time a user may be active in only one of the distinct roles so designated. Dynamic Separation of Duty (DSD) involving multiple roles is expressed pairwise, using the DSD relation. DSD is a memoryless property insofar as no history of activation is kept for a user. Although DSD roles are prevented from being activated simultaneously by a user, they may be activated consecutively by simply dropping one role and assuming another, negating the usefulness of the property for some environments.

$$\forall x \forall y \forall i \forall j \; i \in \text{active-roles}[x] \wedge j \in \text{active-roles}[y] \wedge \text{subject-user}[x] = \text{subject-user}[y] \rightarrow \langle i,j \rangle \notin \text{DSD}$$

or alternatively

$$\forall x \forall y \forall i \forall j \; \langle i,j \rangle \in \text{DSD} \wedge \text{subject-user}[x] = \text{subject-user}[y] \rightarrow (i \in \text{active-roles}[x] \rightarrow j \notin \text{active-roles}[y])$$

DSD Safety *(new rule):* As with SSD, there may also be implicit DSD relationships regarding other roles possessing comparable capabilities, besides explicit DSD relationships. This property ensures that implicit DSD relationships are upheld at activation time. If a user is active in a role, which has a DSD relationship with another role whose effective permissions are a subset of a third role's, then the user cannot also be active in the third role.

$$\forall x \forall y \forall i \forall j \forall p\ i \in \text{active-roles}[x] \wedge \text{subject-user}[x] = \text{subject-user}[y] \wedge \langle i,j \rangle \in DSD \wedge \exists k (p \in \text{effective-permissions}[j] \rightarrow p \in \text{effective-permissions}[k]) \rightarrow k \notin \text{active-roles}[y]$$

DSD Hierarchical Consistency *(new rule):* A DSD relationship cannot exist between two roles that have a containment relationship between them or are contained by another role in common. The same rationale that applied for SSD applies for DSD, namely that a DSD relationship cannot be asserted in such situations without contradicting the capability intended by the containment relation.

$$\forall i \forall j\ (i \succeq j \vee \exists k\ (k \succeq i \wedge k \succeq j) \rightarrow \langle i,j \rangle \notin DSD)$$

DSD Inheritance *(new rule):* DSD relationships are inherited by containing roles. If one role contains another role that has a DSD relationship with a third role, then the containing role also has a DSD relationship with the third role. Through DSD Inheritance, distinct hierarchies of roles, isolated with regard to role activation, are asserted from DSD relationships among a basic set of roles.

$$\forall i \forall j \forall k\ i \succeq j \wedge \langle j,k \rangle \in DSD \rightarrow \langle i,k \rangle \in DSD$$

Dynamic Mutual Exclusion *(new rule):* A group of roles may be designated as mutually exclusive of one another with regard to role activation, ensuring that at any one time a user may be active in only one of the distinct roles so designated. While Dynamic Mutual Exclusion is similar in function to DSD, the underlying motivation for isolating roles is not one of conflict of interest, but of codifying organizational policy. This distinction is subtle, but critical in differentiating associated properties. In particular, there is no need for any associated Safety or Hierarchical Consistency properties, since no fundamental incompatibilities exist among designated roles. As with DSD, this property is memoryless and of limited usefulness in some environments.

$$\forall x \forall y \forall i \forall j\ i \in \text{active-roles}[x] \wedge j \in \text{active-roles}[y] \wedge \text{subject-user}[x] = \text{subject-user}[y] \rightarrow \langle i,j \rangle \notin \text{Mutex-activation}$$

or alternatively

$$\forall x \forall y \forall i \forall j\ \langle i,j \rangle \in \text{Mutex-activation} \wedge \text{subject-user}[x] = \text{subject-user}[y] \rightarrow (i \in \text{active-roles}[x] \rightarrow j \notin \text{active-roles}[y])$$

DME Inheritance *(new rule):* DME relationships are inherited by containing roles, if those relationships occur with roles outside the containment chain or lower in the chain. If one role contains another role that has a DME relationship with a third role, then the containing role also has a DME relationship with the third role, provided that the third role does not contain either role and is not contained directly by the first role (i.e., only indirectly through the second role). As with DSD Inheritance, this property requires that a specific type of constraint be upheld among any containing roles. However, the way in which the constraint is upheld within a chain of containing roles is different in some cases.

$$\forall i \forall j \forall k\ i \succeq j \wedge \neg(k \succeq j) \wedge (j \succeq k \vee \neg(i \succeq k)) \wedge \langle j,k \rangle \in \text{Mutex-activation} \rightarrow \langle i,k \rangle \in \text{Mutex-activation}$$

Dynamic Operational Separation of Duty *(new rule):* A group of permissions may be designated as mutually exclusive of one another with regard to the roles activated by a subject on behalf of any single user. One side effect of this property is that no role may be authorized more than one permission from a group of permissions designated mutually exclusive of one another. As with DSD, this property is memoryless and has limited utility in some environments.

$$\forall x \forall y \forall i \forall j \forall p \forall q\ \ i \in \text{active-roles}[x] \wedge j \in \text{active-roles}[y] \wedge \text{subject-user}[x] = \text{subject-user}[y] \wedge p \in \text{role-permissions}[i] \wedge q \in \text{role-permissions}[j] \rightarrow \langle p,q \rangle \notin \text{Mutex-perm}$$

or alternatively

$$\forall x \forall y \forall i \forall j\ \ i \in \text{active-roles}[x] \wedge j \in \text{active-roles}[y] \wedge \text{subject-user}[x] = \text{subject-user}[y] \rightarrow \langle i,j \rangle \notin \text{DOSD}$$

or alternatively

$$\forall x \forall y \forall i \forall j \forall p \forall q\ \ p \in \text{role-permissions}[i] \wedge q \in \text{role-permissions}[j] \wedge \langle p,q \rangle \in \text{Mutex-perm} \wedge \text{subject-user}[x] = \text{subject-user}[y] \rightarrow (i \in \text{active-roles}[x] \rightarrow j \notin \text{active-roles}[y])$$

or alternatively

$$\forall x \forall y \forall i \forall j\ \langle i,j \rangle \in \text{DOSD} \wedge \text{subject-user}[x] = \text{subject-user}[y] \rightarrow (i \in \text{active-roles}[x] \rightarrow j \notin \text{active-roles}[y])$$

DOSD Hierarchical Consistency *(new rule):* The same rationale that applied for SOSD applies for DOSD. If a containment relationship exists between two roles or a common heir to both exists, then a DOSD relationship cannot exist between them.

$$\forall i \forall j\ (i \succeq j \vee \exists k\ (k \succeq i \wedge k \succeq j) \rightarrow \langle i,j \rangle \notin \text{DOSD})$$

Dynamic Operational Separation of Duty Inheritance *(new rule):* DOSD relationships are inherited by containing roles. If one role contains another role that has a DOSD relationship with a third role, then the containing role also has a DOSD relationship with the third role.

$$\forall i \forall j \forall k\ i \succeq j \wedge \langle j,k \rangle \in \text{DOSD} \rightarrow \langle i,k \rangle \in \text{DOSD}$$

7. Other Features and Properties

The scope of both the original and revised models could be expanded in a variety of areas that for certain environments would prove useful. The areas include features related to administration, separation of duty, and role attributes. Though these features are not present in the revised model, they provide both an understanding of the limits of applicability of the current model and a direction for further enhancements.

Administration*:* RBAC can be treated as either a discretionary or non-discretionary access control method. The treatment given in this paper is oriented toward the latter method. The model implicitly requires administration roles to be distinct from user roles, insofar as their permissions deal solely with the policy attribute components of the model: User-to-Role and Role-to-Permission mappings, containment relations, cardinality constraints, and separation of duty constraints. Personnel not authorized administration roles are denied these permissions and must operate within the confines of the roles defined for and assigned to them by an administrator. Conversely, personnel who are authorized administration roles are restricted to administration of policy attribute components when active in those roles. Administrators are expected to maintain strict separation of roles.

Division of roles in this manner supports the principle of Attenuation of Privileges, which states that subjects should not be able to increase their privilege or grant to other subjects privileges they themselves do not own. Separation of authorization aspects from policy attribute management is useful in practice since authorization must be relatively independent of how policy attributes, such as roles, are managed [3]. However, a circular dependency between authorization and policy attribute management exists in such models, since authorization requires defined policy attributes for controlling access, and specification of policy attributes requires authorization to that information.

Features of the model to support strict separation of user and administrator roles are lacking. In order to model the non-discretionary perspective, some additional administrative support properties could be defined. For example, the model could be extended to account for administration and user role distinctions with respect to both the type of objects accessed and the type of role accessing the objects. While motivated by the desire to distinguish between user and administrator domains within the model, these properties may also find utility in defining distinctive sub-domains within those domains.

Separation of Duty*:* A of separation of duty properties have been defined for the revised model, the most basic and common being static and dynamic separation of duty. The rationale for separation of duty properties lies in the notion of conflict of interest avoidance. Users must not be placed in an environment where permitted actions allow a conflict of interest to occur. Each separation of duty property involves either mutually-exclusive collections of permissions (i.e., roles) or single permissions, and constraining the authorization or activation of users associated with those permissions.

In practice, the characteristics and security policy of one environment can be quite different from that of another, affecting the usefulness of a particular separation of duty property [6, 7]. One class of distinction already discussed is the authorization time vs. activation time differences between static and dynamic properties. In the revised model, all properties, including separation of duty, are memoryless with regard to the actions taken by a user active within a role or set of roles. That is, the properties form an assertion that must always hold during authorization or activation, regardless of the history of actions (viz., operations on objects) that may have occurred. While satisfactory for some environments, such as databases or operating systems where the set of operations and objects are somewhat uniform, in other environments, such as workflow applications, these properties may be inadequate in providing the desired controls.

In [4], a number of separation of duty properties are discussed, some of which are memoryfull, contingent upon the past actions taken by a user. While [4] gives a clear overview of both the variety and richness of this topic, it also points out situations where history-based constraints prove useful not only for conflict of interest avoidance, but also for control of workflow. History-based constraints include maintaining the strict sequence of actions taken on an object or collection of objects; restricting users to m of n actions on an object or collection of objects, regardless of the sequence in which roles are activated; and requiring m multiple users acting in m distinct roles to collaborate on a set of actions on an object or collection of objects. For example, the second constraint above prevents a user assigned two roles involved in a memoryfull DSD relationship, from acting on an object in one role and then the other, sequentially, or while simultaneously active in both roles. A normal DSD relationship only prevents the latter from occurring.

Role Attributes*:* The only class of attribute that has been defined for roles in the revised model is membership limits for both static and dynamic cardinality. Other similar characteristics could easily be added to the model, extending its applicability to other environments. These characteristics include constraints on the start and stop activation time for the role, whether a second authentication is needed to assume the role (e.g., to require a stronger form of authentication), and whether user acknowledgments are needed for actions being taken by a role (e.g., to counter a Trojan horse). The type of role, either user or administrator, is another possible attribute already mentioned in this section.

8. Core RBAC Features

Because there is a wealth of properties that one can define in an RBAC model, one question that commonly arises is whether a minimal subset of properties can be considered as the essential or core features of an RBAC implementation. While any determination of core features is subjective, it is, nevertheless, worthwhile to attempt such a categorization in the hopes of reaching an eventual consensus among developers, vendors, and consumers of RBAC systems. A general consensus is important to avoid the situation where products advertise RBAC capabilities, yet only support an alternative mechanism such as user/group access control lists that can be configured to simulate a particular configuration of roles.

One example of where a core feature list could be applied is the NIST RBAC protection profile [5]. The profile identifies minimal security requirements for controlling access to programs, transactions, and information, according to a user's assigned organizational role. The intent of the profile is to form the basis for evaluations of products claiming to meet the functionality and assurance requirements specified.

Table 1 summarizes the basic properties of the RBAC model specified in this paper. The entries within the table indicate whether the property identified from the row and column headings is considered a core feature (i.e., ✓) or not (i.e., ✗). While nearly all properties have static and dynamic counterparts, a couple of them are solely dynamic in nature and blocked out accordingly within the table.

Table 1: Summary of Core RBAC Features

<table>
<tr><th>Property</th><th>Static</th><th>Dynamic</th></tr>
<tr><td>Role Hierarchy</td><td colspan="2">✓</td></tr>
<tr><td>Role Authorization</td><td rowspan="2"></td><td>✓</td></tr>
<tr><td>Operation Authorization</td><td>✓</td></tr>
<tr><td>Cardinality</td><td>✓</td><td>✓</td></tr>
<tr><td>Cardinality Inheritance</td><td>✓</td><td>✓</td></tr>
<tr><td>Separation of Duty</td><td>✓</td><td>✓</td></tr>
<tr><td>Separation of Duty Safety</td><td>✗</td><td>✗</td></tr>
<tr><td>Separation of Duty Hierarchical Consistency</td><td>✓</td><td>✓</td></tr>
<tr><td>Separation of Duty Inheritance</td><td>✓</td><td>✓</td></tr>
<tr><td>Mutual Exclusion</td><td>✗</td><td>✗</td></tr>
<tr><td>Mutual Exclusion Inheritance</td><td>✗</td><td>✗</td></tr>
<tr><td>Operational Separation of Duty</td><td>✗</td><td>✗</td></tr>
<tr><td>Operational Separation of Duty Hierarchical Consistency</td><td>✗</td><td>✗</td></tr>
<tr><td>Operational Separation of Duty Inheritance</td><td>✗</td><td>✗</td></tr>
</table>

One of the main motivations for RBAC is the ease and flexibility it provides for administering system privileges for large numbers of users. Roles can easily be established and modified independently of the user assignments. Role hierarchies further simplify the definition and maintenance of roles, by allowing many subtle differences in capabilities to be captured and represented within a system through the containment relation. The ability to maintain distinctions among roles easily, supports the principle of least privilege, since users can be assigned the exact set of permissions necessary to perform their assigned tasks under the exact set of constraints. Although many systems claiming to be RBAC implementations do not currently support role hierarchies, this property is seen as fundamental for sound administration practices and, therefore, included in the core features.

Role Authorization and Operation Authorization are essential to properly activate a role and, once activated, performing actions (i.e., operations on objects) within the system. Without any doubt, they are mandatory core features. Role Authorization is defined independently of Role Hierarchy, while Operation Authorization is defined explicitly to account for containment relationships within the role hierarchy in terms of the effective-roles function.

The remaining items from the set of basic properties are Cardinality, Separation of Duty, Mutual Exclusion, Operational Separation of Duty, and Separation of Duty Safety. Note that with the inclusion of Role Hierarchy in the core set, any Separation of Duty or Operational Separation of Duty property selected causes all associated properties involving role hierarchies (i.e., Hierarchical Consistency and Inheritance properties) to be selected as well.

Cardinality is perhaps the most difficult property to categorize. The dynamic form is more flexible than the static form, yet more difficult to implement. Its use ranges from controlling the number of user employing a software package for software licensing restrictions, to limiting a role to a single member for managements positions where either the manager or a subordinate acting for the manager may be active, or a number of managers rotate into the role (e.g., during shifts). The static form is simple to implement, but has limited use (e.g., establishing a private role for a single user). As mentioned earlier, the Cardinality property is only one of several constraints associated with role attributes that could be specified. Since it is the only representative from this important class of properties, both the static and dynamic forms are included in the core set. With their inclusion, come the associated Cardinality Inheritance properties.

Separation of Duty is a long-standing security principal and one of the earliest motivations for RBAC [9]. Both Static and Dynamic forms of Separation of Duty properties are well understood and have a broad base of consistent specification [1, 2, 4, 9]; therefore, they are included in the core RBAC properties. Operational Separation of Duty properties, on the other hand, are omitted from the core properties, since they are relatively newer and lack consistency in definition among various models. As mentioned above, with the inclusion of Separation of Duty also come the associated Hierarchical Consistency and Inheritance properties.

Separation of Duty Safety is a new property, not specified in any of the references. While an important feature in some environments, its effects are more subtle and can be mimicked by disciplined administration practices. For this reason, it is omitted from the core set of properties. For similar reasons, all forms of Mutual Exclusion and their associated Inheritance properties are also omitted from the core set.

9. Summary

This report has carefully reexamined the original RBAC model and improved it in a number of areas. The first of these is perfecting the specification by eliminating redundant mappings between model components and correcting errors in the statement of basic properties, such as Dynamic Separation of Duty. The second area is the array of additional properties generated by applying the static and dynamic distinctions to the properties in the original model to state new counterparts. Dynamic Cardinality is an example of an extremely useful property generated in this fashion. A third, and perhaps the most significant area of improvement, is the set of additional properties developed to extend and complete the model. These properties are the Safety properties for Static and Dynamic Separation of Duty, Mutual Exclusion properties, and the Inheritance and Hierarchical Consistency properties for those basic properties affected by role hierarchies.

The report also describes the rationale behind a number of potential areas for enhancement, as a means for understanding the limitations of the revised model. The enhancements identified are the addition of subject/object domains, role attributes, and action histories. The report concludes with the identification of a core set of properties considered the minimum set necessary for a system to be called an RBAC system.

10. References

[1] Role-Based Access Control (RBAC): Features and Motivations, David Ferraiolo et al., Computer Security Applications Conference, December 1995.

[2] Role-Based Access Control Models, Ravi S. Sandhu et al., IEEE Computer, February 1996.

[3] The RBAC Security Policy Model, Virgil Gligor et al., unpublished paper, http://cspa09.ncsl.nist.gov/disk2/rbac/docs/model.ps, December 1995.

[4] Separation of Duty in Role-Based Environments, Richard T. Simon & Mary Ellen Zurko, Proceedings of the Second New Security Foundations Workshop, June 1997.

[5] Role-Based Access Control Protection Profile (PP), National Institute for Standards and Technology, http://csrc.nist.gov/nistpubs/cc/pp/pplist.htm/#RBAC, December 1997.

[6] The Chinese Wall Security Policy, David Brewer & Michael J. Nash, Proceedings IEEE Symposium on Security and Privacy, May 1989, pp.206-214.

[7] Some Conundrums Concerning Separation of Duty, Michael J. Nash & Keith R. Poland, Proceedings IEEE Symposium on Security and Privacy, May 1990, pp.201-207.

[8] Access Rights Administration in Role-based Security Systems, M. Nyanchama & S. Osborn, in Database Security VIII: Status and Prospects, Elsevier Science B.V. North-Holland, 1994

[9] Security Considerations for an Automated Command and Control Information System: Baseline Definition, Frank Mayer, Trusted Information Systems, Report 201, NATO Unclassified, May 1989.