Administrative Scope: A Foundation for Role-Based Administrative Models

JASON CRAMPTON and GEORGE LOIZOU
Birkbeck, University of London

We introduce the concept of administrative scope in a role hierarchy and demonstrate that it can be used as a basis for role-based administration. We then develop a family of models for role hierarchy administration (RHA) employing administrative scope as the central concept. We then extend RHA_4, the most complex model in the family, to a complete, decentralized model for role-based administration. We show that SARBAC, the resulting role-based administrative model, has significant practical and theoretical advantages over ARBAC97. We also discuss how administrative scope might be applied to the administration of general hierarchical structures, how our model can be used to reduce inheritance in the role hierarchy, and how it can be configured to support discretionary access control features.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; I.6.0 [**Computing Methodologies**]: Simulation and Modeling

General Terms: Security, Theory

Additional Key Words and Phrases: Role-based access control, role-based administration, administrative scope, encapsulated range, role hierarchy operation

1. INTRODUCTION

Role-based access control (RBAC) models have been the subject of considerable research in recent years resulting in several important models: the NIST model [Gavrila and Barkley 1998]; the role graph model [Nyanchama and Osborn 1999]; the RBAC96 model [Sandhu et al. 1996]; the graph-based formalism for RBAC [Koch et al. 2002], and the unified NIST RBAC model [Sandhu et al. 2000]. It has been suggested that such models provide an attractive theoretical framework for multi-domain, distributed systems [Joshi et al. 2001]. The features that make RBAC attractive include policy neutrality, principle of least privilege, and ease of management. Gligor [1995] provides a good introduction to the characteristics and advantages of RBAC. The material in this paper is developed in the context of the RBAC96 model. In particular, we assume the

existence of a partially ordered set of roles R (which is visualized as a role hierarchy).

Despite the enthusiasm for RBAC, the use of RBAC principles to manage RBAC systems has been less widely studied. We believe that this is a serious omission: any access control system should be dynamic and changes to such a system must be controlled. In short, we believe that it is important that a sound model be developed for controlling changes to RBAC systems.

Previous approaches to role-based administration have either been centralized, such as the NIST model [Gavrila and Barkley 1998] and the role graph model [Nyanchama and Osborn 1999], or decentralized, such as the RBAC96 model [Sandhu et al. 1996], the ARBAC97 model [Sandhu et al. 1999], and the recent graph-based formalism for RBAC [Koch et al. 2002].

The NIST implementation of RBAC incorporates an Admin Tool and an RBAC database. The latter stores information about user-role, permission-role assignments, and the role hierarchy structure. The former is essentially a collection of functions that determine whether an update to the database is permitted and what the effect of such an update would be. The role graph model includes several algorithms for manipulating the role graph in order to support administrative functions. Neither the NIST nor the role graph model approach to administration is role based.

In contrast, RBAC96 assumes the existence of administrative permissions which are assigned to administrative roles. An important issue when considering administration is the propagation of permissions—the so-called safety problem [Harrison et al. 1976]. However, it is often the case that if administrative (or control) permissions are assigned to subjects, then the safety problem is undecidable. Indeed, Munawer and Sandhu [1999] and Crampton [2002] have shown independently that the safety problem for RBAC96 is undecidable.

ARBAC97 is a model for administration in the context of the RBAC96 model that makes use of the structural properties of the RBAC96 hierarchies rather than relying on administrative permissions. It is widely accepted as the most mature model for role-based administration. ARBAC97 supports decentralized administration and incorporates the functionality provided by the NIST and role graph models.

Although ARBAC97 provides a valuable insight into role-based administration, we believe that it suffers from several problems, which we will consider in detail in Section 7. Informally, we believe that ARBAC97 is not complete as a model, is rather limited in its applicability, and lacks flexibility. (We discuss these issues in detail in Sections 2.4 and 8.) The aim of this paper, therefore, is to develop an alternative approach to role-based administration that is complete, widely applicable, and versatile.

Our approach is built around the concept of administrative scope, which, informally, associates each role in the role hierarchy with a set of roles over which it has control. Administrative scope is developed by considering a simple thought experiment in Section 2.4 and formalized using notation from the theory of partially ordered sets. The graph-based formalism for RBAC defines decentralized administration in terms of operations on graphs and offers some advantages over ARBAC97. Each administrative role is associated

with a *range*, defined by edges in a graph, which is similar to administrative scope.[1]

ARBAC97 was developed as three broadly independent models: URA97, PRA97, and RRA97 which control user-role assignment, permission-role assignment, and role-role assignment (i.e., changes to the role hierarchy), respectively. We believe that some of the problems in ARBAC97 arise from the fact that RRA97, by far the most complex of the three models, was developed after URA97 and PRA97. This has meant that the interaction between the three models is not always well defined.

Therefore, we develop a model for role hierarchy administration (RHA) first, in the belief that it would be easier to then incorporate user-role and permission-role administration. In fact, like the development of several RBAC models, we develop a family of RHA models of increasing complexity, culminating in the RHA_4 model. We believe that RHA_4 is a more robust, flexible, widely applicable, and less complex model than RRA97 (its counterpart in the ARBAC97 model).

We can extend the RHA_4 model in a natural way to SARBAC (scoped administration of role-based access control) in which administrative scope is used to control the assignment of users and permissions to roles. SARBAC offers a number of advantages over ARBAC97, not least because all administrative functions are defined in terms of administrative scope.

In Section 2 we review the essential concepts in partial order theory and RBAC96, and provide a more detailed motivation for our work. In Section 3 we introduce the notion of *administrative scope*, the fundamental concept in RHA. In Section 4 we demonstrate how administrative scope is used to control changes to the role hierarchy and present a family of increasingly complex administrative models. In Section 5 we show how administrative scope can be used to control the assignment of users and permissions to roles. In Section 6 we show how SARBAC can be used in conjunction with the RBAC96 model for a variety of applications. We demonstrate that SARBAC can administer hierarchies in which inheritance is reduced, thereby addressing concerns that a role hierarchy is not necessarily the most suitable structure for modeling access control in a hierarchical enterprise [Moffett and Lupu 1999]. Finally, we discuss how SARBAC might be configured to support discretionary access control in a simple and rather natural way. To our knowledge, ARBAC97 is the most comprehensive and widely accepted model for role-based administration. Therefore, in Section 7 we review the ARBAC97 model. This material is used in Section 8 in which we conduct a detailed comparison of ARBAC97 and SARBAC. This section demonstrates how SARBAC can be used to construct role hierarchies in a decentralized way in a practical role-based system, and also includes a preliminary analysis of the complexity of implementing the two models. To conclude the paper, we summarize the contributions of the paper and discuss future work.

---

[1]A detailed discussion of the graph-based formalism is beyond the scope of this paper. However, we believe that it is not complete in the sense that it does not define how certain administrative operations, such as the assignment of users to administrative roles, are accomplished; nor does it provide a method for constraining the assignment of users to roles, as in ARBAC97.

(a) A role hierarchy   (b) An administrative role hierarchy
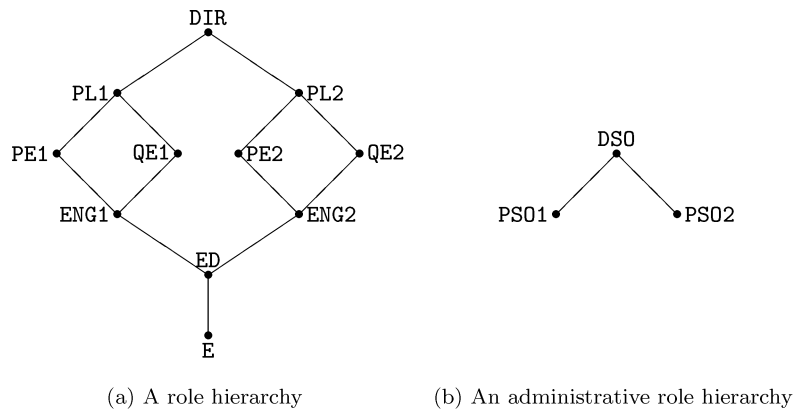
DIR	Director
PL	Project Leader
PE	Production Engineer
QE	Quality Engineer
ENG	Engineer
ED	Engineering Department
E	Employee
DSO	Departmental Security Officer
PSO	Project Security Officer

(c) Legend

Fig. 1.   RBAC96 hierarchies.

2. BACKGROUND

2.1 RBAC96

We define our administrative model in the context of RBAC96 [Sandhu et al. 1996], the most well-known role-based access control model. That is, we assume the existence of a partially ordered set of roles $\langle R, \leqslant \rangle$ which can be visualized as a role hierarchy. Users and permissions are assigned to roles using the binary relations $UA \subseteq U \times R$ and $PA \subseteq P \times R$, where U denotes the set of users and P the set of permissions. That is, u is assigned to r if $(u, r) \in UA$. RBAC96 also assumes the existence of a partially ordered set of administrative roles $\langle AR, \leqslant \rangle$ which is disjoint from R. Figure 1 shows examples of a role hierarchy and an administrative role hierarchy taken from the literature [Sandhu et al. 1999], which will be used as the basis for all further examples in this paper.

2.2 Partial Orders

A more comprehensive introduction to partial orders can be found in the book by Davey and Priestley [1990]. Given a partially ordered set $\langle X, \leqslant \rangle$, we say x *covers* y, denoted by $y \lessdot x$, if $y < x$ and for all $z \in X$, $y \leqslant z < x$ implies $z = y$.

The graph $\langle X, \lessdot \rangle$ is called the *Hasse diagram* of X. (In the context of this paper, the set of roles R is a partial order and the role hierarchy is represented by the Hasse diagram of R.)[2] Given $x, y \in X$, we write $x \parallel y$ if $x \nleqslant y$ and $y \nleqslant x$.

Given $Y \subseteq X$, we say Y is an *antichain* if for all $x, y \in Y$, either $x = y$ or $x \parallel y$. We denote the set of antichains in X by $\mathcal{A}(X)$. The *width* of X is the cardinality of a maximal antichain in X.

An element $y \in Y$ is a *maximal element* (in Y) if for all $z \in Y$, $z \geqslant y$ implies $z = y$. Similarly, $y \in Y$ is a *minimal element* if for all $z \in Y$, $z \leqslant y$ implies $z = y$. We denote the set of maximal elements in Y by $\overline{Y}$ and the set of minimal elements in Y by $\underline{Y}$. For all $Y \subseteq X$, $\overline{Y}$ and $\underline{Y}$ are antichains.

The *lower shadow* of $x \in X$, denoted by Δx, is the set $\{y \in X : y \lessdot x\}$. The *upper shadow* of $x \in X$, denoted by ∇x, is the set $\{y \in X : x \lessdot y\}$.

Given $y \in X$ and $Y \subseteq X$, we define $\downarrow y = \{x \in X : x \leqslant y\}$, $\uparrow y = \{x \in X : x \geqslant y\}$, $\downarrow Y = \{x \in X : \text{there exists } y \in Y \text{ such that } x \leqslant y\}$ and $\uparrow Y = \{x \in X : \text{there exists } y \in Y \text{ such that } x \geqslant y\}$.

A *closed range* in X with endpoints x and y, denoted by $[x, y]$, is the set $\{z \in X : x \leqslant z \leqslant y\}$. An *open range*, denoted by (x, y), is the set $\{z \in X : x < z < y\}$.

Example 2.1.   Using the role hierarchy R in Figure 1,

—{PE1}, {PE1, QE1}, {PE1, QE1, PE2}, and {PE1, QE1, PE2, QE2} are all antichains in the role hierarchy (i.e., {PE1, QE1, PE2, QE2} $\in \mathcal{A}(R)$, for example);

—the width of the role hierarchy is 4;

—ΔENG1 = {ED} and ∇ENG1 = {PE1, QE1};

—$\underline{\text{{ENG1, PE1, QE1, PL1}}}$ = {ENG1} and $\overline{\text{{ENG1, PE1, QE1, PL1}}}$ = {PL1};

—$\downarrow$ENG1 = {ENG1, ED, E} and $\uparrow$ENG1 = {ENG1, PE1, QE1, PL1, DIR};

—[ENG1, PL1] = {ENG1, PE1, QE1, PL1} and (ENG1, PL1) = {PE1, QE1};

—[ENG1, PL1) = {ENG1, PE1, QE1} and (ENG1, PL1] = {PE1, QE1, PL1}.

We will denote the set of roles explicitly assigned to a user u by $R(u)$. That is, $R(u) = \{r \in R : (u, r) \in UA\}$. Hence the set of roles implicitly assigned to u is the set $\downarrow R(u)$. For example, if Anne is only explicitly assigned to the role QE1, then Anne is implicitly assigned to $\downarrow$QE1 = {QE1, ENG1, ED, E}.

2.3 Administrative Operations

We will assume throughout that hierarchy operations are initiated by an administrative role a. We consider the following (*role hierarchy*) *operations*: role insertion, role deletion, edge insertion, and edge deletion. We will denote these by AddRole($a, r, \Delta r, \nabla r$), DeleteRole(a, r), AddEdge(a, c, p), and DeleteEdge(a, c, p), respectively, where Δr is the set of immediate children of (the new role) r, ∇r is the set of immediate parents of r, c the child role, and p the parent role. We assume that an operation does not introduce a cycle into the hierarchy. Specifically, $p \nleqslant c$ and for all $s \in \Delta r$ and all $t \in \nabla r$, $s \ngeqslant t$.

---

[2]Equivalently, the Hasse diagram of X can be thought of as the transitive, reflexive reduction of the graph of the order relation.

(a) Original hierarchy   (b) DeleteEdge(a, r', r'')   (c) AddEdge(a, r', r'')
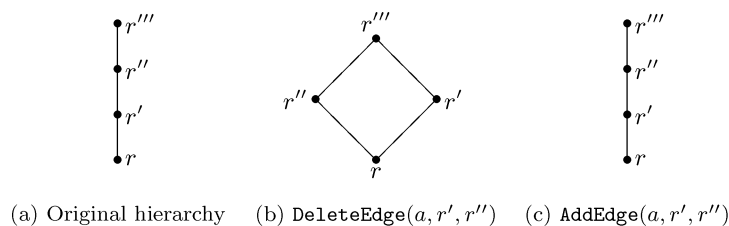
Fig. 2.   Edge operations.

A partial order is a transitive relation. Therefore, if an edge is deleted from the role hierarchy, edges that had previously been implied by transitivity may need to be added. Similarly if an edge is added to the role hierarchy, newly transitive edges may need to be deleted. Figure 2 shows the effect of deleting and then adding the edge (r', r'') in a simple hierarchy consisting of the single chain $r < r' < r'' < r'''$. Hence in Figure 2(b), the edges (r, r'') and (r', r''') (which were transitive edges in the original hierarchy) have been added. In Figure 2(c), edges (r, r'') and (r', r''') (which would be implied by transitivity) have been deleted.[3] We also assume that role deletion will have the side effect of inserting transitive edges that would be lost and that role insertion will delete any transitive edges that arise.

We also include the operations AssignUser(a, u, r), RevokeUser(a, u, r), AssignPermission(a, p, r), and RevokePermission(a, p, r). Each operation has an obvious interpretation: for example, AssignUser(a, u, r) means that administrative role a assigns user u to role r.

Conceptually, an operation consists of a conditional statement and a body, the latter being some sequence of atomic actions as in the protection matrix model, for example. We will not usually concern ourselves with the effect of administrative operations on RBAC96 relations. We believe that such issues are straightforward and would make the discussion of adminstration less focused. In this paper, we will concentrate on the tests in the conditional statement of an administrative operation. For example, we are interested in what conditions a, u, and r must satisfy for the operation AssignUser(a, u, r) to succeed. We will not specify what effect this operation has on the *UA* relation (although it is easy to see that *UA* := *UA* ∪ {(u, r)}, where := denotes "is assigned the value").

2.4 Motivation for a New Administrative Model

Access control models are typically defined using sets, functions, and relations, some of which may change over time. For example, the protection matrix model [Harrison et al. 1976] is defined using the sets *S* (subjects), *O* (objects),

---

[3]We assume, as in RBAC96, that the set of permissions assigned to a role r is given by $\bigcup_{r' \in \downarrow r} P(r')$, where $P(r')$ is the set of permissions assigned to r'. That is, the set of permissions assigned to r'' following the deletion of the edge (r', r'') does not include the permissions assigned to r', but the set of permissions assigned to r''' would.

We note that an alternative strategy in the case of edge deletion would be to destroy the inheritance by not adding transitive edges that would otherwise be lost. However, our interpretation of AddEdge and DeleteEdge means that the two operations are mutually inverse, which seems intuitively reasonable.

A (access modes), and the function $M : S \times O \rightarrow 2^A$ (protection matrix). Of these S, O, and M may change, while A is fixed. We will refer to the latter as a *static component* and the remainder as *dynamic components* of the model. The *state* of a model is a "snapshot" of the dynamic components of the model. (More formally, the state [Harrison et al. 1976] or configuration [Bell and LaPadula 1973] of a model can be regarded as the tuple of all dynamic components in the model.) We will say a model is *complete* if a state transition relation is (or can be) defined (within the framework of the model).

We believe there are two crucial factors that determine the utility and success of any access control model: simplicity and completeness. In particular, the two most enduring paradigms in access control are the Bell–LaPadula model and the protection matrix model, both of which are simple and complete.

In addition, it is desirable that an access control model should be versatile and practical. The versatility and practicality of a model are obviously difficult properties to quantify. Our informal understanding of "versatility" is that the model should be widely applicable, and of "practicality" that the model could be implemented in a real-world system without incurring unacceptable overheads. With this interpretation, the protection matrix model is versatile and practical, while the Bell–LaPadula model arguably only has the second of these qualities.

The ARBAC97 model makes an important contribution to the understanding and modeling of administration in role-based access control. However, we believe there are a number of problems with ARBAC97, which we will discuss in detail in Section 8. Briefly, the RBAC96/ARBAC97 model is not complete, it lacks versatility, and is unlikely to be practical.

Nevertheless, work on ARBAC97 has provided many important insights into role-based administration, not least that it is important to develop a model for role hierarchy administration before considering other administrative functions. This is simply because role hierarchy administration appears to be the most difficult area of role-based administration.

We now consider an example that was used to motivate the development of RRA97. Consider the following sequence of operations on the hierarchy depicted in Figure 1(a): `AddRole(DSO, X, {QE1}, {DIR})`, `AddRole(PSO1, Y, ∅, {PE1})`, and `AddEdge(PSO1, PE1, QE1)`. The cumulative effect of these three operations is to make $Y < X$; this is illustrated in Figure 3 and is considered to be an "anomalous side effect" [Sandhu et al. 1999] of unconstrained changes to the role hierarchy. Therefore, RRA97 seeks to provide a framework in which such side effects cannot occur.[4]

Hence, RRA97 does not permit the operations `AddRole(DSO, X, {QE1}, {DIR})` and `AddRole(PSO1, Y, ∅, {PE1})`, but does permit the operation (`AddEdgePSO1, PE1, QE1`). We believe that the prohibitive nature of RRA97 significantly reduces its utility.

---

[4]It is unclear to us why these side effects should be considered anomalous. For example, in Figure 3, PE1 ≮ QE1, but RRA97 permits the operation `AddEdge(PSO1, PE1, QE1)` which causes PE1to become junior to QE1. In short, there seems to be no qualitative difference between Y being made more junior to X and PE1 being made more junior to QE1.
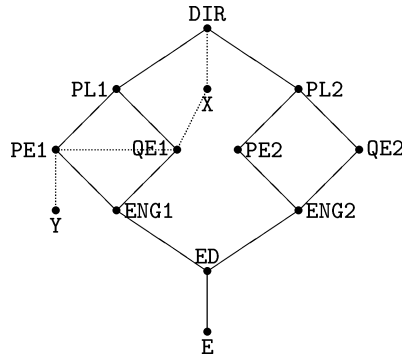
Fig. 3.   An RBAC96 role hierarchy and the effect of undesirable changes.

Informally, our model is motivated by the following two intuitively reasonable suggestions for resolving the problems posed by the hierarchy operations that led to Figure 3. Namely, once role X has been created:

—Remove QE1 from PSO1's administrative range as QE1 is now less than X, a role which is not in PSO1's administrative range. That is, only DSO and above should now be able to administer QE1. In particular, the operation AddEdge(PSO1, PE1, QE1) should not succeed.

—A role r such that $|\nabla r| > 1$ (such as QE1 once X has been inserted into the hierarchy) must be administered by a role which has administrative control over every role in ∇r. In our example, AddEdge(DSO, PE1, QE1) will succeed but AddEdge(PSO1, PE1, QE1) will fail.

These solutions have a similar approach and could be implemented by imposing bounds on the authority of each administrative role. In order to determine such bounds, which may change as the hierarchy changes, we need to find a function $S : R \to 2^R$ such that $S(r)$ models the set of roles that r can administer.

3. ADMINISTRATIVE SCOPE

In this section we will formalize the suggestions we made above, resulting in the definition of administrative scope. In the following section we show how administrative scope provides a natural unit of administration that can be used to impose conditions on hierarchy operations.

Definition 3.1.   The *administrative scope* of a role r is defined as follows:

$$S(r) = \{s \in R : s \leqslant r, \uparrow s \backslash \uparrow r \subseteq \downarrow r\}. \tag{1}$$

Informally, $r \in S(a)$ if every path upwards from r goes through a. That is, any change to r made by a will not have unexpected side effects due to inheritance elsewhere in the hierarchy.

It can be seen that for all $r \in R$, $r \in S(r)$. Hence we define the *strict* administrative scope of r to be $S(r) \backslash \{r\}$, which we will denote by $S^+(r)$. If $s \in S^+(r)$ we say r is an *administrator* of s.

Example 3.2. Consider Figure 1(a). Then $\uparrow$ENG1 $=$ {ENG1, PE1, QE1, PL1, DIR} and $\uparrow$PL1 $=$ {PL1, DIR}; hence $\uparrow$ENG1$\backslash\uparrow$PL1 $=$ {ENG1, PE1, QE1} $\subset$ $\downarrow$PL1. That is, ENG1 $\in$ $\mathcal{S}$(PL1). However, ED $\not\in$ $\mathcal{S}$(PL1), since ENG2 $\in$ $\uparrow$ED, for example, and ENG2 $\not\in$ $\downarrow$PL1. It can easily be seen that $\mathcal{S}$(PL1) $=$ {PL1, PE1, QE1, ENG1}. Hence, PL1 is an administrator of ENG1, PE1, and QE1.

PROPOSITION 3.3. *If $r \lessdot r'$ and $r \in \mathcal{S}^+(a)$ for some $a \in R$, then $r' \in \mathcal{S}(a)$.*

PROOF. Suppose $r' \not\in \mathcal{S}(a)$. Then there exists $r'' \in \uparrow r' \backslash \uparrow a$ such that $r'' \not\in \downarrow a$. That is, $r \lessdot r' \leqslant r''$ and $r'' \not\in \downarrow a$. Hence, $r \not\in \mathcal{S}(a)$, which is a contradiction. □

3.1 Flexibility of Administrative Scope

The administrative scope of a role is determined by the role hierarchy and changes dynamically as the hierarchy changes. (This is in contrast to RRA97, where administration is largely determined by the `can-modify` relation, which in turn imposes restrictions on changes that can be made to the hierarchy.) For example, following the operation AddRole(DIR, X, {QE1}, {DIR}), QE1 $\not\in \mathcal{S}$(PL1). Figure 4 shows how the administrative scope of PL1 changes as edges and roles are added to the hierarchy.

3.2 Decentralization and Autonomy

PROPOSITION 3.4. *If r has an administrator then the set of administrators of r has a unique* minimal *administrator which we refer to as the* line manager *of r.*

PROOF. If r has a single administrator the result follows immediately. Therefore, suppose x and y are minimal administrators of r. (That is, for all administrators z of r, $z \leqslant x$ implies $z = x$ and $z \leqslant y$ implies $z = y$. Hence, $x \not< y$ and $y \not< x$.) Then $r \in \mathcal{S}^+(x)$ and hence $x \in \uparrow r$. Similarly, $r \in \mathcal{S}^+(y)$ and hence by (1)

$$\uparrow r \backslash \uparrow y \subseteq \downarrow y. \tag{2}$$

Since $y \not< x$, $x \not\in \uparrow y$ and hence $x \in \downarrow y$ by 2. Hence $x < y$, which is a contradiction. □

The concept of line manager can be applied to administration of the role hierarchy to ensure maximum decentralization and accountability. That is, we can insist that all changes affecting a role are made by the line manager. This feature could be particularly useful in the management of user-role and permission-role assignments.

4. A FAMILY OF MODELS FOR HIERARCHY ADMINISTRATION

In this section we describe a family of models for hierarchy administration of increasing sophistication (and incurring larger overheads). Unlike in RBAC96, we do not assume the existence of a disjoint set of administrative roles. Table I states the conditions that determine the success or otherwise of a hierarchy operation in the RHA family of models.
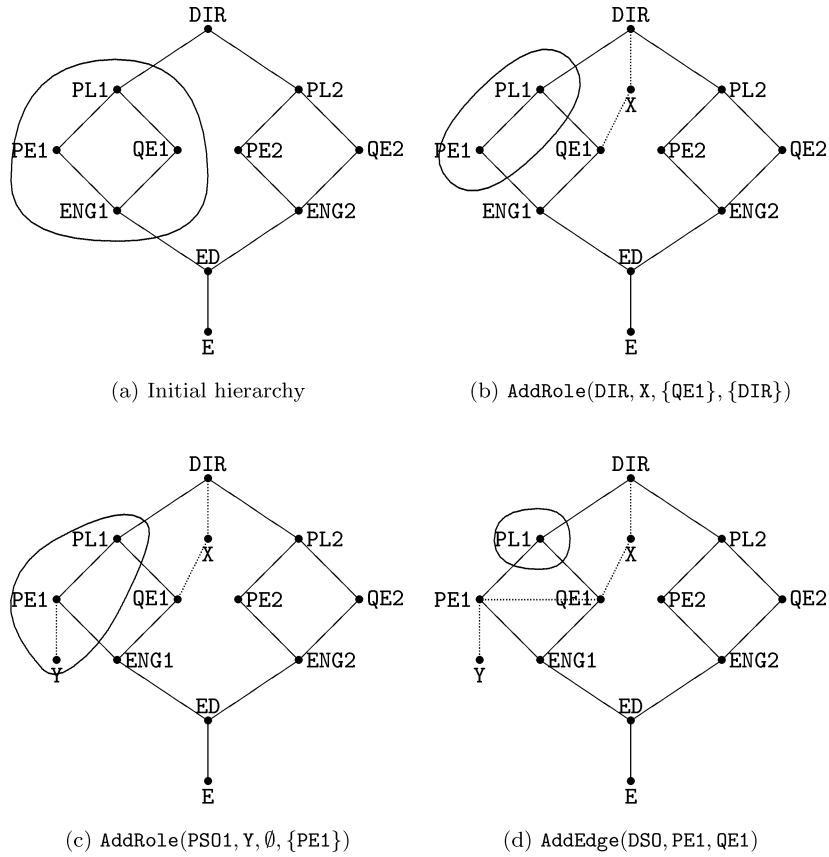
(a) Initial hierarchy

(b) `AddRole(DIR, X, {QE1}, {DIR})`

(c) `AddRole(PSO1, Y, ∅, {PE1})`

(d) `AddEdge(DSO, PE1, QE1)`

Fig. 4. The dynamic nature of administrative scope: The roles inside the closed curve denote the administrative scope of `PL1`.

Table I. Conditions for Success of Hierarchy Operations in RHA

Operation	Conditions
`AddRole(`$a, r, \Delta r, \nabla r$`)`	$\Delta r \subseteq \mathcal{S}^+(a)$
	$\nabla r \subseteq \mathcal{S}(a)$
`DeleteRole(`a, r`)`	$r \in \mathcal{S}^+(a)$
`AddEdge(`a, c, p`)`	$c, p \in \mathcal{S}(a)$
`DeleteEdge(`a, c, p`)`	$c, p \in \mathcal{S}(a)$

4.1 RHA$_1$

RHA$_1$ is the basic model and is applied directly to the role hierarchy. For example, `AddEdge(PL1, PE1, QE1)` and `AddRole(DIR, X, {QE1}, {DIR})` both succeed in RHA$_1$. Clearly RHA$_1$ has the benefit of great simplicity and can be used with the RBAC96 model without the need for any additional relations. Furthermore, it admits the decentralization of administration. For example, the project leader role `PL1` can administer the roles in project 1.

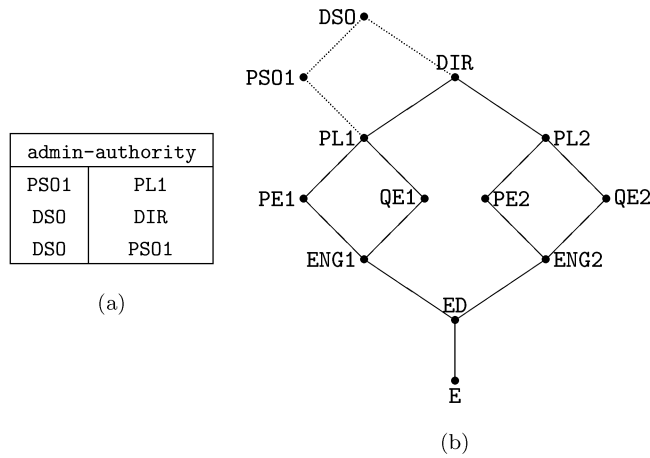| admin-authority	
PSO1	PL1
DSO	DIR
DSO	PSO1

(a)

(b)

Fig. 5. A `admin-authority` relation and the corresponding extended hierarchy.

However, it is unlikely that RHA_1 will provide a sufficiently fine-grained approach to administration and security in many applications. For example, $E \in \mathcal{S}^+(ED)$, but it is probably undesirable that ED should have any control over the hierarchy.

4.2 RHA_2

We can extend RHA_1 by insisting that, in addition to satisfying the conditions in Table I, a must also have appropriate (administrative) permissions assigned to it in order to perform hierarchy operations. RHA_2 can be implemented without introducing additional relations and offers finer granularity than RHA_1 without incurring any significant overheads.

4.3 RHA_3

In this model we introduce a binary relation $\texttt{admin-authority} \subseteq R \times R$. If $(a, r) \in \texttt{admin-authority}$ then a is called an *administrative role*.[5] We also say a *controls* r; we denote the set of roles that a controls by $C(a)$. That is, $C(a) = \{r \in R : (a, r) \in \texttt{admin-authority}\}$.

Informally, we note that `admin-authority` can be visualized as an *extended* hierarchy on the set of roles which includes the original hierarchy. For example, the `admin-authority` relation defined in Figure 5(a) results in the extended hierarchy in Figure 5(b). The elements of `admin-authority` are represented by broken lines. (All subsequent examples in this paper will be visualized using an extended hierarchy rather than explicitly defining the `admin-authority` relation.)

---

[5]We observe that (a, r) could denote a range in the role hierarchy, or an edge in the hierarchy or a tuple in the `admin-authority` relation. However, the interpretation of (a, r) will always be clear from context and the symbols chosen.

We extend the definition of administrative scope in a natural way: namely,

$$\mathcal{S}(a) = \{r \in R : \uparrow r \backslash \uparrow C(a) \subseteq \downarrow C(a)\} \quad \text{and}$$
$$\mathcal{S}^+(a) = \mathcal{S}(a) \backslash C(a),$$

where the evaluation of $\uparrow r$, $\uparrow C(a)$ and $\downarrow C(a)$ takes place in the extended hierarchy. For example, in Figure 5(a), $\mathcal{S}(\texttt{DSO}) = [\texttt{E, DIR}] \cup \{\texttt{PSO1}\}$ and $\mathcal{S}(\texttt{PSO1}) = [\texttt{ENG1, PL1}]$.[6]

There are two self-evident consistency requirements that `admin-authority` must satisfy: for all $(a, r) \in \texttt{admin−authority}$, $a \not< r$; and `admin-authority` is antisymmetric. In addition, we require that the second field in `admin-authority` is unique. In other words each $r \in R$ is controlled by at most one administrative role. This constraint is introduced in order to preserve the line manager feature of the preceding models.

RHA$_3$ provides a level of indirection not available in RHA$_1$ and RHA$_2$ and therefore can be used to implement a far more flexible administrative policy. The `admin-authority` relation states which administrative roles have responsibility for which parts of the role hierarchy. In this sense, it is similar to the `can-modify` relation in RRA97.

Finally we note that RHA$_1$ is a special case of RHA$_3$, where $(r, r) \in \texttt{admin−authority}$ for all $r \in R$.

4.4 RHA$_4$

RHA$_3$ is not complete (unless we assume that the set of administrative roles and the extended hierarchy are static). In short, `admin-authority` should be a dynamic component of the model. In this section, therefore, we consider how RHA$_3$ can be extended to control changes to the `admin-authority` relation. We need to consider when and how the `admin-authority` relation can be updated indirectly (as a side effect of hierarchy operations) and directly (by the actions of administrative roles).

4.4.1 *Direct Updates.*   We assume the existence of two further operations: `AddAdminAuthority`(a, a', r) and `DeleteAdminAuthority`(a, a', r) which respectively add and remove the tuple (a', r) from the `admin-authority` relation. We first observe that removing a tuple from `admin-authority` is equivalent to deleting an edge from the extended hierarchy. Similarly, adding an element to `admin-authority` corresponds to inserting an edge into the extended hierarchy. However, adding a tuple to `admin-authority` may introduce redundancy. For example, it is unnecessary to add $(\texttt{PSO1, PE1})$ to the `admin-authority` relation in Figure 5(a) because $\texttt{PE1} \in \mathcal{S}(\texttt{PSO1})$.

Note also that `AddAdminAuthority`(a, a', r) has the side effect of creating the administrative role a' if $C(a') = \emptyset$ prior to the operation. (Similarly, `DeleteAdminAuthority`(a, a', r) has the side effect of destroying the status of a' as an administrative role if $C(a') = \{r\}$.) Hence, we do not require a new operation to create or destroy administrative roles. Rather, we create a role using the `AddRole` operation and then add tuples to `admin-authority` if we wish to

---

[6]We use ranges because it is more economical than enumerating the elements in the role hierarchy.

Table II. Conditions for Operations on `admin-authority`

Operation	Condition
`AddAdminAuthority`(a, a', r)	$r, a' \in \mathcal{S}(a)$
	$r \notin \mathcal{S}(a')$
`DeleteAdminAuthority`(a, a', r)	$r, a' \in \mathcal{S}(a)$

give that role administrative capabilities. Similarly, to delete an administrative role, we remove all relevant tuples in `admin-authority` and then delete the role using `DeleteRole`. Table II summarizes the conditions that must be satisfied for operations that update the `admin-authority` relation.

4.4.2 *Indirect Updates.* It may be necessary to update `admin-authority` following a role hierarchy operation in order to maintain administrative scope or to eliminate redundancy. We consider the following cases.

`AddRole`$(a, r, \Delta r, \emptyset)$ In this case r has no administrator(s). For example, in Figure 6(a), we see that it is necessary to connect the new role X to the extended hierarchy. The obvious way to do this is to add (PSO1, X) to the `admin-authority` relation. Hence, the operation `AddRole`$(a, r, \Delta r, \emptyset)$ requires that (a, r) be added to the `admin-authority` relation.

`DeleteRole`(a, r) If $(a, r) \in$ admin$-$authority, then it is necessary to reconnect a to the extended hierarchy to preserve a's administrative scope. In this case we add (a, r') to `admin-authority` for all $r' \in \Delta r \cap \mathcal{S}(a)$.[7] For example, in Figure 6(b), we add (PSO1, PE1) and (PSO1, QE1) to `admin-authority`.

`AddEdge`(a, c, p) If $(a, c) \in$ admin$-$authority, it may be possible that the addition of the edge (c, p) makes the tuple (a, c) redundant. In particular, the addition of the edge (c, p) may mean that c is in the strict administrative scope of a. In other words, if $c \in \mathcal{S}^+(a)$ following the insertion of the edge, then we can remove (a, c) from `admin-authority`. For example, in Figure 6(c), we remove (PSO1, PE1) from `admin-authority`.

Similarly, the addition or deletion of a tuple from `admin-authority` may require further updates to `admin-authority`.

`DeleteAdminAuthority`(a, a', r) If r were removed from $\mathcal{S}(a)$ as a result of deleting (a', r), then it is necessary to add (a, r) to `admin-authority` in order to preserve the administrative scope of a. For example, given the `admin-authority` relation in Figure 5(a), DSO can remove (PSO1, PL1) from the relation. In this case it is not necessary to add (DSO, PL1) to `admin-authority` since (DSO, DIR) $\in$ admin$-$authority and hence PL1 $\in \mathcal{S}($DSO$)$.

Remark 4.1. Further research is required to determine whether the set of operations identified above is an exhaustive list of extended hierarchy

---

[7]We take the intersection because there may be elements in Δr that do not belong to $\mathcal{S}(a)$. Note that r' may now occur twice in `admin-authority` as a result of this procedure and hence several more deletions from `admin-authority` may be necessary because of the requirement that each role be controlled by a single role. Detailed algorithms for role hierarchy operations and their effect on the extended hierarchy are beyond the scope of this paper.
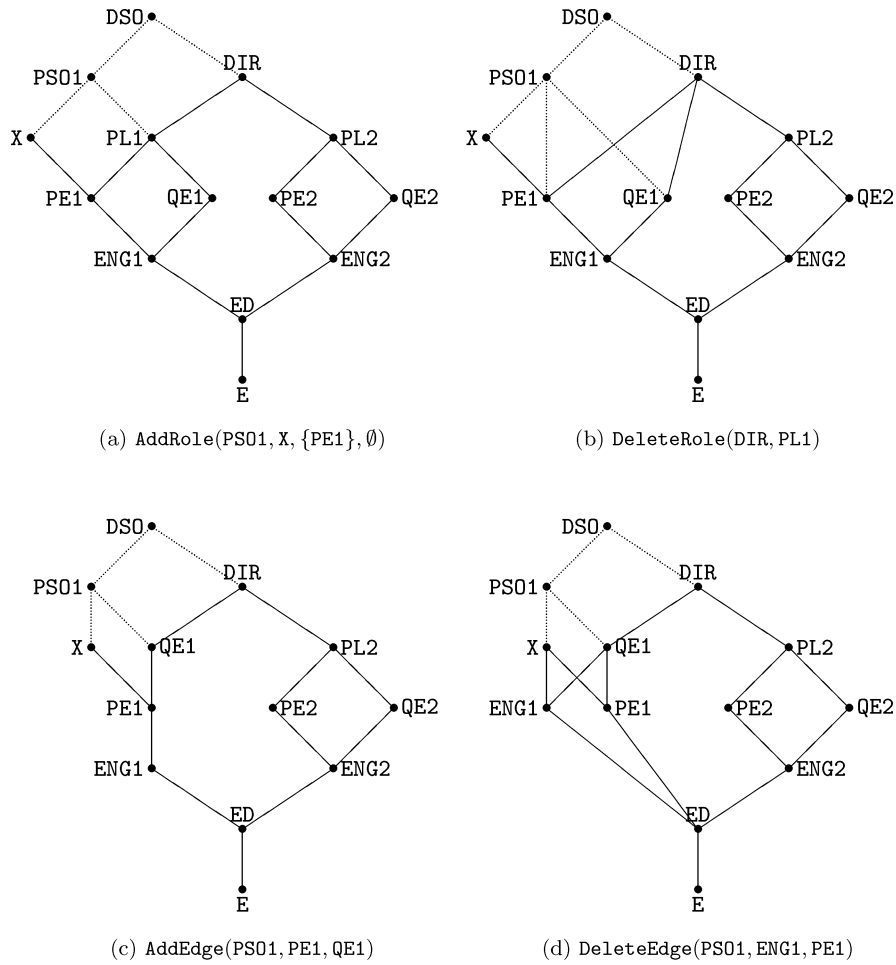
(a) `AddRole(PSO1, X, {PE1}, ∅)`

(b) `DeleteRole(DIR, PL1)`

(c) `AddEdge(PSO1, PE1, QE1)`

(d) `DeleteEdge(PSO1, ENG1, PE1)`

Fig. 6.   Updates to the extended hierarchy.

operations that may have side effects. In particular, the following question is of interest. Let $\overline{R} \subseteq R$ be the set of maximal elements in R. It seems reasonable to insist that for all $r \in \overline{R}$, there exists an administrative role a such that $r \in C(a)$. Does this guarantee that for every `DeleteEdge` operation, no additional edges need to be introduced to the extended hierarchy? In Figure 6(d), for example, the side effect of the role hierarchy operation `DeleteEdge(PSO1, ENG1, PE1)` is to introduce the edges (`ENG1`, `QE1`) and (`ENG1`, `X`), which means that `ENG1` still belongs to $\mathcal{S}($`PSO1`$)$.

5. SARBAC

SARBAC (scoped administration of role-based access control) is intended to be used with RBAC96 as a complete role-based model for administration. It extends the RHA$_4$ model to include administration of user-role and permission-role assignment. In the next section we introduce the idea of a SARBAC

constraint and in Section 5.2 we introduce two relations used to control user-role and permission-role assignment.

5.1 SARBAC Constraints

Let $R' = \{r_1, \ldots, r_k\}$ be a subset of R and let $\bigwedge R'$ denote $r_1 \wedge \ldots \wedge r_k$.

Definition 5.1. A *SARBAC constraint* has the form $\bigwedge C$ for some $C \subseteq R$. A SARBAC constraint $\bigwedge C$ is satisfied by a user u if $C \subseteq \downarrow R(u)$. A SARBAC constraint $\bigwedge C$ is satisfied by a permission p if $C \subseteq \uparrow R(p)$, where $R(p)$ is the set of roles to which p is assigned.

For example, the constraint `PE1` $\wedge$ `QE1` is satisfied by any user assigned to both `PE1` and `QE1`, and by any user assigned to either `PL1` or `DIR`. Note that $\bigwedge \emptyset$ is trivially satisfied by all users and permissions (and corresponds to the URA97 constraint $r \vee \neg r$). We also note that it is sufficient to define a constraint to be $\bigwedge A$, for some $A \in \mathcal{A}(R)$, rather than $\bigwedge C$, where $\mathcal{A}(R)$ is the set of antichains in R. In particular, we have the following result.

PROPOSITION 5.2. *Let $C \subseteq R$ be a SARBAC constraint. Then $\bigwedge C$ is satisfied by a user u if, and only if, $\bigwedge \overline{C}$ is satisfied by u. Similarly, $\bigwedge C$ is satisfied by a permission p if, and only if, $\bigwedge \underline{C}$ is satisfied by p.*

PROOF. Recall that $\overline{C}$ is the set of maximal elements in C.
$\Rightarrow$ The result follows immediately since $C \supseteq \overline{C}$.
$\Leftarrow$ Suppose u satisfies $\bigwedge \overline{C}$. We can assume $\overline{C} \subset C$ (otherwise we are done). Hence let $c \in C \backslash \overline{C}$. Then there exists $c' \in \overline{C}$ such that $c < c'$. Now $c' \in \downarrow R(u)$, since u satisfies $\bigwedge \overline{C}$ by assumption, and hence $c \in \downarrow R(u)$.
The proof for permissions is similar and is omitted. □

5.2 SARBAC Relations

In addition to the `admin-authority` relation from RHA_4, the SARBAC model defines the relations `ua−constraints` $\subseteq R \times \mathcal{A}(R)$ and `pa−constraints` $\subseteq R \times \mathcal{A}(R)$.[8] The purpose of `ua-constraints` and `pa-constraints` is similar to `can-assign` and `can-assignp` in ARBAC97 (see Section 7). Specifically, an administrative role a can assign a user u to a role r provided there exists a tuple $(r, A) \in$ `ua−constraints` such that u satisfies $\bigwedge A$ and r is in the administrative scope of a. (To simplify the presentation we assume that for all $r \in R$ there exists $(r, A) \in$ `ua−constraints` and $(r, A') \in$ `pa−constraints`. We note that in practice, it will be simpler to omit tuples of the form $(r, \emptyset)$ from `ua-constraints`; the assignment of a user u to such a role r succeeds provided r is in the administrative scope of a.) The administrative role a can assign a permission p to a role r provided there exists a tuple $(r, A') \in$ `pa−constraints` such that p satisfies $\bigwedge A'$ and r is in the administrative scope of a. The administrative role a can revoke $(u, r) \in UA$ provided r is in the administrative scope of a.

---

[8]A tuple $(r, A) \in$ `ua−constraints` is equivalent to the role activation rule $r \leftarrow r_1 \wedge \cdots \wedge r_k$ [Yao et al. 2001], where $A = \{r_1, \ldots, r_k\}$.

Table III. Conditions for User-Role Operations to
Succeed in SARBAC

Operation	Conditions
AssignUser(a, u, r)	$r \in \mathcal{S}(a)$
	$(r, \bigwedge C) \in$ ua−constraints
	u satisfies $\bigwedge C$
RevokeUser(a, u, r)	$r \in \mathcal{S}(a)$

(a)   (b)
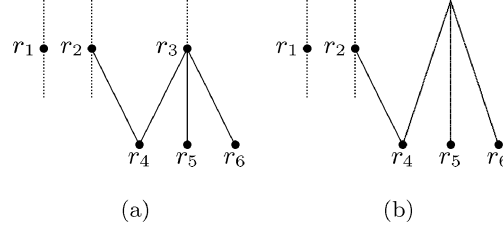
Fig. 7. Role deletion and SARBAC constraints: If r_3 is deleted then the constraint $r_1 \wedge r_2 \wedge r_3$ is replaced by $\bigwedge \overline{\{r_1, r_2, r_4, r_5, r_6\}} = r_1 \wedge r_2 \wedge r_5 \wedge r_6$.

Similarly, a can revoke $(p, r) \in PA$ provided r is in the administrative scope of a.

5.3 Updates to SARBAC Relations

There are two cases to consider: direct updates in which an administrative role makes a change to a SARBAC relation, and indirect updates which occur as a result of a hierarchy operation.

5.3.1 *Direct Updates.* An administrative role a can add a tuple (r, A) to (or delete a tuple from) ua-constraints or pa-constraints provided $r \in \mathcal{S}(a)$ and $A \subseteq \mathcal{S}(a)$.

5.3.2 *Indirect Updates to* ua-constraints. We consider the effect of hierarchy operations on ua-constraints following a hierarchy operation.

AddEdge(a, c, p). For all $(r, A) \in$ ua−constraints such that $c, p \in A$, we replace (r, A) by $(r, A \backslash \{c\})$.

DeleteEdge(a, c, p). For all $(r, A) \in$ ua−constraints such that $p \in A$, we replace (r, A) by $(r, A \cup \{c\})$.

AddRole$(a, r, \triangle r, \triangledown r)$. We may need to update some constraints because the addition of r will result in the creation of transitive edges between elements in $\triangle r$ and $\triangledown r$. Therefore, the procedure outlined above for AddEdge may need to be employed.

DeleteRole(a, r). For all $(r', A) \in$ ua−constraints such that $r \in A$, we replace (r', A) by $(r', \overline{A \cup \triangle r \backslash \{r\}})$. A schematic motivation for this procedure is shown in Figure 7. (Note that role deletion requires that for all $c \in \triangle r$ and all $p \in \triangledown r$, the edge (c, p) is added to the hierarchy. However, no further changes are required to ua-constraints because c and p formed a chain prior to the deletion of r and hence could not have belonged to any SARBAC constraint.)
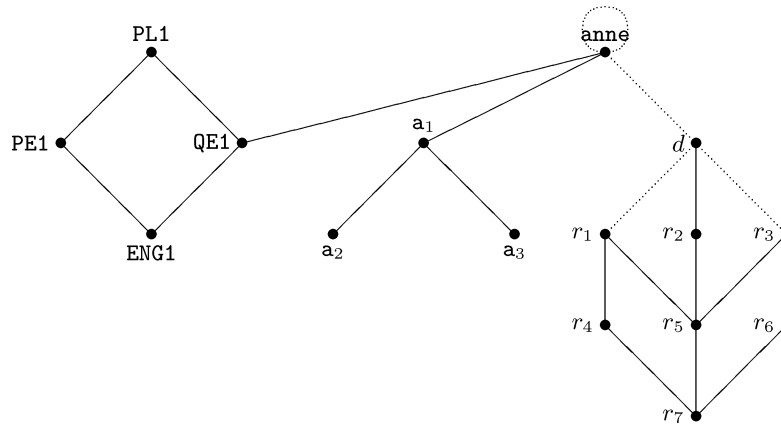
Fig. 8.   Private hierarchies for discretionary access control.

5.3.3 *Indirect Updates to* `pa-constraints`. Clearly, similar procedures must be applied to the `pa-constraints` relation following a hierarchy operation, except that if r is deleted from the hierarchy, then for all $(r', A) \in$ `pa−constraints` such that $r \in A$, we replace (r', A) by $(r', \underline{A \cup \nabla r \setminus \{r\}})$.

6. APPLICATIONS OF SARBAC

In this section we show how SARBAC can be used to support discretionary access control and to reduce the level of inheritance in the role hierarchy.

6.1 Simulating Discretionary Access Control

Discretionary access control is essentially characterized by ownership of objects by users, and by users having permissions to grant and revoke access to objects they own. It is well known that, in general, the security properties of a discretionary access control system cannot necessarily be established [Harrison et al. 1976].

There have been several attempts to establish a correspondence between features of discretionary access control and role-based access control [Barkley 1997; Friberg and Held 1997; Hua and Osborn 1998; Sandhu and Munawer 1998]. The most ambitious of these attempts, by Osborn et al. [2000], tries to address the problem of ownership in RBAC. Unfortunately, the construction presented therein involves the creation of at least three administrative roles and one normal role as well as eight permissions for each object in the system. The authors do acknowledge that discretionary access control appears to be more complex to simulate in RBAC than mandatory access control.

We adopt a different approach to simulating discretionary access control within a role-based framework. Figure 8 shows a fragment of the usual hierarchy connected to a hierarchy rooted at the role `anne`. The intuition here is that user Anne will be (the only user) assigned to the personal role `anne`. The role `anne` is also an administrative role. Hence the hierarchy rooted at role `anne` can be regarded as a private hierarchy which can only be administered by user Anne.

In the ensuing discussion, we will assume that `Anne` and `anne` are effectively identical; we will refer to `Anne` throughout.

For illustrative purposes we have divided `Anne`'s private hierarchy into two subhierarchies. The set of roles $\{a_1, a_2, a_3\}$ are all inherited by `Anne`. The only role that can assign users to roles in this subhierarchy is `Anne` herself. The intended interpretation is that this subhierarchy is to be administered solely by `Anne`.

In contrast, the set of roles $\{r_1, \ldots, r_7\}$ form a subhierarchy that can be partly administered by d. In particular, $C(d) = \{r_1, r_3\}$ and $\mathcal{S}(d) = \{r_1, r_3, r_4, r_6\}$. (Note that $r_2, r_5, r_7 \notin \mathcal{S}(d)$.) Hence, d can assign users to the roles r_1, r_3, r_4, and r_6, for example. In other words, `Anne` can delegate certain administrative functions by assigning a user to the role d. Note also that `Anne` is not implicitly assigned (i.e., by inheritance) to any of the roles in the set $\{r_1, \ldots, r_7\}$. Of course, `Anne` can assign herself to any of the roles in this set if she wishes.

6.2 Reducing Hierarchy Inheritance

It is generally assumed that a role hierarchy incorporates two types of inheritance. Firstly, if a permission p is assigned to a role r, then p is available to all roles in $\uparrow r$. Secondly, if a user u is assigned to a role r, then u can activate any role in $\downarrow r$.

Goh and Baldwin [1998] and Moffett and Lupu [1999] have observed that there may be situations where this may not be appropriate. In particular, this aspect of the model does not necessarily accurately reflect the access control requirements of most enterprises. In Figure 1(a), is it appropriate, for example, that `DIR` has the permissions of `PE1`? Any user assigned to the role `DIR` presumably has little or no day-to-day responsibility for, or competency to perform, the activities expected of a production engineer. These issues are considered in some detail by Goh and Baldwin [1998] in their discussion of *subsidiarity*.

An advantage of using SARBAC instead of ARBAC97 as the administrative model is that the role hierarchy is not required to contain ranges. This is because the extended hierarchy can fill gaps in the role hierarchy using edges in the `admin-authority` relation; hence the number of edges can be reduced. In other words, administration of the role hierarchy is still possible using SARBAC, even if there are no ranges in the role hierarchy. Figure 9 shows an extended hierarchy which is similar to our running example. However, we have reduced the number of edges in the hierarchy. In particular, we have removed the edges (`PE1`, `PL1`), (`QE1`, `PL1`), (`PE2`, `PL2`), and (`QE2`, `PL2`). (Note that ARBAC97 cannot be used to administer this hierarchy.)

In other words, we have restricted the permissions of the `DIR` role to those of the senior roles. This corresponds more accurately with the deployment of responsibilities in a real-world enterprise than in the original hierarchy. Furthermore, `Claire` is assigned to the `DSO` role which means that she can administer junior roles in the hierarchy although she is not actually assigned to those roles. It has been argued [Sadighi Firozabadi and Sergot 1999] that this distinction between having the authority to assign a permission (or role) and being assigned a permission (or role) is an important and largely ignored
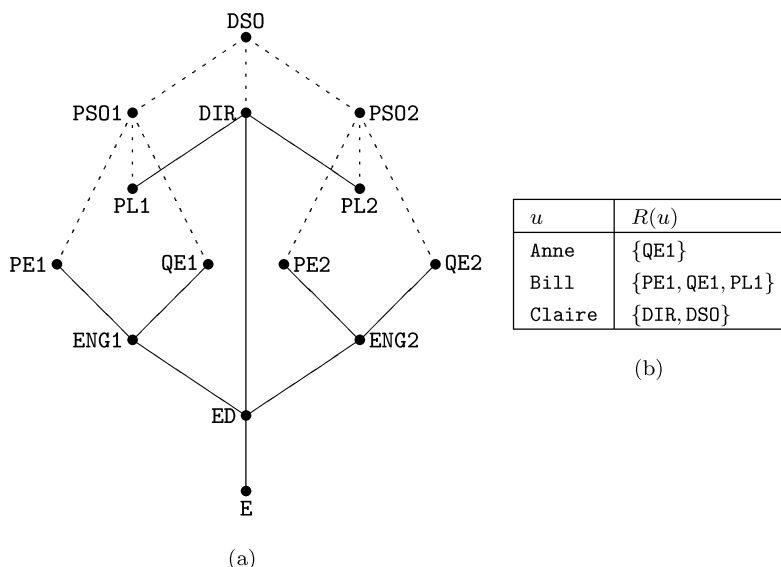
u	$R(u)$
Anne	$\{\texttt{QE1}\}$
Bill	$\{\texttt{PE1}, \texttt{QE1}, \texttt{PL1}\}$
Claire	$\{\texttt{DIR}, \texttt{DSO}\}$

(b)

(a)

Fig. 9.   Applying SARBAC to a reduced role hierarchy.

area in access control. Note also that it is necessary for `Bill` (who is assigned to `PL1`) to be assigned explicitly to `PE1` and `QE1` in order for him to make use of the permissions available to roles in the project for which he is responsible. This corresponds more accurately with the spirit of least privilege.

7. ARBAC97

The SARBAC model is an alternative model to ARBAC97 for role-based administration. In order to make a comparison between the two models, we first summarize the URA97, PRA97, and RRA97 models.

7.1 URA97

The assignment of users to roles is controlled by the URA97 submodel of ARBAC97. Informally, $\texttt{AssignUser}(a, u, r)$ succeeds provided u's existing role assignments satisfy certain requirements and a is permitted to assign users to r. (If the operation succeeds, then the *UA* relation is updated.) These requirements are expressed as a URA97 constraint.

Formally, let $r \in R$. Then r and $\neg r$ are URA97 constraints. Let c_1 and c_2 be URA97 constraints. Then $c_1 \wedge c_2$ and $c_1 \vee c_2$ are URA97 constraints. The conditions for a user u to satisfy a URA97 constraint are given in Table IV. We denote the set of URA97 constraints by $\mathcal{C}$ and the set of ranges in R by $\mathcal{R}$. Note that all users satisfy the constraint $r \vee \neg r$.[9]

---

[9]Technically, we believe (c) (where c is a URA97 constraint) should also be a URA97 constraint which is satisfied by u provided u satisfies c, otherwise we have no way of expressing a constraint of the form $(c_1 \vee c_2) \wedge c_3$, for example (assuming the evaluation of $\wedge$ takes precedence over $\vee$). We also note that Sandhu et al. [1999] used the notation $\bar{r}$ rather than $\neg r$.

Table IV. Satisfaction of URA97 Constraints

Constraint	Conditions
r	$r \in \downarrow R(u)$
$\neg r$	$r \notin \downarrow R(u)$
$c_1 \wedge c_2$	u satisfies c_1 and u satisfies c_2
$c_1 \vee c_2$	u satisfies c_1 or u satisfies c_2

Table V. Conditions for User Assignment Operations to Succeed in URA97

Operation	Condition
`AssignUser`(a, u, r)	$\exists (a', c, G) \in \texttt{can-assign}$ $a' \in \downarrow a$ u satisfies c $r \in G$
`RevokeUser`(a, u, r)	$\exists (a', G) \in \texttt{can-revoke}$ $a' \in \downarrow a$ $r \in G$

URA97 defines the relations $\texttt{can-assign} \subseteq AR \times \mathcal{C} \times \mathcal{R}$ and $\texttt{can-revoke} \subseteq AR \times \mathcal{R}$ to control the assignment and revocation of roles to users. Table V states the conditions that must be satisfied for user-role assignment operations to succeed.

Remark 7.1.   URA97 distinguishes between *weak* and *strong* revocation. This distinction is only meaningful if $R(u)$ is not an antichain [Crampton 2002]. We will discuss this further in Section 8.

Figure 11 shows examples of the two URA97 relations [Sandhu et al. 1999]. For illustrative purposes we assume $(\texttt{Anne}, \texttt{QE1}), (\texttt{Bill}, \texttt{PL1}) \in UA$. Hence, any role in $\uparrow\texttt{PSO1}$ can assign $\texttt{Anne}$ to $\texttt{PE1}$ (by the first row of $\texttt{can-assign}$). Furthermore, if $\texttt{Anne}$ is not assigned to $\texttt{PL2}$, any role in $\uparrow\texttt{DSO}$ can assign her to $\texttt{PL1}$ (by the fourth row of $\texttt{can-assign}$). Conversely, any role in $\uparrow\texttt{PSO1}$ can revoke the assignment $(\texttt{Anne}, \texttt{QE1})$ (by the first row of $\texttt{can-revoke}$).

7.2 PRA97

The structure and semantics of the PRA97 relations $\texttt{can-assignp}$ and $\texttt{can-revokep}$ are identical to their counterparts in URA97, except that a permission p satisfies the PRA97 constraint r if $r \in \uparrow R(p)$ and satisfies $\neg r$ if $r \notin \uparrow R(p)$, where $R(p)$ denotes the set of roles assigned to p. Permission-role assignment is broadly analogous to user-role assignment in both ARBAC97 and SARBAC. We will refer to permission-role assignment explicitly only when it differs from user-role assignment.

7.3 RRA97

The fundamental idea in RRA97 is that of an encapsulated range. The following definition is due to Sandhu et al. [1999].

Table VI. Conditions for Hierarchy Operations to Succeed in RRA97

Operation	Conditions	
$\texttt{AddRole}(a, r, \{c\}, \{p\})$	$\exists (a', (w, z)) \in \texttt{can-modify}$ such that $a' \in \uparrow a$	$w \leqslant c \leqslant p \leqslant z$ $I(c) = I(p)$
$\texttt{AddRole}(a, r, \{c\}, \{p\})$		$w \leqslant c \leqslant p \leqslant z$ $I(c) = (w, p)$
$\texttt{AddRole}(a, r, \{c\}, \{p\})$		$w \leqslant c \leqslant p \leqslant z$ $I(p) = (c, z)$
$\texttt{DeleteRole}(a, r)$		$r \in (w, z)$
$\texttt{AddEdge}(a, c, p)$	No operation can violate the encapsulation of any authority range	$w \leqslant c, p \leqslant z$ $I(c) = I(p)$
$\texttt{AddEdge}(a, c, p)$		$w \leqslant c, p \leqslant z$ (c, z) is an authority range
$\texttt{AddEdge}(a, c, p)$		$w \leqslant c, p \leqslant z$ (w, p) is an authority range
$\texttt{DeleteEdge}(a, c, p)$		(c, p) is not an authority range $w \leqslant c, p \leqslant z$

Definition 7.2.   A range (x, y) is said to be *encapsulated* if for all $w \in (x, y)$, and for all $z \notin (x, y)$,

$$z > w \quad \text{if, and only if, } z > y, \quad \text{and} \tag{3}$$

$$z < w \quad \text{if, and only if, } z < x. \tag{4}$$

Informally, an encapsulated range is a self-contained subhierarchy in the role hierarchy with all external edges passing through one of the end points of the range. (E, ED), (ENG1, PL1), and (ED, DIR) are examples of encapsulated ranges in Figure 1(a).

The $\texttt{can-modify} \subseteq AR \times \mathcal{E}(R)$ relation, where AR is the set of administrative roles and $\mathcal{E}(R)$ is the set of encapsulated ranges in R, determines the encapsulated ranges over which administrative roles can act. Figure 10(a) shows a typical example of the $\texttt{can-modify}$ relation [Sandhu et al. 1999]. An encapsulated range that appears in the $\texttt{can-modify}$ relation is called an *authority range*. RRA97 also requires that for any two authority ranges, they either be disjoint or one be entirely contained in the other.

Hence, for every role $r \in R$, there is a unique smallest authority range to which r belongs. This is called the *immediate authority range* of r, which we will denote by $I(r)$. For example, given the $\texttt{can-modify}$ relation in Figure 10(a), the immediate authority range of PE1 is (ENG1, PL1), not (ED, DIR).

Table VI shows the conditions that need to be satisfied in order to perform a hierarchy operation. The second column indicates the conditions that must be satisfied by all hierarchy operations. Multiple rows for a given operation indicate that there are several different sets of criteria that the arguments of the operation can satisfy. Note that RRA97 requires that a new role has precisely one parent role p and one child role c such that $c \leqslant p$. Furthermore, $\texttt{DeleteRole}(a, r)$ fails if r is the end-point of any range in any ARBAC97 relation.

Remark 7.3.   We note that Definition 7.2 implies no range can be encapsulated since $y \notin (x, y)$, $y > w$ for all $w \in (x, y)$ but $y \not> y$. Hence conditions

admin-authority	
PS01	PL1
PS02	PL2
DS0	PS01
DS0	PS02
DS0	DIR

can-modify	
PS01	(ENG1, PL1)
PS02	(ENG2, PL2)
DS0	(ED, DIR)

(a)

(b)

Fig. 10.   ARBAC97 and SARBAC relations for hierarchy administration.

can-assign		
PS01	ED	[ENG1, PL1)
PS02	ED	[ENG2, PL2)
DS0	ED ∧ ¬PL1	[PL2, PL2]
DS0	ED ∧ ¬PL2	[PL1, PL1]

(a)

can-revoke	
PS01	[ENG1, PL1)
PS02	[ENG2, PL2)
DS0	[ED, DIR]

(b)

ua-constraints	
ENG1	{ED}
PE1	{ED}
QE1	{ED}
ENG2	{ED}
PE2	{ED}
QE2	{ED}
PL1	{PE1}
PL1	{QE1}
PS01	{PL1}

(c)

Fig. 11.   ARBAC97 and SARBAC relations for administration of user-role assignment.

(3) and (4) should be replaced by

$$z \;>\; w \quad \text{if, and only if, } z \geqslant y \quad \text{ and} \tag{5}$$

$$z \;<\; w \quad \text{if, and only if, } z \leqslant x, \tag{6}$$

respectively.

8. COMPARISON OF SARBAC AND ARBAC97

In this section we first consider some concrete examples of administrative operations and whether they would succeed using ARBAC97 and SARBAC. We then compare the two models using several different criteria including completeness, simplicity, practicality, and versatility.

Figure 10 shows an example of the can-modify relation [Sandhu et al. 1999] and of the admin-authority relation. Figure 11 shows examples of the can-assign and can-revoke relations [Sandhu et al. 1999] and of the ua-constraints relation.

Note that the tuples in can-assign and ua-constraints are divided by a horizontal line. In the case of can-assign, the tuples below the line cannot be expressed within the SARBAC framework. In the case of ua-constraints, the tuples above the line are used to approximate the behavior of the can-assign relation. For example, since $\mathcal{S}(\text{PS01}) = \{\text{ENG1}, \text{PE1}, \text{QE1}, \text{PL1}\}$, PS01 can assign a user to any of these roles, provided the user satisfies the constraint ED. The tuples below the line are used to illustrate features of SARBAC that are different

from ARBAC97. The tuples (PL1, {QE1}) and (PL1, {PE1}) illustrate alternative criteria that a user can satisfy in order to be assigned to PL1 (and would be expressed using the constraint PE1 ∨ QE1 in ARBAC97). The tuple (PSO1, {PL1}) illustrates how the assignment of users to administrative roles can be controlled.

Table VII provides a comparison of the success or otherwise of ARBAC97 and SARBAC administrative operations applied to the hierarchy in Figure 1(a) using the administrative relations in Figures 10 and 11. We assume that the effect of the operations is not cumulative. That is, each operation is applied in turn to the hierarchy in Figure 1(a). We will use this table to illustrate several points in the course of the discussion that forms the remainder of this section. For illustrative purposes, we assume (as before) that (Anne, QE1), (Bill, PL1) ∈ *UA*.

It is immediately obvious from Table VII that SARBAC is a more permissive model than ARBAC97. In particular, the requirement that encapsulated ranges be preserved significantly reduces the number of legitimate hierarchy operations in ARBAC97. This observation can be formalized in the following two propositions.

PROPOSITION 8.1.   *A range* (x, y) *is encapsulated if, and only if,*

$$\uparrow(x, y)\backslash\uparrow y = (x, y) \quad and \tag{7}$$

$$\downarrow(x, y)\backslash\downarrow x = (x, y). \tag{8}$$

PROOF.   This proof assumes the characterization of encapsulated range given in Remark 7.3.

⇒ Suppose for all $z \notin (x, y)$ and for all $w \in (x, y)$ we have $z > w$ if, and only if, $z \geqslant y$. We now prove that $\uparrow(x, y)\backslash\uparrow y \subseteq (x, y)$. Let $a \in \uparrow(x, y)\backslash\uparrow y$. Then there exists $b \in (x, y)$ such that $b \leqslant a$ and $y \nleqslant a$. Since (x, y) is encapsulated, $a \in (x, y)$ (otherwise we have $a \notin (x, y)$ such that $a \geqslant b$ for some $b \in (x, y)$ and $y \ngeqslant a$). Clearly, $(x, y) \subseteq \uparrow(x, y)\backslash\uparrow y$ and hence we have $\uparrow(x, y)\backslash\uparrow y = (x, y)$. The corresponding proof for $\downarrow(x, y)\backslash\downarrow y$ is similar; we omit the details.

⇐ Suppose $\uparrow(x, y)\backslash\uparrow y = (x, y)$. Let $w \in (x, y)$ and $z \notin (x, y)$ with $z > w$. Hence $z \in \uparrow(x, y)$. Since $z \notin (x, y)$, $z \in \uparrow y$ and hence $z \geqslant y$. The corresponding proof for $\downarrow(x, y)\backslash\downarrow y$ is similar; we omit the details. ☐

PROPOSITION 8.2.   *If* (x, y) *is an authority range, then* $(x, y) \subseteq \mathcal{S}^+(y)$.

PROOF.   Suppose $z \in (x, y)$. Then $x < z < y$ and hence $\uparrow x \supset \uparrow z \supset \uparrow y$. Therefore, $\uparrow z\backslash\uparrow y \subseteq \uparrow(x, y)\backslash\uparrow y = (x, y)$ by (7), and $(x, y) \subseteq \downarrow y\backslash\{y\}$. That is, $z \in \mathcal{S}^+(y)$. ☐

8.1 Completeness

The RBAC96/ARBAC97 model includes the following components: *R*, *RH*, *AR*, *ARH*, *UA*, *PA*, can-assign, can-revoke, can-assignp, can-revokep, and can-modify. The dynamic components are *R*, *RH*, *UA*, and *PA*. However, it is unlikely to be appropriate that can-assign, for example, is a static relation. For example, if a new role is added to the hierarchy, how can constraints be imposed on the assignment of users (and permissions) to that role? It is also impossible

Table VII. Operations in ARBAC97 and SARBAC (with reference to Figures 1, 10, and 11)

ARBAC97	SARBAC	
×	$\texttt{AddRole}(\texttt{DSO}, \texttt{X}, \{\texttt{QE1}\}, \{\texttt{DIR}\})$	√
encapsulation of authority range $(\texttt{ENG1}, \texttt{PL1})$ is violated	$\mathcal{S}(\texttt{PSO1}) = \{\texttt{PE1}, \texttt{PL1}\}$	
×	$\texttt{AddRole}(\texttt{PSO1}, \texttt{Y}, \emptyset, \{\texttt{PE1}\})$	√
encapsulation of authority range $(\texttt{ENG1}, \texttt{PL1})$ is violated (and also Y has no child role)	$\texttt{Y} \in \mathcal{S}(\texttt{PSO1})$	
×	$\texttt{AddRole}(\texttt{PSO1}, \texttt{Z}, \{\texttt{PE1}, \texttt{QE1}\}, \emptyset)$	√
encapsulation of authority range $(\texttt{ENG1}, \texttt{PL1})$ is violated (and also Z has two child roles and no parent role)	$\texttt{Z} \in \mathcal{S}(\texttt{PSO1}); (\texttt{PSO1}, \texttt{Z}) \in \texttt{admin-authority}$ (as a side effect of the $\texttt{AddRole}$ operation)	
×	$\texttt{AddRole}(\texttt{PSO1}, \texttt{W}, \{\texttt{ED}\}, \{\texttt{PE1}\})$	×
encapsulation of authority range $(\texttt{ENG1}, \texttt{PL1})$ is violated	$\texttt{ED} \notin \mathcal{S}(\texttt{PSO1})$	
×	$\texttt{AddRole}(\texttt{DSO}, \texttt{W}, \{\texttt{ED}\}, \{\texttt{PE1}\})$	√
encapsulation of authority range $(\texttt{ENG1}, \texttt{PL1})$ is violated	$\texttt{W} \in \mathcal{S}(\texttt{PSO1})$	
×	$\texttt{AddRole}(\texttt{DSO}, \texttt{PSO3}, \emptyset, \emptyset)$	√
PSO3 does not have a child or parent (although the addition of administrative roles is not part of the ARBAC97 model)	$\texttt{PSO3} \in \mathcal{S}(\texttt{DSO})$	
×	$\texttt{DeleteRole}(\texttt{PSO1}, \texttt{ENG1})$	√
$(\texttt{ENG1}, \texttt{PL1}) \in \texttt{can-modify}$	$\mathcal{S}(\texttt{PSO1}) = \{\texttt{PE1}, \texttt{QE1}, \texttt{PL1}\}$	
√	$\texttt{DeleteRole}(\texttt{PSO1}, \texttt{PE1})$	√
	$\mathcal{S}(\texttt{PSO1}) = \{\texttt{ENG1}, \texttt{QE1}, \texttt{PL1}\}$	
×	$\texttt{DeleteRole}(\texttt{PSO1}, \texttt{PL1})$	√
$(\texttt{ENG1}, \texttt{PL1}) \in \texttt{can-modify}$	$C(\texttt{PSO1}) = \{\texttt{PE1}, \texttt{QE1}\}; \mathcal{S}(\texttt{PSO1}) = \{\texttt{ENG1}, \texttt{QE1}, \texttt{PE1}\}$	
×	$\texttt{DeleteEdge}(\texttt{DSO}, \texttt{ED}, \texttt{ENG1})$	√
encapsulation of authority range $(\texttt{ENG1}, \texttt{PL1})$ is violated (since transitive edges $(\texttt{ED}, \texttt{PE1})$ and $(\texttt{ED}, \texttt{QE1})$ must be added to hierarchy)		
√	$\texttt{DeleteEdge}(\texttt{PSO1}, \texttt{ENG1}, \texttt{QE1})$	√
×	$\texttt{AddEdge}(\texttt{PSO1}, \texttt{ENG1}, \texttt{PE2})$	×
there does not exist $(a, G) \in \texttt{can-modify}$ such that $a \in {\downarrow}\texttt{PSO1}$ and $\texttt{PE2} \in G$	$\texttt{PE2} \notin \mathcal{S}(\texttt{PSO1})$	
×	$\texttt{AddEdge}(\texttt{DSO}, \texttt{ENG1}, \texttt{PE2})$	√
encapsulation of authority range $(\texttt{ENG2}, \texttt{PL2})$ is violated	$\mathcal{S}(\texttt{PSO1}) = \{\texttt{PE1}, \texttt{QE1}, \texttt{PL1}\}$	
√	$\texttt{AssignUser}(\texttt{PSO1}, \texttt{Anne}, \texttt{PE1})$	√
√	$\texttt{RevokeUser}(\texttt{PSO1}, \texttt{Anne}, \texttt{QE1})$	√
?	$\texttt{AssignUser}(\texttt{DSO}, \texttt{Bill}, \texttt{PSO1})$	√

A tick indicates the operation would be permitted; a cross indicates the operation would not be permitted. Immediately below most operations is a comment indicating either why the operation fails or the effect of the operation on administrative relations. Note that, given the role hierarchy in Figure 1(a) and the $\texttt{admin-authority}$ relation in Figure 10, $\mathcal{S}(\texttt{PSO1}) = \{\texttt{ENG1}, \texttt{PE1}, \texttt{QE1}, \texttt{PL1}\}$.

to delete a role that is the end-point of a range in an ARBAC97 component. In short, we believe that the ARBAC97 components should be dynamic.

The RBAC96/SARBAC model, however, includes the following components: R, RH, UA, PA, ua-constraints, pa-constraints, and admin-authority. We have ensured that each of these is dynamic. In particular, the deletion of a role in a SARBAC relation is permissible, the relation being updated in a well-defined manner.

Furthermore, the set of administrative roles is static in ARBAC97 and the assignment of users to administrative roles is not considered. (We note that it is probably possible to extend the ARBAC97 relations to include administrative roles. For example, can-assign could contain the tuple (DSO, PL1, [PSO1, PSO1]). The operation AssignUser(DSO, Bill, PSO1) in the last row of Table VII would then succeed in ARBAC97.) SARBAC defines a set of roles R, where $a \in R$ is an administrative role if there exists $r \in R$ such that $(a, r) \in$ admin-authority. This approach means that the assignment of users (and permissions) to roles (including administrative roles) can be performed within the same framework.

8.2 Simplicity

Obviously the simplicity of a model is a somewhat subjective quality, but we believe that the relationship between SARBAC relations and the role hierarchy is more intuitive in SARBAC than in ARBAC97. Administrative scope is a simple notion to describe informally. Furthermore, we believe that the extended hierarchy provides a useful way of visualizing administration. In short, there is an intuitive and immediate interpretation of the admin-authority relation that is lacking in the can-modify relation.

Possibly the most compelling evidence to support our claim for the greater simplicity of SARBAC is the conditions that must prevail for an administrative operation to succeed. If we compare the conditions for hierarchy operations in Table I (SARBAC) with Table VI (ARBAC97) and for user-role assignment in Tables III (SARBAC) and V (ARBAC97), it is apparent that the tests that need to be applied are both simpler and have a much greater uniformity in SARBAC. We believe that this gives the SARBAC model greater coherency and that the interaction between hierarchy operations and SARBAC relations is far simpler than in the ARBAC97 model.

Structurally every SARBAC relation is simpler than its ARBAC97 counterpart. (Note also that SARBAC defines three relations compared to five in ARBAC97.) It is clear that the admin-authority relation is far simpler than can-modify. In addition, the only constraints on tuples $(a, r) \in$ admin-authority are that $a \not< r$ and $(r, a) \notin$ admin−authority. However, every authority range in can-modify must be an encapsulated range and any pair of authority ranges must not overlap. A tuple in the ua-constraints relation consists of a role and an antichain. It can be seen from the examples in Figure 11(c) that this antichain will often consist of a single role.

8.3 Practicality and Versatility

It is apparent from Table VII that RHA$_4$ is more "permissive" than RRA97, in the sense that it is less likely to cause hierarchy operations to fail. (This

observation is confirmed by Proposition 8.2.) In particular, of the thirteen hierarchy operations in Table VII, only two succeed in ARBAC97, while only two fail in SARBAC. Given that none of these operations are unreasonable, it raises concerns about the utility of ARBAC97.

We can immediately see that the requirement that an authority range be an encapsulated range imposes considerable limitations on the hierarchy operations that can be performed. In particular, many of the hierarchy operations in Table VII fail in ARBAC97 because the encapsulation of one or more ranges would be violated by the operation.

The other factor that severely limits the success of hierarchy operations is the requirement in RRA97 that a new role has precisely one child and one parent in the resulting hierarchy. This requirement is imposed because the creation of a role with either no parent or no child will violate the encapsulation of some authority range in the hierarchy (unless the only authority range is the whole hierarchy). In short, encapsulated ranges, although conceptually appealing, limit RRA97 to the point where it is unlikely to be of any practical use.

The required existence of encapsulated ranges also limits the number of hierarchies to which RRA97 can usefully be applied. For example, Figure 12(a) shows a hierarchy in which the only encapsulated range is (E, ED). Figure 12(b) shows the same hierarchy with a bottom element `MinRole` appended. This gives rise to a hierarchy with the same characteristics as a role graph [Nyanchama and Osborn 1999]. However, it only introduces a single encapsulated range (`MinRole`, `DIR`), which does little to contribute to decentralized and autonomous administration of the hierarchy. (The hierarchy depicted in Figure 12(a) can easily be administered by RHA_4. In particular, the `admin-authority` relation defined in Figure 5(a) is perfectly suitable. In other words, RHA_4 is applicable to many more classes of role hierarchy than RRA97.) In short, encapsulated ranges place strict requirements both on the nature of initial role hierarchies and on their subsequent development.

Finally, we note that SARBAC can be used to build real hierarchies in a decentralized manner. Consider Figure 13. We first assume that an administrative role `DSO` has been created. `DSO` then creates the "backbone" of the department containing the roles `ED` and `DIR`. `DSO` now creates two administrative roles `PSO1` and `PSO2` each of which builds a (sub)hierarchy. Finally, `DSO` connects the backbone to the two hierarchies. Of course, this construction is not unique. We are merely illustrating how SARBAC might actually work in practice.

In contrast, it is not obvious how ARBAC97 is intended to work. Note that the administrative relations in ARBAC97 are static and are defined in terms of the role hierarchy. In other words, the role hierarchy is assumed to have an existence (at least conceptually) before ARBAC97 relations can be defined. The question is: at what point does ARBAC97 become applicable to a role hierarchy? Let us assume that `can−modify` = {(DSO, (ED, DIR))}, the backbone of the role hierarchy exists and that the administrative role hierarchy exists (far stronger assumptions than are required to build a hierarchy using SARBAC). Then `DSO` can now build the role hierarchy. However, additional tuples need to be added to `can-modify` (which is assumed to be a static relation in ARBAC97).
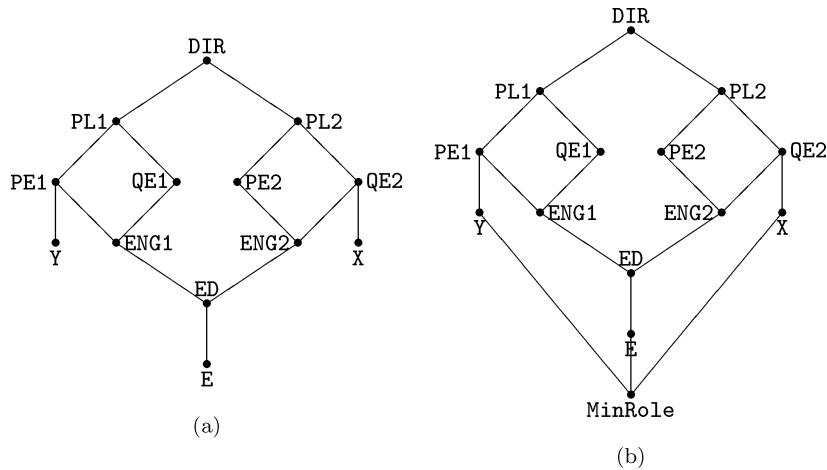
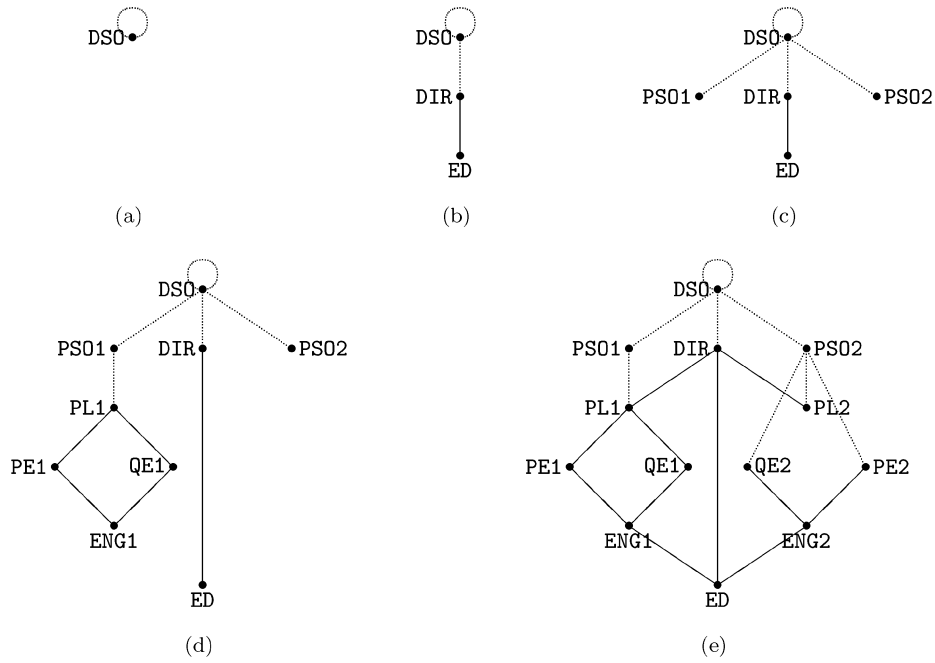Fig. 12.   A problematic hierarchy for RRA97.

8.4 Implementation Considerations

An implementation of role-based administration must include operations that update the relations of the underlying model as well as the RBAC96 relations *RH*, *UA*, and *PA*. Hence, the number of operations required will be at least twice the number of relations involved. (We can envisage operations in an implementation of role-based access control to be analogous to the six primitive operations in the protection matrix model due to Harrison et al. [1976]. These operations can be grouped into three pairs consisting of a delete operation and a create operation for the three dynamic components in the model.) Hence, SARBAC will require a minimum of 14 primitive operations (since there are four RBAC96 relations and three SARBAC relations), while ARBAC97 will require at least 18 primitive operations.

A detailed analysis of complexity is beyond the scope of this paper. This will be the subject of further research. However, we believe that hierarchy operations will be easier to implement and have lesser algorithmic complexity in SARBAC than in ARBAC97. We note in passing that to establish whether or not an administrative operation in ARBAC97 fails, it is necessary to examine every tuple in the appropriate ARBAC97 relation. In SARBAC, however, it is sufficient to establish whether the parameters of the operation satisfy a suitable condition pertaining to administrative scope.

8.5 Expressive Power

A rigorous analysis of expressive power is beyond the scope of this paper. Furthermore, we do not agree with some of the assumptions that informed the development of ARBAC97. In other words, we do not necessarily think it is worthwhile simulating ARBAC97 using SARBAC or vice versa. However, we will sketch the differences between ARBAC97 and SARBAC in order to convince the reader that little has been sacrificed to gain the simplicity and versatility of SARBAC. There are two main differences to consider.

Fig. 13.   Constructing a role hierarchy using SARBAC operations.

First, it is not possible in general to define administration over a range in SARBAC because the administrative scope of a given role will not necessarily be a range. However, it is not obvious to us that the correct unit of administration is a range. In particular, we are unaware of a practical justification for this approach, beyond the requirement that undesirable side effects be avoided. Obviously, a range may be appropriate in certain cases: for example, the range [ENG1, PL1] is a natural unit of administration for an administrative role with limited powers, but this range is also identified by SARBAC as the correct unit of administration. Moreover, we have seen compelling evidence to suggest that the use of ranges in general, and encapsulated ranges in particular, severely limits the applicability of ARBAC97. Indeed, the fact that no role can administer the set {Y, ENG1, PE1, QE1, PL1} in Figure 12(a), a set of roles obviously associated only with project 1, suggests that ARBAC97 is rather lacking in expressive power.

Second, it is not possible in SARBAC to define arbitrary constraints on user-role and permission-role assignments. However, we do not believe that this is a

significant shortcoming in our model. We have shown that it is not necessary to include disjunctions in constraints. Nevertheless, it is impossible in SARBAC to prohibit the assignment of a user u to a role if u is currently assigned some other specified role r because we do not use constraints of the form $\neg r$. Clearly, such constraints could be used to implement static separation of duty. Moreover, given that URA97 constraints apply to all users, it would seem that the sole purpose of a tuple such as (DSO, ED $\wedge \neg$PL1, [PL2, PL2]) in can-assign is to enforce static separation of duty between PL1 and PL2. It seems to us, therefore, that constraints of the form $\neg r$ can be specified using RCL 2000 [Ahn and Sandhu 2000], a role-based constraint specification language which is specifically designed to express separation of duty constraints in RBAC96. (Constraints of the form $\neg r$ in can-assignp can be used to implement operational separation of duty constraints, which can also be specified using RCL 2000.) In short, although constraints of the form $\neg r$ provide more options in controlling user-role and permission-role assignments, we would argue that RCL 2000 is a more appropriate way of specifying such behavior. (It should not be forgotten that constraints of the form $\neg r$ give rise to the possibility that the can-assign and can-assignp relations are not consistent, and also increase the complexity of administration.)

ARBAC97 supports weak and strong revocation for users and permissions. We noted in Remark 7.1 that a distinction can only be made between weak and strong revocation if it is assumed that the set of roles assigned to a user is not an antichain. However, this assumption introduces redundancy into the model because implicit user-role assignments can be recovered from explicit user-role assignments in the *UA* relation and the structure of the role hierarchy. Nevertheless, if we do not assume that $R(u)$ is an antichain, we can support both weak and strong revocation.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we have defined the concept of administrative scope and used it to construct a series of models for administering a role hierarchy. This culminated in the RHA$_4$ model which can be extended to a complete model for role-based administration. SARBAC, the resulting model, has a number of advantages over ARBAC97, the most comprehensive role-based administrative model in the literature. In particular, SARBAC is a complete model in that it provides for updates of all RBAC96 and SARBAC relations. We also believe that it is easier to understand the semantics of SARBAC and hence should be easier to maintain, and that the complexity of implementing SARBAC should be no worse than ARBAC97.

In the preceding section we discussed the relative merits of ARBAC97 and SARBAC, and found that our model is more attractive than RRA97 according to several different criteria. In short, we believe that our model offers significant practical and theoretical advantages over RRA97.

A couple of points worth noting about the extended hierarchy is that it can be used even when the set of roles is unordered (i.e., there is no role hierarchy), as in OASIS [Yao et al. 2001], for example. It can also be used to administer a set

of groups which naturally form a hierarchy under subset inclusion. That is, we can envisage a set of administrative subjects and an `admin-authority` relation where the most senior administrative subject assigns users to the largest groups and devolves the responsibility of assigning users to more specialized groups to less senior administrative subjects.

SARBAC also offers some unexpected benefits. First, it is possible to make the role hierarchy more fragmented because there is no reliance on ranges, one of the characteristics of the ARBAC97 model. Second, we have shown that it is possible to support discretionary access control in a rather natural way compared to existing approaches [Barkley 1997; Osborn et al. 2000].

Our immediate priority is to complete the SARBAC model by incorporating administration of separation of duty constraints into SARBAC. This should be relatively straightforward since a separation of duty constraint can be modeled as an antichain, and the administration of antichains arises when considering the `ua-constraints` and `pa-constraints` relations. We also intend to give operational semantics for RBAC96/SARBAC by writing pseudo-code functions to implement the SARBAC operations. In particular, we must define the atomic actions and the body of the functions. The work in this paper will be used to specify the conditional statements in the functions. Having achieved this, we will be in a position to consider complexity issues in more detail and examine the safety problem in the context of role-based access control. In the longer term, we hope to implement RBAC96/SARBAC in a prototype system and to consider the administration of separation of duty constraints using SARBAC. In addition, we intend to undertake a more thorough investigation of how SARBAC can be used to support discretionary access control in general, and delegation in particular.

REFERENCES

AHN, G.-J. AND SANDHU, R. 2000. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Sec. 3*, 4, 207–226.

BARKLEY, J. 1997. Comparing simple role-based access control models and access control lists. In *Proceedings of Second ACM Workshop on Role-Based Access Control* (Fairfax, VA), 127–132.

BELL, D. AND LAPADULA, L. 1973. Secure computer systems: mathematical foundations. Tech. Rep. MTR-2547, vol. I, Mitre Corporation, Bedford, MA.

CRAMPTON, J. 2002. Authorizations and antichains. Ph.D. thesis, Birkbeck College, Univ. of London, UK. Available at `http://www.isg.rhul.ac.uk/~umai001/Pubs/thesis.pdf`.

DAVEY, B. AND PRIESTLEY, H. 1990. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK.

FRIBERG, C. AND HELD, A. 1997. Support for discretionary role-based access control in ACL-oriented operating systems. In *Proceedings of Second ACM Workshop on Role-Based Access Control* (Fairfax, VA), 83–93.

GAVRILA, S. AND BARKLEY, J. 1998. Formal specification for role based access control user/role and role/role relationship management. In *Proceedings of Third ACM Workshop on Role-Based Access Control* (Fairfax, VA), 81–90.

GLIGOR, V. 1995. Characteristics of role-based access control. In *Proceedings of First ACM Workshop on Role-Based Access Control* (Gaithersburg, MD), II9–II14.

GOH, C. AND BALDWIN, A. 1998. Towards a more complete model of role. In *Proceedings of Third ACM Workshop on Role-Based Access Control* (Fairfax, VA), 55–61.

HARRISON, M., RUZZO, W., AND ULLMAN, J. 1976. Protection in operating systems. *Commun. ACM 19*, 8, 461–471.

HUA, L. AND OSBORN, S. 1998. Modeling UNIX access control with a role graph. In *Proceedings of International Conference on Computers and Information* (Winnepeg, MB, Canada).

JOSHI, J., GHAFOOR, A., AREF, W., AND SPAFFORD, E. 2001. Digital government security infrastructure design challenges. *IEEE Comput. 34*, 2, 66–72.

KOCH, M., MANCINI, L. V., AND PARISI-PRESICCE, F. 2002. A graph-based formalism for RBAC. *ACM Trans. Inf. Syst. Sec. 5*, 3, 332–365.

MOFFETT, J. AND LUPU, E. 1999. The uses of role hierarchies in access control. In *Proceedings of Fourth ACM Workshop on Role-Based Access Control* (Fairfax, VA), 153–160.

MUNAWER, Q. AND SANDHU, R. 1999. Simulation of the augmented typed access matrix model (ATAM) using roles. In *Proceedings INFOSECU99 International Conference on Information Security*.

NYANCHAMA, M. AND OSBORN, S. 1999. The role graph model and conflict of interest. *ACM Trans. Inf. Syst. Sec. 2*, 1, 3–33.

OSBORN, S., SANDHU, R., AND MUNAWER, Q. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Sec. 3*, 2, 85–106.

SADIGHI FIROZABADI, B. AND SERGOT, M. 1999. Power and permissions in security systems. In *Proceedings of 7th International Workshop on Security Protocols*, B. Christianson, B. Crispo, J. Malcolm, and M. Roe, Eds. Cambridge, UK, 48–59.

SANDHU, R., BHAMIDIPATI, V., AND MUNAWER, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. and Syst. Sec. 1*, 2, 105–135.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-based access control models. *IEEE Comput. 29*, 2, 38–47.

SANDHU, R., FERRAIOLO, D., AND KUHN, D. 2000. The NIST model for role-based access control: towards a unified standard. In *Proceedings of Fifth ACM Workshop on Role-Based Access Control* (Phoenix, AZ), 47–63. Available at `http://www.acm.org/sigsac/nist.pdf`.

SANDHU, R. AND MUNAWER, Q. 1998. How to do discretionary access control using roles. In *Proceedings of Third ACM Workshop on Role-Based Access Control* (Fairfax, VA), 47–54.

YAO, W., BACON, J., AND MOODY, K. 2001. A role-based access control model for supporting active security in OASIS. In *Proceedings of Sixth ACM Symposium on Access Control Models and Technologies* (Chantilly, VA), 171–181.