Uni-ARBAC: A Unified Administrative Model for Role-Based Access Control

Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan

Institute for Cyber Security
University of Texas at San Antonio
prosun.csedu@gmail.com, {ravi.sandhu, ram.krishnan}@utsa.edu

Abstract. Many of the advantages of Role Based Access Control (RBAC) accrue from the flexibility of its administrative models. Over the past two decades, several administrative models have been proposed to manage user-role, permission-role and in some cases role-role relations. These models are based on different administrative principles and bring inherent advantages and disadvantages. In this paper, we present a unified model, named Uni-ARBAC, for administering user-role and permission-role relations by combining many of the administrative principles and novel concepts from prior models. For example, instead of administering individual permissions Uni-ARBAC combines permissions into tasks which are assigned to roles as a unit. Slightly differently, users are assigned to user-pools from where individual users are assigned to roles. The central concept of Uni-ARBAC is to integrate user-role and task-role administration into a more manageable unit called an Administrative Unit (AU). AUs partition roles, tasks and user-pools and they are organized in a rooted tree hierarchy. Administrative users are assigned to AUs with possibility of restricting their authority to user-role assignment or task-role assignment. While most existing models assume existence of administrative roles for managing regular roles, we present an approach for engineering AUs based on structured partitioning of roles and tasks.

1 Introduction

Role Based Access Control (RBAC) [6, 17] is one of the most widely deployed and studied access control models. Instead of directly assigning permissions to users, RBAC assigns permissions to roles and users to roles. While the number of roles in a large organization might vary from dozens to thousands, the number of users or permissions could vary from tens of thousands to hundreds of thousands and even millions. Thus, maintaining the user-role and permission-role relations are the most commonly carried out administrative actions in RBAC. While some models also speak to administering the role-role hierarchy [5, 15], it is evident that modifications to the role-role relationship can have significant impact, so it might be advisable to keep this authority relatively

centralized. Hence, we limit our scope in this paper to decentralized administration of the user-role and permission-role relations.

To a large degree the advantages of RBAC accrue from the flexibility of administering the permission-role and user-role relations. In this regard, several administrative models have been proposed in the literature (see Section 2). These models are based on different administrative principles and offer inherent advantages and disadvantages. Each one incorporates some novel and putatively useful concepts relative to the others. To our knowledge, there has been no effort so far to comprehensively consolidate the various novel concepts introduced in different administrative models into a coherent unified model that potentially brings together the inherent advantages of the individual models.

In this paper, we present a novel unified model, named Uni-ARBAC, for administering user-role and permission-role relations by combining many of the existing administrative principles and novel concepts. For example, instead of administering individual permissions, Uni-ARBAC combines permissions into tasks and assigns tasks to roles. For administrative purposes, Uni-ARBAC decouples users and tasks from roles following the decoupling principle of ARBAC02 [13]. Uni-ARBAC utilizes user-pools as sets of candidate users who can be assigned to a role, while tasks act as permission-pools. One advantage of using tasks as permission-pools is that tasks can be designed during role engineering according to some top-down approaches (e.g. [11]). User-pools on the other hand can be designed via the organization structure.

Uni-ARBAC integrates user-role and task-role administration into a more manageable unit, we call Administrative Unit (AU). AUs partition roles, tasks and user-pools, and are organized in a rooted tree hierarchy. Administrative users are assigned to AUs with possibility of restricting their authority to user-role assignment or task-role assignment. The partitioning of roles and tasks across AUs leads us to propose an engineering process for AUs for given role hierarchy and/or task hierarchy. The potential for engineering AUs in this manner is a significant advantage of Uni-ARBAC.

This paper makes the following contributions. We have presented a unified model (Uni-ARBAC) for administering user-role and permission-role relation for RBAC. Uni-ARBAC combines several novel concepts and administrative principles from prior models into a more powerful and manageable unit called administrative unit (AU). We proposed an engineering approach for developing AUs. While most other administrative models assume the existence of separate administrative roles,we relax

this assumption and our approach for engineering administrative units can also be used for engineering administrative roles.

The remainder of this paper is organized as follows. We discuss related work in Section 2 highlighting the concepts we have adopted from prior administrative models. We present our model in Section 3 and some variations of the model in Section 4. Section 5 discusses our approach for engineering AUs. We conclude the paper in Section 6.

2 Background and Related Work

In this section, we review prior models for administering RBAC, emphasizing those of their driving principles which have been incorporated in Uni-ARBAC. These concepts and principles are summarized in Table 1.

The value of grouping permissions into a higher level abstraction has often been recognized in the literature. Task-role based access control (TRBAC) [12] proposes the notion of a task as a group of permissions which constitute a fundamental unit of business work in an enterprise. Similar to TRBAC, two-sorted RBAC [9] and scenario-based role engineering [11] organize tasks into another higher level abstract.

One of the central notions of RBAC administration is to separate user-role and permission-role assignments. Introduced in ARBAC97 [15], this notion is adopted by many other models including [13, 14, 16]. Uni-ARBAC accepts this separation to be at the core of the model.

Another essential concept of ARBAC97 is to keep administration of roles separate from regular roles. To this end, ARBAC97 introduced the concept of administrative roles. Uni-ARBAC adopts the former separation principle, but eschews the use of administrative roles for this purpose. Instead, Uni-ARBAC introduces a more sophisticated construct of Administrative Units to achieve the desired separation.

ARBAC02 [13] is another influential model for administrative RBAC. It documents a number of problems with ARBAC97 and introduces the notions of user-pools and permission-pools. Uni-ARBAC adopts the user-pool and user-pool hierarchy from ARBAC02, while on the permission side it adopts the task and task hierarchy from [12] as discussed above.

Crampton et al. developed a model called SARBAC [4, 5] based on the concept of administrative scope, which confines the side effects of role hierarchy modification in a highly disciplined manner relative to ARBAC97. Notably, administrative scope becomes a means to define administrative roles which are otherwise assumed to be given in ARBAC97 and ARBAC02. Administrative scope is mathematically defined based

Table 1: Concepts motivating Uni-ARBAC (* denotes source of the concept)

Concepts & Principles	ARBAC97 [15]	ARBAC02 [13]	SARBAC [4], [5]	UARBAC [10]	Role graph model [18]	Uni-ARBAC
Task & task hierarchy						✓
Separation of user & permission administration	✓*	✓	✓		✓	✓
Separation of regular roles from administration	✓*	✓			✓	✓
User pools & user pool hierarchy		✓*				✓
Administrative structure design			✓*		✓	✓
Reversibility & administrative structure flexibility				✓*		✓
Senior most administrators					✓*	✓

on the given role hierarchy. Uni-ARBAC incorporates the general notion that the role hierarchy should influence the administrative structure. However, it departs from the strict mathematical definition of SARBAC to accommodate a heuristic top-down approach in designing administrative units, based on the role-hierarchy and task-role allocation.

The UARBAC [10] model proposes a number of principles for RBAC administration, such as scalability and flexibility, psychological acceptability and economy of mechanism. As noted earlier Uni-ARBAC departs from UARBAC on the question of whether or not administrative permissions should be assigned to regular roles. However, all the other principles of UARBAC are considered similarly desirable in Uni-ARBAC. As it stands some of the UARBAC principles, such as psychological acceptability and scalability, are qualitative and difficult to convincingly claim for a given model. Here we confine our attention to the two principles of reversibility and administrative structure flexibility, which have been explicitly adopted from UARBAC into Uni-ARBAC. The reversibility principle requires that administrative operations should be reversible. This is incorporated in Uni-ARBAC by coupling grant and revoke operations for user-role or task-role assignment in a single administrative unit. The principle of administrative structure flexibility (called policy neutrality in [10]) argues against the tight coupling of administrative structure to role hierarchy, such as in SARBAC.

It remains to consider the Role-Graph Administration Model [18]. It partitions roles into units called administrative domains. The model explicitly includes a single highest administrative domain which includes the MaxRole and MinRole from the underlying Role-Graph model. Uni-ARBAC adopts this concept embodying it in the highest administrative unit at the root of the administrative unit tree hierarchy.

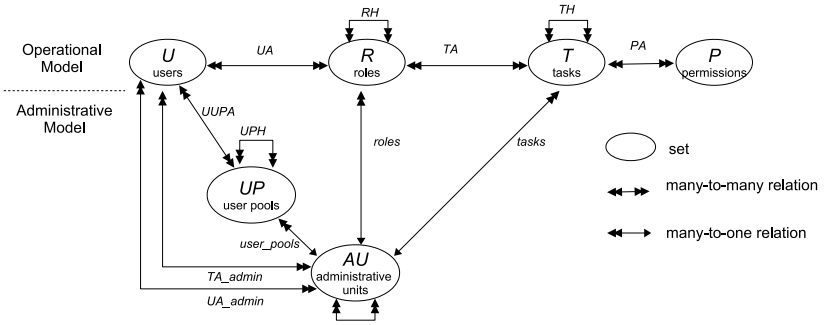

Fig. 1: The **Uni-ARBAC** Model for User-Role and Task-Role Administration

In addition to the administrative models discussed above, there are other notable models developed in various applied contexts, especially in temporal/location aware RBAC [1, 2], Enterprise RBAC [7, 8]), event driven RBAC [3], administration of cryptographic RBAC [19] etc.

3 The Uni-ARBAC Model

In this section, we describe the Uni-ARBAC model, along with formal definitions. The overall structure of Uni-ARBAC is illustrated in Figure 1. We consider Uni-ARBAC in two parts: the operational model for RBAC with respect to regular roles and permissions, and the administrative model for administering the user-role and task-role relations of the former. These are respectively discussed in the following two subsections.

3.1 Uni-ARBAC Operational Model

The sets and relations in the top part of Figure 1 represent the Uni-ARBAC operational model, which is slightly different from the standard RBAC model [6]. The most salient difference is that there is a level of indirection in role-permission assignment, so permissions are assigned to tasks and tasks are assigned as a unit to roles. As discussed in Section 2, this additional indirection has emerged in several different administrative models in the literature. Additionally tasks are organized in a partial order $\succeq_t$, whereby a senior task inherits all permissions from its juniors. For example, in Figure 2(b), task t1 is senior to tasks t2 and t3, so it inherits permissions from both of them, and so on. User-role assignment remains unchanged from standard RBAC, so individual users are assigned to and deassigned from roles. For simplicity, we have not considered the standard RBAC concepts of sessions and role activation.

Table 2: Uni-ARBAC Operational Model

I. Traditional RBAC Sets & Relations	*III. Derived Function*
- U, R and P (users, roles and permission)	- $authorized\_perms(r : R) \rightarrow 2^P$, defined as
- $RH \subseteq R \times R$, partial order on roles $\succeq$	$authorized\_perms(r) = \{p \mid (\exists t, t' \in T)$
- $UA \subseteq U \times R$, user-role assignment relation	$(\exists r' \in R)[r \succeq r' \wedge t \succeq_t t' \wedge$
	$(t, r') \in TA \wedge (p, t') \in PA]\}$
II. Added RBAC Sets & Relations	
- T, set of tasks	*IV. Authorization Function*
- $TH \subseteq T \times T$, partial order on tasks $\succeq_t$	- $can\_exercise\_permission(u : U, p : P) =$
- $PA \subseteq P \times T$, permission-task assignment rel.	$(\exists r \in R)[p \in authorized\_perms(r)$
- $TA \subseteq T \times R$, task-role assignment relation	$\wedge (u, r) \in UA]$

The Uni-ARBAC operational model is formalized in Table 2. Item I specifies the familiar components carried over from traditional RBAC. Item II specifies the additional components which effect the additional indirection between permissions and roles via tasks. Item III formalizes the interaction between the role hierarchy, task hierarchy, and permission-task and task-role assignments. The interaction is schematically depicted in Figure 3 and formally expressed in the *authorized_perms* function. The authorization function in item IV specifies the authorization required for a user to exercise a permission, which is that the permission must be authorized to at least one role assigned to the user. A familiar role hierarchy from the literature and an example task hierarchy are shown in Figures 2(a) and 2(b), respectively.

3.2 Uni-ARBAC Administrative Model

We now turn to the Uni-ARBAC administrative model illustrated in the lower part of Figure 1, and formalized in Table 3. The administrative model introduces a number of additional components. First we have the notion of user-pools and user-pool hierarchy adopted from ARBAC02 [13]. An example user-pool hierarchy is shown in Figure 2(c). This example has three independent user-pools DP, DevP and EP, with DevP being senior to a number of other user-pools, i.e., CTP, CPLP, MTP and MPLP. Motivation for the user-pool hierarchy in this instance is by virtue of qualifications, so every user in the CPLP pool is also eligible to be a developer in the DevP pool. The hierarchy obviates the need to do multiple assignments of a user to both pools in such cases. Users are assigned to user-pools via the $UUPA$ user to user-pool assignment relation. The user-pool notion is formally specified in item I of Table 3.

The central mechanism in Uni-ARBAC is the administrative unit. The set of administrative units is denoted as AU, while individual administrative units are indicated as au, au_i, au_j, etc. Uni-ARBAC requires

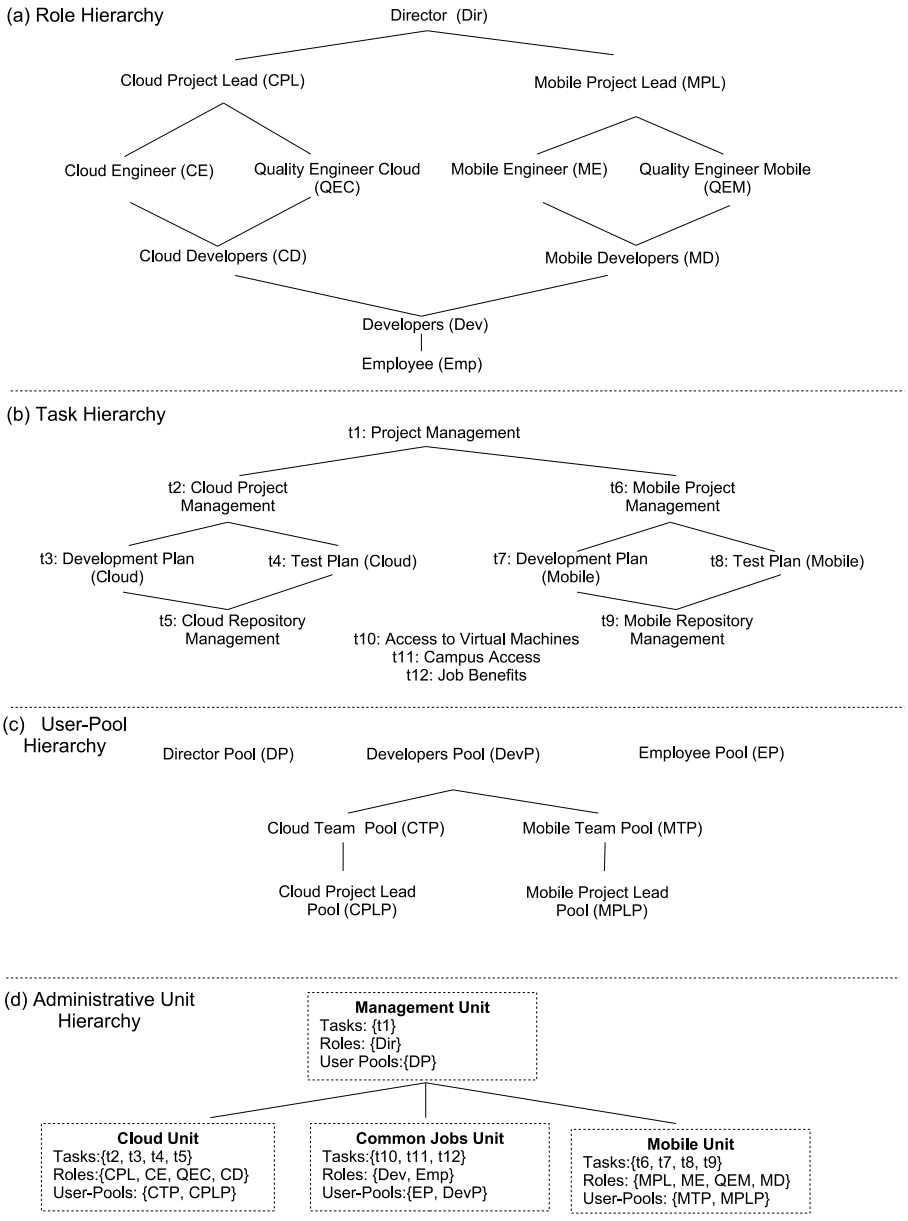

Fig. 2: Examples of Uni-ARBAC hierarchies

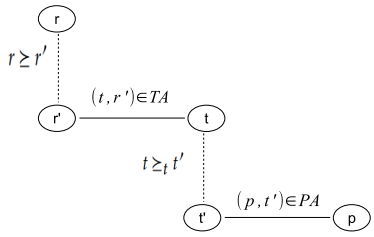

Fig. 3: Interaction of role and task hierarchies in operational model

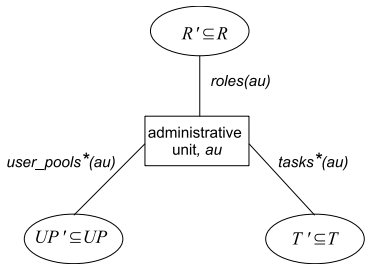

Fig. 4: Scope of control of an AU

that each *au* manages an exclusive set of roles which is not under the purview of another *au*. The *roles* function in item II of Table 3 is a partitioning assignment in that it must satisfy the requirements that $roles(au_i) \cap roles(au_j) = \emptyset$ for $au_i \neq au_j$, and $\bigcup_{au \in AU} roles(au) = R$. The effect of partitioning is that each role is allocated to exactly one *au* for administration. Each *au* only manages the roles it is directly assigned. The effect of the role hierarchy is limited to the operational model.

The partitioning concept is further applied in Uni-ARBAC to tasks and user-pools via the *tasks* and *user_pools* functions in item II of Table 3. These functions must satisfy the requirements $tasks(au_i) \cap tasks(au_j) = \emptyset$ for $au_i \neq au_j$, $\bigcup_{au \in AU} tasks(au) = T$, $user\_pools(au_i) \cap user\_pools(au_j) = \emptyset$ for $au_i \neq au_j$, and $\bigcup_{au \in AU} user\_pools(au) = UP$.

In this manner an administrative unit manages a explicitly assigned partition of roles, to which it can assign users from an assigned partition of user-pools and tasks from an assigned partition of tasks. Unlike for roles, Uni-ARBAC extends the authority of an *au* to junior tasks and user-pools, for which purpose we define the $tasks^*$ and $user\_pools^*$ functions in item III of Table 3. The net effect is illustrated in Figure 4, and further discussed in context of item V of Table 3. An example partitioning of roles, tasks and user-pools across four administrative units is shown in Figure 2(d). We also note that for a given *au* it is permissible to assign an empty partition of roles, tasks or user-pools. While, such situations may be unusual, the model does not prohibit them.

Next we consider assignment of users to administrative units (item IV of Table 3). Users can be assigned via the TA_admin or the UA_admin relation. The former authorizes the task-role assigment power of an *au*, while the latter authorizes the user-role assignment power. In this way, these two capabilities can be separately assigned to users, even though they are coupled in the *au*. This embodies the separation of user and permission assignment principle of Section 2. A user assigned to any *au* via TA_admin or UA_admin is said to be an administrative user.

Table 3: Uni-ARBAC administrative model

I. User-Pools Sets & Relations - UP, set of user-pools - $UPH \subseteq UP \times UP$, partial order $\succeq_{up}$ - $UUPA \subseteq U \times UP$, user to user-pool assignment relation *II. AU and Partitioned Assignments* - AU, set of administrative units - $roles(au : AU) \rightarrow 2^R$, assignment of roles - $tasks(au : AU) \rightarrow 2^T$, assignment of tasks - $user\_pools(au : AU) \rightarrow 2^{UP}$, assignment of user-pools *III. Derived Functions* - $tasks^*(au : AU) \rightarrow 2^T$, defined as $tasks^*(au) = \{t' \mid (\exists t \in tasks(au)) t \succeq_t t'\}$ - $user\_pools^*(au : AU) \rightarrow 2^{UP}$, and $user\_pools^*(au) = \{up' \mid (\exists up \in user\_pools(au)) up \succeq_{up} up'\}$ *IV. Administrative User Assignments* - $TA\_admin \subseteq U \times AU$ - $UA\_admin \subseteq U \times AU$ - $AUH \subseteq AU \times AU$, rooted tree partial order $\succeq_{au}$	*V. Authorization Functions* - $can\_manage\_task\_role(u : U, t : T, r : R) =$ $(\exists au_i, au_j)[(u, au_i) \in TA\_admin \wedge$ $au_i \succeq_{au} au_j \wedge r \in roles(au_j) \wedge t \in tasks^*(au_j)]$ - $can\_manage\_user\_role(u_1 : U, u_2 : U, r : R) =$ $(\exists au_i, au_j)[(u_1, au_i) \in UA\_admin \wedge$ $au_i \succeq_{au} au_j \wedge r \in roles(au_j) \wedge (\exists up \in UP)$ $[(u_2, up) \in UUPA \wedge up \in user\_pools^*(au_j)]$ *VI. Administrative Actions* - $assign\_task\_to\_role(u : U, t : T, r : R)$ Authorization: $can\_manage\_task\_role(u, t, r)$ = True Effect: $TA' = TA \cup \{(t, r)\}$ - $revoke\_task\_from\_role(u : U, t : T, r : R)$ Authorization: $can\_manage\_task\_role(u, t, r)$ = True Effect: $TA' = TA \backslash \{(t, r)\}$ - $assign\_user\_to\_role(u_1 : U, u_2 : U, r : R)$ Authorization: $can\_manage\_user\_role(u_1, u_2, r)$ = True Effect: $UA' = UA \cup \{(u, r)\}$ - $revoke\_user\_from\_role(u_1 : U, u_2 : U, r : R)$ Authorization: $can\_manage\_user\_role(u_1, u_2, r)$ = True Effect: $UA' = UA \backslash \{(u, r)\}$

For convenience in maintaining the TA_admin and UA_admin relations, Uni-ARBAC also defines a hierarchy $\succeq_{au}$ on administrative units. Assignment of user u to a senior au_i for task-role administration, i.e., $(u, au_i) \in TA\_admin$, effectively also assigns u for task-role administration to all au_j for $au_i \succeq_{au} au_j$. Likewise for $(u, au_i) \in UA\_admin$. For simplicity, Uni-ARBAC requires $\succeq_{au}$ to be a rooted tree hierarchy. One effect of this is that there is a seniormost au. An example administrative units tree is shown in Figure 2(d).

The authorization functions of Uni-ARBAC are specified in item V of Table 3 as boolean functions that return true or false. The function $can\_manage\_task\_role(u : U, t : T, r : R)$ specifies the conditions for user u to assign/revoke task t to/from role r. The requirement is schematically depicted in Figure 5. User u must be assigned as a TA_Admin to the unique administrative unit au_j which has jurisdiction over role r, or alternately so assigned to an administrative unit $au_i \succeq_{au} au_j$. In either case task t must be assigned to au_j or be junior to a task assigned to au_j.

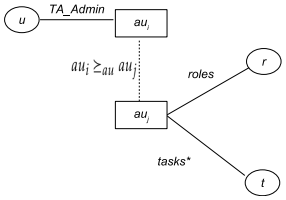

Fig. 5: Task-role authorization

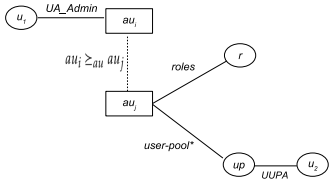

Fig. 6: User-role authorization

The $can\_manage\_user\_role(u_1 : U, u_2 : U, r : R)$ similarly specifies the conditions for user u_1 to assign/revoke user u_2 to/from role r, and is schematically depicted in Figure 6. User u_1 must be assigned as UA_Admin to the unique administrative unit au_j which has jurisdiction over role r, or alternately to an administrative unit $au_i \succeq_{au} au_j$. In either case user u_2 must be assigned via $UUPA$ to a user-pool which is directly assigned to au_j or to some user-pool junior to a user-pool directly assigned to au_j.

Item VI of Table 3 formalizes the four administrative actions of Uni-ARBAC. Assigning and revoking have the same authorization and the effect is self-explanatory. The alignment of authorization for assign and revoke embodies the principle of reversibility of administrative actions discussed in Section 2.

3.3 Uni-ARBAC Invariants

Invariants are properties that hold for the lifetime. For the moment assume we start with an initial state in which both TA and UA are empty, i.e., the roles have no tasks or users assigned. Let us denote TA where all possible TA assignments have been made as TA_{max}. It is evident that,

$$TA_{max} = \bigcup_{au \in AU} \{(t, r) | t \in tasks^*(au) \wedge r \in roles(au)\}.$$

Because of reversibility of assign and revoke, and the independence of each assignment from another, in any state TA must satisfy

$$\emptyset \subseteq TA \subseteq TA_{max}. \tag{1}$$

It is further evident that any value of TA bounded as in equation 1 is realizable. We can either build up from an empty TA or build down from TA_{max}. In fact we can take the system from any value of TA compliant with equation 1 to any other compliant value. Further, we can relax our assumption of an empty TA in the initial state. Any initial state with TA compliant with equation 1 will ensure that TA is maintained within

Table 4: Uni-ARBAC with Aggressive Inheritance

V'. Modified Authorization Functions
- $can\_manage\_task\_role(u:U, t:T, r:R) = (\exists au_i, au_j, au_k)[(u, au_i) \in TA\_admin \wedge$ $au_i \geq_{au} au_j \wedge au_i \geq_{au} au_k \wedge r \in roles(au_j) \wedge t \in tasks^*(au_k)]$
- $can\_manage\_user\_role(u_1:U, u_2:U, r:R) = (\exists au_i, au_j, au_k)[(u_1, au_i) \in UA\_admin$ $\wedge au_i \geq_{au} au_j \wedge au_i \geq_{au} au_k \wedge r \in roles(au_j) \wedge (\exists up \in UP)[(u_2, up) \in UUPA \wedge$ $up \in user\_pools^*(au_k)]$

these bounds. Finally, let TA_0 denote TA in the initial state. Any $(t,r) \in TA_0/TA_{max}$ cannot be revoked and will persist in all subsequent states. These observations can be proved formally but are quite evident. As an example, the upper bound of TA for the administrative unit hierarchy of Figure 2(d) does not contain the pair of $(t1, CPL)$. Thus the task $t1$ cannot be assigned to the role CPL using the instance of the AUs in Figure 2(d).

We can make similar observations with respect to the maximal possible values of UA as follows, $UA_{max} =$

$$\bigcup_{au \in AU} \{(u,r) | (\exists up \in UP)[(u,up) \in UUPA \wedge r \in roles(au) \wedge up \in user\_pools^*(au)]\}$$

so UA is bounded as follows.

$$\emptyset \subseteq UA \subseteq UA_{max} \tag{2}$$

4 Variations of Uni-ARBAC

In this section we discuss some variations of Uni-ARBAC which materially alter the characteristics of the model. Uni-ARBAC has a rich structure so it is not surprising that many variations are possible. Some are relatively incremental, such as allowing the administrative hierarchy to be a general partial order rather than a rooted tree. Here we discuss a few variations that raise some substantial policy issues.

4.1 Aggressive Inheritance Model

In this variation we allow a senior *au* to do more than simply the sum of what the *au* itself is authorized to do plus what each of the junior *au*'s are allowed. To be concrete consider the AU hierarchy of Figure 2(d), and an administrative user u assigned as TA_admin and UA_admin to the Management Unit administrative unit. This user also inherits membership in the Cloud Unit and Mobile Unit administrative units. User u is thereby authorized, for example, to assign task t2 to role CPL and users from

user-pool CTP to role CPL. However, u cannot make assignments across the junior administrative units, such as task t2 to role MPL and users from user-pool MTP to role CPL. We denote this form of inheritance in the AU hierarchy as membership inheritance.

With the alternate aggressive inheritance we allow cross assignments across junior administrative units by a senior administrator. There is clearly a major policy difference between the two forms of inheritance. The senior *au* effectively serves as a single consolidated *au* with freedom to assign any task to any role, and users from any user-pool to any role. With aggressive inheritance the user u discussed above will be able to assign t2 to role MPL and users from user-pool MTP to role CPL. Perhaps more dangerously, user u will be able to assign task t1 to MPL. The effect of aggressive inheritance is formally stated in the modified authorization functions of Table 4. Everything else from Table 3 applies unchanged to Uni-ARBAC with aggressive inheritance. The senior most *au* at the root of the AU hierarchy can assign any task to any role and any user (assuming every user is in at least one user-pool) to any role, so $TA_{max} = T \times R$ and $UA_{max} = U \times R$. Equation 1 and equation 2 continue to hold. Other less aggressive variations can also be considered.

4.2 No Self-Administration Model

Consider the administrative actions $assign\_user\_to\_role(u_1 : U, u_2 : U, r : R)$ and $revoke\_user\_from\_role(u_1 : U, u_2 : U, r : R)$ of Table 3 item VI. In general, u_1 can equal u_2, so it is permissible for u_1 to assign and revoke himself to and from roles. In some contexts, this may be considered as a conflict of interest. To avoid this, an additional check that $u_1 \neq u_2$ can be added to the $can\_manage\_user\_role$ authorization function of Table 3.

5 Engineering Administrative Units

There can be different meaningful AU hierarchies for a given set of roles. For example, for the roles in Figure 2(a), two different instances of AUs are given in Figure 2(d) and Figure 7, based on different partitioning of the roles. Crampton and Loizou partition roles in defining 'administrative scopes' in [5] to confine the side effects of role hierarchy modification in a highly disciplined manner. The AU structure in Figure 2(d) is based on the partitioning of roles defined in administrative scope.

As we have argued, one particular partition is not suitable for all requirements. UARBAC [10] argues that role partitioning according to

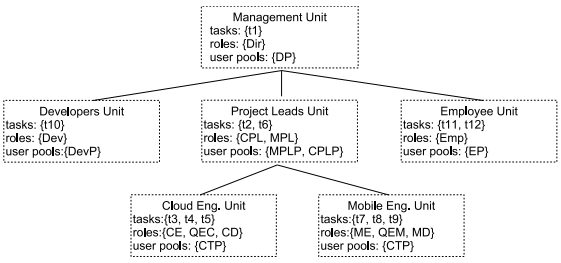

Fig. 7: Alternate AUs for roles, tasks and user-pools as given in Figure 2

administrative scope does not work well for all different types of role hierarchies and there should be flexibility in the administration structure.

We develop the following simple and flexible partitioning heuristics, which are applicable to many different types of hierarchies and can be configured to produce different partitions for a given set of roles or tasks.

1. Most senior roles in the role hierarchy contain critical tasks. They should be administered separately. Similarly, most junior roles contain tasks that most other roles inherit. They should also be administered separately.
2. For rest of the roles, iteratively select the most senior and most junior roles until all roles are partitioned.

After partitioning and specifying role set for each AU, we can populate tasks in the AUs using given task-role allocation. Scenario based top-down approach for engineering roles [11] derives tasks in an intermediate process and assign them to roles. Thus, we believe, the process of engineering roles can be utilized to derive task-role allocation. On the other hand, the process of engineering user-pools and allocating them in AUs, assigning administrative users into AUs should also be carried out to develop working AUs. We do not further elaborate these issues here.

6 Conclusion

In this paper, we present Uni-ARBAC, a unified model for administering user-role and task-role relations in Role Based Access Control. It combines various novel concepts and administrative principles from prior works. It integrates user-role and permission-role administration into a manageable unit, we call Administrative Unit. While most of the previous models assume existence of administrative roles for managing regular roles, we relax this assumption and our approach for engineering administrative units can be used for engineering administrative roles.

Nonetheless, Uni-ARBAC has limitations. It uses several sets and relations of which it administers only task-role and user-role relations. Further research in this area is needed to realize a complete model.

7 Acknowledgement

This research is partially supported by NSF Grants CNS-1111925 and CNS-1423481.

References

1. E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *TISSEC*, 4(3):191–233, 2001.
2. E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. GEO-RBAC: a spatially aware RBAC. In *Proc. of 10th SACMAT*, pages 29–37. ACM, 2005.
3. P. Bonatti, C. Galdi, and D. Torres. ERBAC: event-driven RBAC. In *Proc. of 18th SACMAT*, pages 125–136. ACM, 2013.
4. J. Crampton. Understanding and developing role-based administrative models. In *Proc. of 12th ACM CCS*, pages 158–167, 2005.
5. J. Crampton and G. Loizou. Administrative scope: a foundation for role-based administrative models. *ACM TISSEC*, 6(2):201–231, 2003.
6. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC*, 4(3):224–274, 2001.
7. A. Kern. Advanced features for enterprise-wide role-based access control. In *Proc. of 18th ACSAC*, pages 333–342. IEEE, 2002.
8. A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. In *Proc. of 8th ACM SACMAT*, pages 3–11, 2003.
9. W. Kuijper and V. Ermolaev. Sorting out role based access control. In *Proc. of 19th ACM SACMAT*, pages 63–74, 2014.
10. N. Li and Z. Mao. Administration in role-based access control. In *Proc. of 2nd ACM ASIACCS*, pages 127–138, 2007.
11. G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional RBAC roles. In *Proc. of 7th ACM SACMAT*, pages 33–42, 2002.
12. S. Oh and S. Park. Task–role-based access control model. *Information systems*, 2003.
13. S. Oh and R. Sandhu. A model for role administration using organization structure. In *Proc. of 7th ACM SACMAT*, pages 155–162, 2002.
14. R. Sandhu. The ASCAA principles for next-generation role-based access control. In *Proc.of 3rd ARES*, 2008.
15. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM TISSEC*, 2(1):105–135, 1999.
16. R. Sandhu and Q. Munawer. The ARBAC99 model for administration of roles. In *Proc. of 15th Annual ACSAC*, pages 229–238. IEEE, 1999.
17. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
18. H. Wang and S. L. Osborn. An administrative model for role graphs. In *Data and Applications Security XVII*, pages 302–315. Springer, 2004.
19. L. Zhou, V. Varadharajan, and M. Hitchens. Secure administration of cryptographic role-based access control for large-scale cloud storage systems. *JCSS*, 80(8), 2014.