Administration Model for Or-BAC

Frédéric Cuppens[1] and Alexandre Miège[2]

[1] GET/ENST Bretagne/Département RSM,
BP 78, 2 rue de la Châtaigneraie, 35512 Cesson Sévigné Cedex, France
frederic.cuppens@enst-bretagne.fr
[2] ENST,
46, rue Barrault, 75634 Paris Cedex 13, France
alexandre.miege@cert.fr

Abstract. Even though the final objective of an access control model is to provide a framework to decide if actions performed by subjects on objects are permitted or not, it is not convenient to directly specify an access control policy using concepts of subjects, objects and actions. This is why the Role Based Access Control (RBAC) model suggests using a more abstract concept than subject to specify a policy. The Organization Based Access Control (Or-BAC) model further generalizes the RBAC model by introducing the concepts of activity and view as abstractions of action and object. In the Or-BAC model, it is also possible to specify privileges that only apply in some given contexts.

In this paper, we present AdOr-BAC, an administration model for Or-BAC. This model is fully homogeneous with the remainder of Or-BAC. AdOr-BAC can control assignment of user to role (User Role Administration), assignment of permission to role (Permission Role Administration) and assignment of user to permission (User Permission Administration). This last possibility is useful to control fine grained delegation, when a user wants to grant a specific permission to another given user. AdOr-BAC is compared with other administration models, such as the ARBAC model suggested for RBAC, showing some of its advantages.

1 Introduction

The final objective of an access control policy is to specify the *permissions*, *obligations* and *prohibitions* that control the *actions* performed by *subjects* on *objects*. However, when defining its access control policy, an *organization* does not directly specify that a given subject (for instance John) is permitted to perform a given action (for instance *read*) on a given object (for instance Jack's medical record). Organization based access control policies never mention that they apply to specific subjects, actions or objects. Instead, they use more abstract concepts such as the concept of role [14, 7]. In this case, the access control policy does not directly grant permissions to subjects but to roles. A given subject will then obtain permissions by playing roles, in which case this user will inherit all the permissions associated with these roles. In [3], the Organization Based Access Control (Or-BAC) model is defined and it is argued that similar abstractions should be associated with actions and objects. For this purpose, the Or-BAC model introduces the abstract concepts of activities and views. In Or-BAC, the access control policy defines permissions (or obligations or prohibitions) that control the *activities* performed by *roles* on *views*. For instance, the policy might specify that role *physician* has permission to perform activity *consult* on view *medical record*[3]. We can then derive that user John is permitted to perform action read on object Jack_med_record if John is playing the role physician, read is an action that corresponds to the activity consult and object Jack_med_record belongs to the view medical record.

A complete access control model must provide an administration model. For instance, the Role Based Access Control (RBAC) model is associated with the ARBAC97 model [11], further refined in the ARBAC99 [13] and ARBAC02 [9] models. The ARBAC model includes two main components:

- The URA (User Role Administration) model to control who is permitted to assign a user with a new role and who is permitted to revoke a user from an existing role.
- The PRA (Permission Role Administration) model to control who is permitted to assign a role with a new permission and who is permitted to revoke a role from an existing permission.

The RBAC model also includes the RRA component for Role-Role Administration to manage the role hierarchy.

The objective of this paper is to present an administration model for the Or-BAC model called AdOr-BAC. The AdOr-BAC model includes three main components: URA, PRA and UPA (for User Permission Administration). The objective of the URA and PRA components is similar to components with similar names already defined in the RBAC model, but the model we suggest in AdOr-BAC is different. This model is fully homogeneous with the remainder of Or-BAC. In particular, the syntax used to specify permissions in the URA and PRA models is similar to the one suggested in the Or-BAC model. We shall see that this approach has several advantages over the ARBAC model. The UPA component does not exist in the ARBAC model. It is useful when a given user wants to grant a permission to another given user. For instance, John (a physician) may want to grant to Jane (his secretary) a permission to have an access to Jack's medical record. The UPA model applies in this case. It is used to specify that subjects playing the role of physician are permitted to grant to other subjects playing the role of medical secretary a permission to have an access to objects belonging to the view medical record. Notice that, using UPA, we can also specify that subjects playing the role physician are forbidden to grant a permission to subjects playing another role than medical secretary, for instance journalist or insurer. Notice also that the PRA model does not apply in this case since the general

[3] Actually, the Or-BAC model allows the administrators to specify more complex permissions since one can consider that each permission only applies in some given *contexts* (see section 3 for further details).

permissions of role medical secretary do not change, only Jane's permissions are updated.
The remainder of this paper is organized as following. In section 2, we present ARBAC, the administration model suggested for RBAC, and discuss some of its weaknesses. Section 3 briefly recalls the Or-BAC model. Section 4 presents the AdOr-BAC model that is used to administer the Or-BAC model. Finally section 5 concludes the paper.

2 ARBAC

The Role-Based Access Control (RBAC) model [6, 8, 14] aims to use the *role* as a central concept. Ravi Sandhu proposed an administration model, ARBAC, dedicated to the management of a RBAC policy.

ARBAC97: ARBAC97 [11] is the first RBAC administration model. ARBAC has two main features. First, it provides the possibility of administrating an RBAC policy in a decentralized, but without loosing the control over rights' propagation. Second, though the administrative roles and permissions are based on RBAC, they are completely separated from the regular roles and permissions. ARBAC97 provides three sub-models:

- URA97 [10]. This model describes how to assign users to the predefined roles. The assignment by an administrative role of a user to a regular role is based on a ternary relation "can_assign" between the administrative role, the prerequisites roles and the regular role. That is, a member of an administration role can assign a user to a regular role if this user satisfies the condition corresponding to the prerequisite roles.
- PRA97 [11]. This model is the dual of URA97 and it describes the assignment of permissions to roles. It is also based on a ternary relation with prerequisite conditions.
- RRA97 [12]. This last model proposed rules for the role-role assignment, that is the construction of the role hierarchy.

Therefore, ARBAC97 offers a proper administration model which is not exactly the case for the other security models. In order to obtain a decentralized administration of a RBAC policy, it could be used this way: The management of the role hierarchy and the assignment of the permissions is carried out by a centralized authority on the one hand. On the other hand, the assignment and the revocation can be left under the responsibility of the chiefs of the different departments or units, through the assignment of these chiefs to administrative roles.
However, it is noteworthy to point out the following shortcomings. ARBAC claims that it is an auto-administrated model. This is not completely true because it does not use the RBAC model to define administrative permissions. Instead, it creates new assignment and revocation rules (such as can_assign and can_revoke) used by the administrative roles. These rules are distinct from the approach suggested in RBAC to define permissions associated with regular roles.

As we mentioned, the assignment relation is ternary. Thus the prerequisite conditions depend on the administrative role and the regular role. However, it seems that the prerequisite conditions generally depend logically and only on the regular role and should rather be considered as a constraint on the regular role.

Moreover, ARBAC does not give any information on the creation of the roles, and does not offer any delegation mechanism. Since ARBAC97, a proposal has been made to manage delegation in RBAC model through RBDM [1]. Delegation will be further discussed in section 4.4.

ARBAC does not offer means to express contextual conditions. Thus, it is not possible to express that a given administrative role is permitted to assign a permission to a regular role only at working hours or only from his own terminal. This kind of restrinction can be usefull to detect administrator's abuse of power for instance.

ARBAC extensions: Two ARBAC extensions have been proposed. ARBAC99 [13] presents a way to manage the mobile and immobile users and permissions. Unlike a mobile user, an immobile user can be seen as a non-permanent user such as a user under training, a visitor, a consultant, etc. In this case, the user can be a member of a role and get the corresponding permissions. But an administrative role cannot use this membership to put the immobile user into other roles. That is, an immobile user cannot climb the hierarchy. The same idea is used for the immobile permissions.

The objective of ARBAC02 [9] is different. Several weaknesses of ARBAC97 have been pointed out. Through ARBAC02 some improvements were proposed to resolve, among others, the multi-step user assignment which generate a lot of work for the security officers and which causes redundant tuples in the URA management. The main modification made in ARBAC02 affects the prerequisite conditions for the user and the permission assignment. An organization structure of user pool and an organization structure of a permission pool are created. The first one is managed by the human resources group, the second one by the IT group. We obtain two hierarchies independent from the role hierarchy. User and permission assignment is made by the security officers by picking user and permissions in these pools. This simplifies the assignment processes.

These two extensions of ARBAC97 are interesting but do not answer the shortcomings we have just mentioned. Moreover ARBAC02 simplifies the assignment process, but transfer the problem of the prerequisite conditions onto the human resources group and the IT group.

SARBAC (Scoped Administration of Role-Based Access Control) [2] suggests an extension of RRA97, called RHA$_4$, and an alternative to ARBAC97. SARBAC relies on administrative scope which changes dynamically as the role hierarchy changes. Thus, update operations over RBAC96 and SARBAC relations become easier and cannot lead to inconsistent rules. In particular, SARBAC makes it possible to delete a role without any restriction. Unlike in ARBAC97, it is possible to assign administrative roles to users as SARBAC does not make any distinction between regular and administrative roles.

3 Or-BAC

Before presenting the administration model for the Or-BAC model, we shall briefly recall the main components of this model (see [3] for further details). The most important entity in Or-BAC is the entity *Organization*. Roughly speaking, an organization can be seen as an organized group of subjects, playing some role or other. Notice that a group of subjects does not necessarily correspond to an organization. More precisely, the fact that each subject plays a role in the organization corresponds to some agreement between the subjects to form an organization.

In the organization, subjects will request to perform actions on objects and, as mentioned in the introduction, the final objective of an access control policy is to decide if these requests are permitted or not. In the Or-BAC model, a subject will be either an active entity, i.e. a user, or an organization. Actions will mainly correspond to concrete computer actions such as "read", "write", "send", etc.

However, permission in the Or-BAC model does not directly apply to subject, action and object. Instead, subject, action and object are respectively abstracted into role, activity and view. A view corresponds to a set of objects that satisfy a common property. Similarly, an activity will join actions that partake of the same principles.

A given access control policy is then specified by a set of *facts* having the form:[4] $Permission(org, role, activity, view, context)$. These facts specify that, in organization org, a given $role$ is permitted to perform a given $activity$ on a given $view$ in a given $context$. Examples of context may be *Night*, *Working-Hours* or *Urgency* (see section 3 for further details about the context definition).

Specifying the access control policy by facts is an important difference compared with other approaches based on logical *rules*, such as Ponder [5]. This will represent a major advantage when we shall define how to administer Or-BAC (see section 4). Notice that the specification of the security policy is parameterized by the organization so that it is possible to handle simultaneously several security policies associated with different organizations.

Basic concepts of Or-BAC: In Or-BAC, there are height basic sets of entities: Org (a set of organizations), S (a set of subjects), A (a set of actions), O (a set of objects), $\mathcal{R}$ (a set of roles), $\mathcal{A}$ (a set of activities), $\mathcal{V}$ (a set of views) and $\mathcal{C}$ (a set of contexts).

We shall assume that $Org \subseteq S$ (that is any organization is a subject) and that $S \subseteq O$ (that is any subject is an object). Any entities in the Or-BAC model may have some attributes. This is represented by functions that associate the entities with the value of these attributes. For instance, if s is a subject, then $name(s)$ represents the name of s, $address(s)$ its address, etc.

[4] Actually, in [3], it is also possible to specify prohibitions and obligations using Or-BAC. Here, for the sake of simplicity, we shall only consider permissions. This is mainly to eliminate the problem of conflicts between permission and prohibition. However, we plan to analyze this problem of conflict in a forthcoming paper.

Modelling the organization components: In the organization, subjects are empowered in roles, objects are used into views and actions fall within activities. This is represented by the following relationships:

- *Empower* is a relation over domains $Org \times S \times \mathcal{R}$. If org is an organization, s a subject and r a role, then $Empower(org, s, r)$ means that org empowers subject s in role r. Unlike the TMAC model or the RBAC model which consider binary relations between organizations and subjects or between subjects and roles, notice that our model consider a ternary relation between organizations, subjects and roles. This is useful to model situations where a given subject plays several roles but in different organizations. Let us also remark that subjects might be users as well as organizations.
- Use is a relation over domains $Org \times O \times \mathcal{V}$. If org is an organization, o is an object and v is a view, then $Use(org, o, v)$ means that org uses object o in view v. This ternary relation makes ourselves able to characterize organizations that give different definitions to the same view. For instance, take the case of the view "medical record" defined in Purpan hospital as a set of Word documents and defined in Rangueil hospital as a set of tuples in a relational database.
- *Consider* is a relation over domains $Org \times A \times \mathcal{A}$. If org is an organization, α is an action and a is an activity, then $Consider(org, \alpha, a)$ means that org considers that action α falls within the activity a. Since *Consider* is a ternary relation, different organizations may decide that one and the same action comes under distinct activities or that different actions come under the same activity. For instance, activity "consulting" corresponds, in Purpan hospital, to an action "read" that can be ran on data files whereas it corresponds, in Rangueil hospital, to action "select" that can be performed on relational databases.

Context definition: Contexts are used to specify the concrete circumstances where organizations grant roles permissions to perform activities on views. In the health care domain, the entity *Context* will cover circumstances such as "urgency", "industrial medicine", "attending physician", etc. Every context can be seen as a ternary relation between subjects, objects and actions defined within a given organization. Therefore, entities *Organization*, *Subject*, *Object*, *Action* and *Context* are linked together by the relationship $Define$:

- $Define$ is a relation over domains $Org \times S \times A \times O \times \mathcal{C}$ If org is an organization, s is a subject, α is an action, o is an object and c a context, then $Define(org, s, \alpha, o, c)$ means that within organization org, context c holds between subject s, action α and object o.

The conditions required for a given context to be linked, within a given organization, to subjects, objects and actions will be formally specified by logical rules. For instance, we may define the context *Night* as follows:[5]

[5] In the remainder of this paper, we shall use a logical notation to represent relationship: if R is a n-ary relationship over domains $D_1 \times ... \times D_n$, then the predicate $R(d_1, ..., d_n)$ is *true* if and only if $\langle d_1, ..., d_n \rangle \in R$.

- $\forall s, \forall \alpha, \forall o, \forall c_1, \forall c_2, (Define(H1, s, \alpha, o, Night)$
  $\leftrightarrow (20:00 \leq time(global\_clock) \vee time(global\_clock) \leq 8:00))$
  that is, in H1, the context "night" is *true* between subject s, action α and object o between 20:00 and 8:00.

In the following, we shall use another context called "default". This context is *true* in every circumstance. It is defined as follows:

- $\forall org, \forall s, \forall \alpha, \forall o, Define(org, s, \alpha, o, Default)$
  that is, in every organization *org*, the context "default" is always *true* between subject s, action α and object o.

Policy definition: In the Or-BAC model, the access control policy is defined using the relationship *Permission* as follows:

- *Permission* is a relation over domains $Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C}$, If *org* is an organization, r is a role, a is an activity, v is a view and c a context then $Permission(org, r, a, v, c)$ means that organization *org* grants role r permission to perform activity a on view v within context c.

Deriving concrete permission: The relationship *Permission* enables a given organization to specify permissions between roles, activities and views in a given context. However, an access control model must provide a framework for describing the concrete actions that may be performed by subjects on objects. For the purpose of modelling concrete permissions, we introduce the relationship $Is\_permitted$ as a relationship between subjects, actions and objects:

- *Is_permitted* is a relation over domains $S \times A \times O$
  If s is a subject, α is an action and o is an object then $Is\_permitted(s, \alpha, o)$ means that subject s is permitted to perform action α on object o.

In our model, triples that are instances of the relationship $Is\_permitted$ are logically derived from permissions granted to roles, views and activities by the relationship *Permission*. This is modelled by the following general rule:

- $\forall org, \forall s, \forall o, \forall \alpha, \forall r, \forall v, \forall a, \forall c,$
  $Permission(org, r, a, v, c) \wedge$
  $Empower(org, s, r) \wedge Use(org, o, v) \wedge Consider(org, \alpha, a) \wedge$
  $Define(org, s, \alpha, o, c) \rightarrow Is\_permitted(s, \alpha, o)$
  that is, if organization *org*, within the context c, grants role r permission to perform activity a on view v, if *org* empowers subject s in role r, if *org* uses object o in view v, if *org* considers that action α falls within the activity a and if, within *org*, the context c is *true* between s, α and o then s is permitted to perform α on o.

Notice that we do not assume that all instances of relationship *Is_permitted* comes from the specification of relationship *Permission*. This means that there may exist other instances of relationship *Is_permitted*. These instances may be viewed as exceptions to the general security policy specified by the relationship *Permission*. This will be used in UPA (see section 4.4) when a user wants to grant a specific permission to another given user.

Notice that in [3], it is also suggested to define hierarchies over roles (as in the RBAC model) but also organization, activity and view, and to associate permission inheritance with these different hierarchies. However, since this possibility will not be used in the remainder of this paper, we prefer to omit it.

4 AdOr-BAC: an administration model for Or-BAC

4.1 Introduction

The objective of this section is to define an administration model for Or-BAC, called AdOr-BAC. A complete administration model should provide means to control the following activities: management[6] of organizations, management of roles, activities, views and contexts, assignment (and revocation) of users to roles, assignment (and revocation) of permissions to roles, assignment (and revocation) of users to permissions.

Due to space limitation, we focus in this paper on the user-role assignment, the permission-role assignment and the user-permission assignment. The approach we suggest in AdOr-BAC is to define these administration functions by considering three different views respectively called URA, PRA and UPA. Each organization will manage such views. Objects belonging to these views have specific semantics; namely they will be respectively interpreted as an assignment of user to a role, a permission to role and a permission to a user.

Intuitively, inserting an object in these views will enable an authorized user to respectively assign a user to a role, assign a permission to a role or assign a permission to a user. Conversely, deleting an object from these views will enable a user to perform a revocation.

Defining the administration functions in AdOr-BAC then corresponds to define which roles is permitted to have an access to views URA, PRA and UPA, or to more specific views when the role has not a complete access to one of these views. For instance, the role physician may be only permitted to assign users to the role medical secretary. In this case, the role physician will have not a complete access to the view URA, but only to the sub-part corresponding to the role medical secretary.

The approach we suggest is homogeneous with the remainder of the Or-BAC model. The syntax we use in AdOr-BAC to define permission to administer the policy is completely similar to the remainder of Or-BAC. Actually, strictly speaking, it is even incorrect to consider that AdOr-BAC is a distinct model from Or-BAC. Since, we have simply to consider three new views, namely URA, PRA and UPA in the Or-BAC model, it would be more appropriate to say that Or-BAC is an auto-administered model. In the following we shall present the

[6] By manage, we mean create, delete and update.

structure of these three views and further analyze the administration functions associated with management of these views.
Notice that, in the ARBAC model, there are two types of fully separated roles called regular roles and administration roles. Administration roles are only permitted to perform administration functions and regular roles are only permitted to perform other functions excluding administration functions. In some circumstances this separation is superfluous. For instance the role physician may hold a plurality of administrative and non administrative permissions. In such case, it is not necessary to create two roles, this is, a role *physician* and a role *admnistration_physician*. The AdOr-BAC model does not impose to create these two roles. But, as a security policy designer could legitimatly want to separate them anyway, because of separation of duty and least privilege questions, the AdOr-BAC model makes it possible to do so. Thus, we leave such separation optional in the AdOr-BAC model. Keeping this separation makes The AdOr-BAC model compliant with ARBAC.

4.2 URA in AdOr-BAC

The view URA: The aim of the user-role administration is to determine who is allowed to assign a user to a role and on which conditions. Assigning a user to a role equals adding a new object in a given view called URA. Three attributes are associated with this view: *subject* to designate the subject which is related to the assignment, *role* that corresponds to the role to which the subject will be assigned and *org* to represent the organization in which the subject is assigned
We consider the function associated to these three attributes. For example, if a security officer is allowed to assign a user to the role *physician* in the department of cardiology *cardio_dpt* of its hospital H, we can create the $URA\_physician\_cardio\_dpt$ view. This view is defined as follows:

- $\forall ura$
  $$Use(H, ura, URA\_physician\_cardio\_dpt) \rightarrow$$
  $$Use(H, ura, URA) \wedge role(ura) = physician \wedge org(ura) = cardio\_dpt$$

In the Or-BAC model, an organization empowers a user in a role. It is characterized by the relationship *Empower*. Therefore, there is a link between the object belonging to the view URA and the relationship *Empower*. This link is modelled through the following rule:

- $\forall org, \forall ura, Use(org, ura, URA)$
  $$\rightarrow Empower(org(ura), subject(ura), role(ura))$$

It means that a user empowered in a given organization corresponding to org can manage user-role assignment of another organization (corresponding to $org(ura)$). For instance, org might be the human resources department of a given company and $org(ura)$ might be the different departments of this company.

The activity *manage*: The view URA makes it possible to model the assignment of a user to a role. We have to consider now the activity that corresponds to the permission of assigning someone. We call *assign* this activity. The permission granted to the role $sec\_officer$ to assign a user to the role *physician* in the department of cardiology *cardio_dpt* of its hospital H is expressed as follows:

- $Permission(H, sec\_officer, assign, URA\_physician\_cardio\_dpt, Default)$

Up to now, we have only dealt with assignment but not with revocation. Notwithstanding it is easy to create the activity *revoke* in the way as the activity *assign*.

- $Permission(H, sec\_officer, revoke, URA\_physician\_cardio\_dpt, Default)$

When a role is authorized to both assign and revoke users to a specific role, we create the activity *manage*, and consider the activities *assign* and *revoke* as two sub-activities of *manage*:

- $\forall org, \forall role, \forall view, \forall context,$
  $Permission(org, role, manage, view, context) \rightarrow$
  $Permission(org, role, assign, view, context) \wedge$
  $Permission(org, role, revoke, view, context)$

that is if a given role is permitted to manage a given view in a given context, this role is also permitted to perform assignment and revocation of this view in the same context.

The prerequisite conditions: In the ARBAC model, the relation *can_assign* makes it possible to add prerequisite conditions on the role of the user concerned by the assignment. It is possible to express this kind of condition in the AdOr-BAC model. Let us consider the following example:

- The director is permitted to designate a user as the head of department of cardiology but only if this user is a member of the role *physician*:
  $P3 : Permission(H, director, assign, URA\_head\_cardio\_dpt, Default)$
  The view $URA\_head\_cardio\_dpt$ is defined as follows:
  $\forall ura, Use(H, ura, URA\_head\_cardio\_dpt) \leftrightarrow$
  $Use(H, ura, URA\_head\_dpt) \wedge$
  $Empower(H, subject(ura), physician)$

We can thus specify that for the department of cardiology the head must be a physician. It is no use having any perquisite condition for the revocation of this head, that is why this last permission is granted just for the activity *assign*. This is specified in a similar way as permission $P1$ but the permission only applies to activity *assign*.

The user-role assignment in AdOr-BAC is very flexible. A large number of conditions can be expressed such as the prerequisite conditions of ARBAC, thanks to the use of views which make it possible to model the assignments.

4.3 PRA in AdOr-BAC

In the previous section we dealt with the user-role administration. We discuss here the permission-role administration. As we have just seen, we modelled user assignment with the view URA. Here, the permission assignment is modelled with a new view called PRA. Giving a new permission to a role corresponds to create a new object which complies with the view PRA.

The view *PRA*: Five attributes are associated with the view PRA:

- *issuer*: the organization where the permission applies
- *grantee*, *privilege*, *target*: the role, activity and view concerned by the permission
- *context*: designate the context in which the rule can be applied

There is a link between the objects belonging to the view PRA and the relationship $Permission$. This link is modelled as follows :

- $\forall org, \forall context, Use(org, pra, PRA) \rightarrow$
  $Permission(issuer(pra), grantee(pra), privilege(pra), target(pra), context(pra))$

The activity *manage*: The same activities *assign*, *revoke* and *manage* defined in the previous section are used to express the authorization given to a role to assign and revoke permissions to other roles.

The prerequisite conditions: The prerequisite conditions defined in ARBAC related to the permission-role assignment can be expressed in our model through the view PRA as we saw in the URA section.

4.4 UPA in AdOr-BAC

The URA and PRA components respectively allow an authorized user to assign users to roles and permissions to roles. Thus, these components indirectly enable this authorized user to assign permissions to users. We argue that sometimes a more direct process should enable a user to grant a permission to another user. For instance, let us consider a situation where there are two users, John a physician and Jane his medical secretary. The role medical secretary is not permitted to have an access to the view medical record. John makes a consultation on Jack, a patient and, after this consultation, wants to update Jack's medical record. However, John is too busy to do so; he decides to grant Jane a permission to update Jack's medical record. Notice that permissions of the role medical secretary do not change, Jane simply gets a new permission from John. This is the objective of the UPA component to control the assignment of a new permission to a user and revocation of an existing permission. For this purpose, we consider the same activities *assign*, *revoke* and *manage* as the ones suggested in URA and PRA. Actually, we can consider two different cases called UPA1 and UPA2. UPA1 enables an authorized user to grant another user a permission to perform a specific action on a specific object. UPA2 is more general. It enables an authorized user to grant another user a permission to perform a given activity on a given view. Due to space limitation, we only present UPA1; UPA2 can be similarly defined. We shall then analyze how UPA1 applies to model the concept of delegation.

UPA1: granting permissions on specific objects and actions In this case, we consider a view $UPA1$ with five attributes having the same names as PRA but with sightly different meaning: *issuer* represents the organization who is issuing the permission, *grantee* is the subject who is receiving the permission, *privilege* represents the action the grantee is authorized to perform, *target* represents the object the grantee is authorized to have an access to and *context* is the context in which the permission applies. There is a rule that specifies that we can derive, from objects belonging to the view $UPA1$, the fact that a subject is permitted to perform an action on an object. This is modelled by the following rule:

- $\forall org, \forall upa,$
  $use(org, upa, UPA1) \wedge$
  $Define(issuer(upa), grantee(upa), privilege(upa), target(upa), context(upa))$
  $\rightarrow Is\_permitted(grantee(upa), privilege(upa), target(upa))$
  that is, if an object *upa* is used by a given organization *org* in view $UPA1$ and the issuer of *upa* defines that the context holds between the grantee, the privilege and the target specified by upa, then the grantee is permitted to use his privilege on the target.

The permissions derived from this rule may be viewed as exceptions to the general permissions defined by the predicate *Permission*. This is exactly the purpose of the UPA component to provide means to specify such exceptions.

Let us now show how this material is used to specify that, in a given hospital H1, a physician is permitted to grant his or her medical secretary a permission to update the medical record of one of his or her patient. We have first to consider a sub-view $SPUMR$ (for Secretary Permission of Update Medical Record) of view UPA1 defined as follows:

- $\forall upa, Use(H1, upa, SPUMR) \leftrightarrow$
  $Use(H1, upa, UPA1) \wedge$
  $Empower(H1, grantee(upa), medical\_secretary) \wedge$
  $Consider(H1, privilege(upa), update) \wedge$
  $Use(H1, target(upa), medical\_record)$
  that is object *upa* is used in view $SPUMR$ if and only if it is used in view $UPA1$ and the values of attributes *grantee*, *privilege* and *target* respectively correspond to a user empowered as a medical secretary, an action considered as an updating activity and an object used as a medical record.

The permission is then specified as follows:

- $Permission(H1, physician, assign, SPUMR, PSP)$
  that is, in H1, the role physician has permission to assign a permission belonging to view SPUMR in context PSP (for Physician's Secretary and Patient). The context PSP is defined as follows:
  $\forall s, \forall a, \forall upa, Define(H1, s, a, upa, PSP) \leftrightarrow$
  $\quad Empower(H1, s, physician) \wedge Use(H1, upa, UPA1) \wedge$
  $\quad grantee(upa) \in secretary(s) \wedge name(target(upa)) \in patient(s)$
  that is, H1 defines that subject s performs action a on object upa in context PSP if s is a physician in H1, upa is used in view $UPA1$, the grantee of upa is a secretary of s and the target name of upa is a patient of s.

Using this permission, John (a physician of H1), is permitted to grant Jane (his secretary) a permission to update Jack's medical record (his patient).

Application to delegation Modelling delegation is a complex problem. The analysis performed in [1] shows that there are several subtleties leading to many possible definitions of the concept of delegation. The objective of this paper is not to fully investigate this problem. We shall simply show that the expressiveness of AdOr-BAC is sufficient to model several of these subtleties.

In AdOr-BAC, permission to delegate may be represented by facts having the following forms:

- $Permission(org, role, delegate, view, context)$
  meaning that, in organization org, $role$ is permitted to delegate a permission on $view$ in a given $context$. $view$ is a sub-view of $UPA1$ or $UPA2$ (depending on the delegation is to perform a specific action on an object, or an activity on a view).

It is generally assumed that to delegate a permission to a user, the grantor must first hold the permission he wants to delegate. In AdOr-BAC, this is modelled by a context *AG* (for Authorized Grantor) defined as follows:

- $\forall org, \forall s, \forall a, \forall upa,$
  $\quad Define(org, s, a, upa, AG) \leftrightarrow$
  $\quad\quad Use(org, upa, UPA1) \wedge$
  $\quad\quad Is\_permitted(s, privilege(upa), target(upa))$
  that is, in any organization org, subject s performs action a in context AG if org uses upa in view $UPA1$[7] and s is permitted to perform the delegated privileged action on the delegated target object.

In some circumstances, we may also specify that the delegation only applies temporarily and will be automatically revoked after a given deadline. In AdOr-BAC, this may be modelled by a temporal context. Temporal and other types of contexts are further investigated in [4]. Another possible restriction is that the grantor will loose the permission he has delegated. In AdOr-BAC, this means

[7] The definition of context AG must be slightly changed if we consider view $UPA2$.

that delegation is not an elementary activity but the combination of assigning a permission (as modelled in UPA1 or UPA2) and self-revoking this permission on the grantor (this may be also modelled in UPA1 or UPA2). We do not further develop this analysis of the delegation concept in this paper. We plan to continue this investigation in the future.

5 Conclusion

In this paper, we have presented AdOr-BAC, an administration model for the Or-BAC model. Using AdOr-BAC, the definition of an administration policy is defined in a similar way as the remainder of the security policy specified in Or-BAC. Thus, Or-BAC is a fully auto-administered model, we suggest a logical-based model to express both Or-BAC and AdOr-BAC. In a forthcoming paper, we plan to give an interpretation of this model using a syntax closed to SQL.

AdOr-BAC provides a good compromise between fully centralized (and too rigid) administration as in the MAC model, or fully decentralized (but uncontrolled) administration as in the DAC model. When creating a new Or-BAC policy, we suggest starting with a unique user (the creator of the policy), a unique organization (whose name is defined by the creator) and a unique predefined role *policy-designer* assigned to the creator. The role *policy-designer* has minimal permissions to create new organizations, define roles to administer these organizations and specify permissions associated with these roles. Thus, using AdOr-BAC, one can specify a decentralized administration, but it is always possible to control and limit the capabilities to administer associated with the different created roles.

We develop three main components for AdOr-BAC called URA for User-Role Administration, PRA for Permission-Role Administration and UPA for User-Permission Administration. The UPA component is useful to control User to User delegation, when a user wants to grant another user a specific permission. We suggest two variations of the UPA component: UPA1 that enables a user to delegate a permission to perform a specific action on a specific object and UPA2 to delegate a permission to perform an activity on a view. Applying the UPA component to model delegation still requires further work. As mentioned in [1], there are several different characteristics related to delegation such as permanence, monotonicity, totality, levels of delegation, cascading revocation. We have started modelling some of these criteria in the context of the AdOr-BAC model through context definitions. We plan to continue this work, in particular to model how to refine non elementary activities (such as non monotonic delegation) into elementary ones (such as permission assignment and self-revocation). We have not here taken to account the role hierarchy, and the inheritance cascading revocation issues which might appear then. Multi-step delegations also require further investigation.

Finally, we do not discuss the enforcement of an Or-BAC security policy administrated with AdOr-BAC. This problem will be studied in a forthcoming paper.

Acknowledgement: For this work, Alexandre Miège is funded by France Télécom R&D and Frédéric Cuppens is funded by the RNRT project MP6 and the ACI CASC of the ministery of Research.

References

1. Ezedin Barka and Ravi Sandhu. Framework for Role-Based Delegation Models. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, New Orleans, Louisiana, December 2000.
2. J. Crampton and G. Loizon. SARBAC: A New Model for Role-Based Administration. Technical Report BBKCS-02-09, Birkbeck College, University of London, July 2002.
3. F. Cuppens, P. Balbiani, S. Benferhat, Y. Deswarte, A. Abou El Kalam, R. El Baida, A. Mige, C. Saurel, and G. Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Come, Italy, June 2003.
4. F. Cuppens and A. Mige. Modelling Contexts in the Or-BAC Model. In *Proceedings of 19th Applied Computer Security Associates Conference (ACSAC 2003)*, Las Vegas, Nevada, December 2003.
5. N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proceedings of IEEE 2th International Workshop on Policies for Distributed Systems and Networks (POLICY 2001)*, Bristol, UK, January 2001.
6. David F. Ferraiolo and D. Richard Kuhn. Role-Based Access Controls. In Z. Ruthberg and W. Polk, editors, *Proceedings of the 15th NISt-NSA National Computer Security Conference*, pages 554–563, Baltimore, MD, October 1992.
7. S. I. Gavrila and J. F. Barkley. Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management. In *Third ACM Workshop on Role-Based Access Control*, pages 81–90, October 1996.
8. L. Guiri. A new model for role-based access control. In *Proceedings of the 11th Annual Computer Security Applications Conference*, pages 249–255, New Orleans, LA, December 1995.
9. S. Oh and R. Sandhu. A Model for Role Administration Using Organization Structure. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 155–162, Monterey, California, June 2002.
10. R. Sandhu and V. Bhamidipati. The URA97 Model for Role-Based User-Role Assignment. In *Proceedings of IFIP WG 11.3 Workshop on Database Security.* North-Holland, Lake Tahoe, California, 1997.
11. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 Model for Role-Based Administration of Roles. *ACM Transactions on Information and System Security*, 2(1), February 1999.
12. R. Sandhu and Q. Munawer. The RRA97 Model for Role-Based Administration of Role Hierarchies. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC'98).* Phoenix, Arizona, December 1998.
13. R. Sandhu and Q. Munawer. The ARBAC99 Model For Administration of Roles. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC'99)*, Phoenix, Arizona, December 1999.
14. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.