

AdOrBAC: An Administration Model for Or-BAC

Frédéric Cuppens
ENSTB

BP 78 - 2, rue de la Châtaigneraie
35512 Cesson Sévigné Cedex, France
Email: cuppens@enst-bretagne.fr

Alexandre Miège
ENST

46, rue Barrault
75634 Paris Cedex 13, France
Email: 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 the concept of role as 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 facility 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. Finally, we show how the administration policy specified in AdOr-BAC is also used to control accesses to OToKit, a module we have developed to specify security policies in Or-BAC and their administration in AdOr-BAC.

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* generally 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 [20, 11]. 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 [13], 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*¹. 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 [17], further refined in the ARBAC99 [19] and ARBAC02 [14] models. The ARBAC model includes two main components:

¹Actually, the Or-BAC model allows the administrators to specify more complex permissions since one can consider that each permission only applies in some specific *contexts* (see section 3 for further details).

- 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 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 an overview of administration principles that apply to other access control models such as DAC, MAC and RBAC and discuss some of their 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. In section 5, we show how to use AdOr-BAC to specify access control to OToKit (Or-BAC Tool Kit), a module that allows to specify security policies in Or-BAC and their administration in AdOr-BAC. Finally section 6 concludes the paper.

2 Existing administration models

2.1 Centralized and decentralized administration models

In a Mandatory Access Control (MAC) policy, a clearance is assigned to each subject and a classification is assigned to each object. Thus, the administration of a MAC policy consists of managing the object classification and the subject clearance. A unique authority first defines the criteria used for the classification and the clearance, and then determines the security level for each subject and object. It is a fully centralized administration model in which the subjects have no rights to administer. The MAC Administration is very rigid. What is more, the transposition to IT systems is complex especially regarding networks. Finally, this administration mode does not seem to be convenient for commercial needs [9]. Regarding the administration of a MAC policy, the main problem is the management of object classification. Since the classification rules are often difficult to apply, automatic support tools are suitable to assist the users when choosing the initial classification of an object. The object classification of an object is not static but evolves over time, leading to downgrading the initial classification when the object content becomes less sensitive. In this case, tools that manage the object classification and apply rules to automatically downgrade object classifications are useful. The general principles for such a tool were presented in [6]. See also [3] that presents the design and implementation of SACADDOS, a support tool to automatically manage object classification.

On the other hand, discretionary access control (DAC) models are fully decentralized. In most implementations, right management is based on the notion of object ownership: if a user owns an object, he is allowed to give another user rights to access it. Thus, the main authority fully delegates the management of the user's rights: Users are responsible for deciding who is permitted to have an access to the objects they own. The administration's work is reduced to its minimum. The main consequence is the risk of losing control over the rights' propagation [4].

The administration of the MAC and DAC models shows two opposite aspects of a security policy administration. One of them is fully centralized whereas the other is completely decentralized. The implementation of a security model and high security needs encourage us to look for a half-way solution. In this case, the RBAC administration model is an option to be investigated.

2.2 ARBAC

The Role-Based Access Control (RBAC) model [10, 12, 20] aims to use the *role* as a central concept. Ravi Sandhu proposed an administration model, ARBAC, dedicated to the management of a RBAC policy.

2.2.1 ARBAC97

ARBAC97 [16, 17] is the first RBAC administration model. ARBAC has three main features. First, it provides the possibility of administrating an RBAC policy in a decentralized, but without losing the control over rights' propagation. Second, through ARBAC, it presents an RBAC auto-administration model. Third, 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 [15]. 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 [17]. 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 [18]. 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 is not a real auto-administrated model because it does not use the RBAC permissions but it creates new assignment and revocation rules (such as `can_assign` and `can_revoke`) used by the administrative 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. For instance, this kind of restriction can be useful to detect administrator's misuse of power.

2.2.2 ARBAC extensions

Two ARBAC extensions have been proposed. ARBAC99 [19] 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 [14] 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) [5] is an extension of 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

3.1 Introduction

Before presenting the administration model for the Or-BAC model, we shall briefly recall the main components of this model (see [13] 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 group actions that share the same principles.

A given access control policy is then specified by a set of *facts*. Each fact has the following form:²

- $Permission(org, role, activity, view, context)$

and specifies 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 3.5 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 [8]. 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.

²Actually, in [13], 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.

3.2 Basic concepts of Or-BAC

In Or-BAC, there are eight 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 entity 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. Notice that instead of using functions, we could use binary predicates. For instance, instead of writing $name(s) = n$ where $name$ is a function, it is actually possible to write $name(s, n)$. In the following, we shall use functions because we guess formulas are easier to read. However, in OToKit, the prototype we have implemented (see section 5), every entity attribute is modelled by a predicate.

3.3 Entity-organization assignment

Every role, activity and view may be not relevant in all organizations we consider. For instance, the role *physician* is relevant in an hospital but probably not in a bank whereas the role *cashier* is relevant in a bank but not in an hospital. So, we need to specify which roles, activities and views are relevant in each organization. This is modelled by the following relationships:

- *Relevant_role* is a relation over $Org \times \mathcal{R}$

If org is an organization and r a role, then $Relevant_role(org, r)$ means that playing role r is defined in organization org . For instance, we may have $Relevant_role(Rangueil, physician)$, which means, in hospital *Rangueil*, *physician* is a relevant role.

- *Relevant_activity* is a relation over $Org \times \mathcal{A}$

If org is an organization and a is an activity, then $Relevant_activity(org, a)$ means that performing activity a is defined in organization org . For instance, we may have $Relevant_activity(Rangueil, consult)$, which means, in hospital *Rangueil*, *consult* is a relevant activity.

- *Relevant_view* is a relation over $Org \times \mathcal{V}$

If org is an organization and v is a view, then $Relevant_view(org, v)$ means that using view v is defined in organization org . For instance, we may have $Relevant_view(Rangueil, medical_record)$, which means, in hospital *Rangueil*, *medical record* is a relevant view.

3.4 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. For instance, we may have $Empower(Rangueil, ICU31, intensive_care_unit)$: “the Rangueil hospital empowers ICU31 as an intensive care unit”.

- *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.

3.5 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 contexts *Night* and *Attending_physician* as follows:³

- $\forall s, \forall \alpha, \forall o, (Define(H1, s, \alpha, o, Attending_physician) \leftrightarrow patient_name(o) \in patient(s))$
that is, in H1, the context “attending physician” is *true* between subject *s*, action α and object *o* if and only if *o* is a record corresponding to a patient of subject *s*.
- $\forall s, \forall \alpha, \forall o, \forall c_1, \forall c_2, (Define(H1, s, \alpha, o, Night) \leftrightarrow (20 : 00 \leq time(global_clock) \vee time(global_clock) \leq 8 : 00))$
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*.

We also consider conjunctive, disjunctive and negative contexts. For this purpose, we introduce functions $\&$, \oplus and $\bar{}$. If c_1 and c_2 are two contexts, then $\&(c_1, c_2)$ is a conjunctive contexts, $\oplus(c_1, c_2)$ is a disjunctive context and \bar{c}_1 is a negative context. We shall use the infix notations $c_1 \& c_2$ and $c_1 \oplus c_2$ in place of the prefix notations $\&(c_1, c_2)$ and $\oplus(c_1, c_2)$. These composed contexts are defined by the following rules:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c_1 \in \mathcal{C}, \forall c_2 \in \mathcal{C}, (Define(org, s, \alpha, o, c_1 \& c_2) \leftrightarrow Define(org, s, \alpha, o, c_1) \wedge Define(org, s, \alpha, o, c_2))$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c_1 \in \mathcal{C}, \forall c_2 \in \mathcal{C}, (Define(org, s, \alpha, o, c_1 \oplus c_2) \leftrightarrow Define(org, s, \alpha, o, c_1) \vee Define(org, s, \alpha, o, c_2))$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c \in \mathcal{C}, (Define(org, s, \alpha, o, \bar{c}) \leftrightarrow \neg Define(org, s, \alpha, o, c))$

³In the remainder of this paper, we shall use a logical notation to represent relationship: if *R* is a n-ary relationship over domains $D_1 \times \dots \times D_n$, then the predicate $R(d_1, \dots, d_n)$ is *true* if and only if $\langle d_1, \dots, d_n \rangle \in R$.

3.6 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*.

3.7 Deriving concrete permission

The relationship *Permission* enables a given organization to specify permissions between roles, activities and views in a given context. We call that *abstract* permissions. 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 abstract 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 [13], 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 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

We focus in this paper on the user-role assignment, the permission-role assignment and the user-permission assignment. Other administration activities are more briefly discussed in section 4.5. The approach we suggest in AdOr-BAC is to define these administration functions by considering 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 a 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 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, namely a role *physician* and a role *administrative_physician*. The AdOr-BAC model does not impose to create these two roles. But, as a security policy designer could legitimately 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

4.2.1 The view URA

The aim of the user-role administration activity is to determine who is allowed to assign a user to a role and in which conditions. Assigning a user to a role equals to inserting a new objet in a given view called *URA*. Three attributes are associated with this view: *subject* to designate the subject that 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.

If a given role is allowed to insert any object in view *URA* in every condition, then this means that this role is actually allowed to assign any user to any role in any organization in every condition. This would be generally not sufficiently restrictive. To enforce further restrictions, we have to define sub-views of view *URA*. For example, if the role *head_cardio_team* is only allowed to assign a user to the role *physician* in the department of cardiology *H_cardio_dpt* of a given hospital *H*, we have to create the *URA_physician_cardio_dpt* view. This view is a sub-view of *URA* defined as follows:

- $\forall ura$
 $Use(H, ura, URA_physician_cardio_dpt) \leftrightarrow$
 $Use(H, ura, URA) \wedge$

$$\begin{aligned} \text{role}(ura) &= \text{physician} \wedge \\ \text{org}(ura) &= H_cardio_dpt \end{aligned}$$

The role *head_cardio_team* will then be permitted to assign subject to the view *URA_physician_cardio_dpt* (see section 4.2.2 below). We assume that this means that a subject playing the role *head_cardio_team* is allowed to insert an object in the view *URA_physician_cardio_dpt* if this object corresponds to the definition of this view. This is similar to the “WITH CHECK OPTION” used in relational databases to control that an object can be inserted in a view only if this object matches the definition of the view.

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:

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

This rule says that from each object *ura* belonging to the view *URA* (or to any sub-views of *URA*), we can derive that a given subject (corresponding to *subject(ura)*) is empowered in a given role (corresponding to *role(ura)*) in a given organization (corresponding to *org(ura)*). Notice that in this rule, there are actually two different organization, namely *org* and *org(ora)*. This means that a user empowered in a given organization corresponding to *org* can actually manage the user-role administration activity 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.

4.2.2 Managing the view *URA*

There are three different activities that apply to view *URA*: *manage*, *assign* and *revoke*. The activity *assign* corresponds to assigning a user to a role through the view *URA*. For instance, the permission granted to the role *head_cardio_team* to assign a user to the role *physician* in the department of cardiology *H_cardio_dpt* of the hospital *H* is expressed as follows:

- $Permission(H, head_cardio_team, assign, URA_physician_cardio_dpt, Default)$

Up to now, we have only dealt with assignment but not with revocation. For this purpose, we use the activity *revoke*. For instance, we may specify:

- $Permission(H, head_cardio_team, revoke, URA_physician_cardio_dpt, Default)$

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*. This means that, if a given role is permitted to *manage* a given view in a given context, then this role is also permitted to perform *assignment* and *revocation* of this view in the same context. This is modelled by the following rule:

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

Remember that, in Or-BAC, we make a distinction between the activity and the action that actually implements this activity. This means that implementation of activities *assign* and *revoke* may change from one organization to another. For instance, in a relational database, activity *assign* may be implemented by the action “INSERT” (of objects in view *URA*) and activity *revoke* by the action “DELETE” (of objects in view *URA*). If this is the case in organization *H*, it is specified by the following facts:

- $Consider(H, INSERT, assign)$
- $Consider(H, DELETE, revoke)$

4.2.3 The prerequisite conditions

In the ARBAC model, there is the following ternary relation *can_assign*:

- $can_assign(admin_role, regular_role, prerequisite)$

This relation is used to specify that some *admin_role* is permitted to assign subjects to some *regular_role* provided that these subjects are empowered in some *prerequisite* role. In AdOr-BAC, there is no such ternary relation. However, it is possible to specify a similar requirement as follows. First, there is a permission having the following form⁴:

- $Permission(org, admin_role, assign, URA_regular_role, context)$

It is then possible to include the prerequisite condition when specifying the view *URA_regular_role* as follows:

- $\forall ura, Use(org, ura, URA_regular_role) \leftrightarrow$
 $Use(org, ura, URA) \wedge$
 $role(ura) = regular_role \wedge$
 $Empower(org, subject(ura), prerequisite)$

To illustrate the approach, let us consider the following example:

- In hospital *H*, the director is permitted to assign a user as the head of the cardiology department but only if this user is empowered in the role *physician* (prerequisite):

$Permission(H, director, assign, URA_head_cardio_dpt, Default)$

where the view *URA_head_cardio_dpt* is defined as follows:

- $\forall ura, Use(H, ura, URA_head_cardio_dpt) \leftrightarrow$
 $Use(H, ura, URA) \wedge$
 $role(ura) = head_dpt \wedge$
 $org(ura) = H_cardio_dpt \wedge$
 $Empower(H, subject(ura), physician)$

Remember that a subject can only insert an object in a given view if this object matches the definition of this view. This means that a subject empowered in the role *director* can only assign another subject in view *URA_head_cardio_dpt* if this subject is empowered in role *physician*. We can thus specify that for the department of cardiology the head must be a physician.

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. It is also important to mention that in the AdOr-BAC model there is no distinction between the regular roles and the administrative roles as suggested in ARBAC. Since, there is no specific permissions for the administrative tasks such as *can_assign* and *can_revoke* in ARBAC, permissions corresponding to the activities *manage*, *assign* and *revoke* can be given to any role, and not only to specific roles such as *senior_security_officer* or *project_security_officer*.

4.3 PRA in AdOr-BAC

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

⁴Notice that the organization *org* and the context *context* cannot be specified using the *can_assign* relation

4.3.1 The view *PRA*

Five attributes are associated with the view *PRA*:

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

There is a link that specifies that each object belonging to view *PRA* corresponds to a permission. This is modelled as follows:

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

This rule says that from each object *pra* belonging to the view *PRA* (or to any sub-views of *PRA*) in any organization *org*, we can derive that, in a given organization (corresponding to *issuer(pra)*), a given role (corresponding to *grantee(pra)*) is permitted to perform a given activity (corresponding to *privilege(pra)*) on a given view (corresponding to *target(pra)*) in a given context (corresponding to *context(pra)*).

4.3.2 Managing the view *PRA*

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

4.3.3 Prerequisite conditions

We can consider many examples of constraints that apply to permission-role assignment. As a special case, let us consider the prerequisite condition suggested in the ARBAC model. As mentioned in section 2.2, there is a ternary relation *can_assignp* in ARBAC defined as follows:

- $can_assignp(admin_role, regular_role, prerequisite)$

The relation *can_assignp* is used to specify that some *admin_role* is permitted to assign permissions to some *regular_role* provided that these permissions are already assigned to some *prerequisite* role. Notice that the meaning of the prerequisite condition differs from *can_assign* to *can_assignp*. In the *can_assign* relation, the *subject* must be empowered in the prerequisite role before being empowered in the regular role. In the *can_assignp* relation, the *permission* must be assigned to the prerequisite role before being assigned to the regular role.

In AdOr-BAC, it is possible to specify a similar prerequisite requirement as follows. First, we specify a permission having the following form:

- $Permission(org, admin_role, assign, PRA_regular_role, context)$

We then include the prerequisite condition when specifying the view *PRA_regular_role* as follows:

- $\forall pra, Use(org, pra, PRA_regular_role) \leftrightarrow$
 $Use(org, pra, PRA) \wedge$
 $grantee(pra) = regular_role \wedge$
 $Permission(issuer(ura), prerequisite, privilege(ura), target(ura), context(ura))$

Notice that our approach makes explicit the different meaning of the prerequisite condition between *can_assign* and *can_assignp*.

In AdOr-BAC, this prerequisite condition is simply a special case of constraint we can consider in the PRA assignment. We can actually consider many other examples of such constraint. For instance, we can specify that a given role is only permitted to assign particular *activities* to some regular role. The following provides an example.

- The head staff is permitted to grant the nurses of the department of cardiology a permission to consult the medical records in a context of emergency:

$Permission(H, head_staff, assign, PRA_nurse_consult_med_record, Default)$

The view $PRA_nurse_consult_med_record$ is defined as follows:

$$\begin{aligned} \forall pra, Use(H, pra, PRA_nurse_consult_med_record) \leftrightarrow \\ & Use(H, pra, PRA) \wedge \\ & issuer(pra) = cardio_dpt \wedge \\ & grantee(pra) = nurse \wedge \\ & privilege(pra) = consult \wedge \\ & target(pra) = medical_record \wedge \\ & context(pra) = emergency \end{aligned}$$

4.4 UPA in AdOr-BAC

The URA and PRA components respectively allow an authorized user to assign users to roles and (abstract) permissions to roles. Thus, these components indirectly enable this authorized user to assign (concrete) permissions to users. We argue that sometimes a more direct process should enable a user to grant a concrete 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 UPA and UPA'. UPA enables an authorized user to grant another user a permission to perform a specific action on a specific object. UPA' is more general. It enables an authorized user to grant another user a permission to perform a given activity on a given view. To avoid redundancy, we only present UPA; UPA' can be similarly defined. We shall then analyze how UPA applies to model the concept of delegation.

4.4.1 UPA: granting permissions on specific objects and actions

In this case, we consider a view UPA with five attributes having the same names as PRA but with slightly 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 UPA, the fact that a subject is permitted to perform an action on an object. This is modelled by the following rule:

- $\forall org, \forall upa,$
 $Use(org, upa, UPA) \wedge$
 $Define(issuer(upa), grantee(upa), privilege(upa), target(upa), context(upa))$
 $\rightarrow Is_permitted(grantee(upa), privilege(upa), target(upa))$

that is, if an object upa is used by a given organization org in view UPA 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 invoke his privilege on the target.

The concrete 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 H, 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 *UPA_Sec_Update_Med_Record* (for Secretary Updates Medical Record) of view UPA defined as follows:

- $\forall upa, Use(H, upa, UPA_Sec_Update_Med_Record) \leftrightarrow$
 $Use(H, upa, UPA) \wedge$
 $Empower(H, grantee(upa), medical_secretary) \wedge$
 $Consider(H, privilege(upa), update) \wedge$
 $Use(H, target(upa), medical_record)$

that is object *upa* is used in view *UPA_Sec_Update_Med_Record* if and only if it is used in view UPA 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:

- $Permission(H, physician, assign, UPA_Sec_Update_Med_Record, Physician_Sec_Patient)$

that is, in H, the role *physician* is permitted to assign a permission in view *UPA_Sec_Update_Med_Record* in context *Physician_Sec_Patient* (for Physician's Secretary and Patient). The context *Physician_Sec_Patient* is used to constrain that the role *physician* can only grant a permission to a subject corresponding to his or her secretary and on an object corresponding to a medical record of one of his or her patient. *Physician_Sec_Patient* is defined as follows:

$$\forall s, \forall a, \forall upa, Define(H, s, a, upa, Physician_Sec_Patient) \leftrightarrow$$

$$Empower(H, s, physician) \wedge Use(H, upa, UPA) \wedge$$

$$grantee(upa) \in secretary(s) \wedge$$

$$Use(H, target(upa), medical_record) \wedge patient_name(target(upa)) \in patient(s)$$

that is, H defines that subject *s* performs action *a* on object *upa* in context *Physician_Sec_Patient* if *s* is a physician in H, *upa* is used in view UPA, the grantee of *upa* is a secretary of *s* and the target of *upa* is used as a medical record corresponding to a patient of *s*.

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

4.4.2 Application to delegation

Modelling delegation is a complex problem. The analysis performed in [2] 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 UPA or UPA' (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 constraint is modelled by a context *AG* (for Authorized Grantor) defined as follows:

- $\forall org, \forall s, \forall a, \forall upa,$
 $Define(org, s, a, upa, AG) \leftrightarrow$
 $Use(org, upa, UPA) \wedge$
 $Is_permitted(s, privilege(upa), target(upa))$

that is, in any organization org , subject s performs action a on object upa in context AG if org uses upa in view UPA^5 and s is permitted to perform the action corresponding to $privilege(upa)$ on the object corresponding to $target(upa)$.

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 [7]. Another possible restriction is that the grantor will lose the permission he has delegated. In AdOr-BAC, this means that delegation is not an elementary activity but the combination of assigning a permission (as modelled in UPA or UPA') and self-revoking this permission on the grantor (this may be also modelled in UPA or UPA'). We do not further develop this analysis of the delegation concept in this paper. We plan to continue this investigation in the future.

4.5 Other administration functions

In previous sections, we show how to model user-role assignment (by the URA view), permission-role assignment (by the PRA view) and user-permission assignment (by the UPA view). We model other administrations in Or-BAC by using the following views:

- *Organization* and *Subject*.

Inserting or deleting objects in these views enables a subject empowered in an authorized role to respectively create or delete organizations and subjects. In our model, views *Organization* and *Subject* have at least the attribute *name* to respectively represent the organization or subject name and possibly other application specific attributes to model other organization or subject attributes.

- *ROA* (for Role-Organization assignment), *AOA* (for Activity-Organization assignment) and *VOA* (for View-Organization assignment).

Inserting objects in these views respectively enables a subject empowered in an authorized role to assign roles (resp. activities) (resp. views) to a given organization. These views have two attributes: *role* (resp. *activity*) (resp. *view*) and *org*. From objects belonging to these views, we can respectively derive instances of relations *Relevant_role*, *Relevant_activity* and *Relevant_view*. This is modelled by the following rule:

$$\forall org, \forall roa, \\ Use(org, roa, ROA) \wedge \\ \rightarrow Relevant_role(org(roa), role(roa))$$

and similarly for relations *Relevant_activity* and *Relevant_view*.

As suggested at the end of section 3.7, the Or-BAC model also includes the possibility to specify hierarchies of roles, activities and views. We can manage these hierarchies using the following views:

- *RHA* (for Role-Hierarchy administration) *AHA* (for Activity-Hierarchy administration) and *VHA* (for View-Hierarchy administration).

View *RHA* is used to specify that in a given organization, a given role is a sub-role of another role. This view has three attributes: *org*, *sub_role* and *super_role*. Views *AHA* and *VHA* are similarly defined by respectively replacing role by activity or view.

5 Application

We have developed OToKit (for Or-BAC Tool-Kit), a module that allows authorized users to define security policies in Or-BAC and their administration in AdOr-BAC. OToKit is implemented in Prolog with an interface in Java. The administration policy specified in the AdOr-BAC model is also used to control accesses to OToKit so that only authorized users will be allowed to create new organizations and subjects, assign roles, activities and

⁵The definition of context AG must be slightly changed if we consider view UPA' .

views to organizations, and in each organization, assign permissions to roles, subjects to roles and permissions to subjects.

When an authorized user s wants to define the security policy of a given organization org , this user activates OToKit with org as a parameter. OToKit will then create a new policy base containing the following entities and facts:

- Two organizations org and org_admin (for org administration)
- Subject s
- Role $policy_designer$
- Fact $Relevant_role(org_admin, policy_designer)$
- Fact $Relevant_activity(org_admin, manage)$ and two other similar facts to specify that activities $assign$ and $revoke$ are relevant to organization org_admin
- Fact $Relevant_view(org_admin, Organization)$ and other similar facts to specify that views $Subject$, ROA , AOA , VOA , URA , PRA and UPA are relevant to organization org_admin
- Fact $Empower(org_admin, s, policy_designer)$ to specify that subject s is empowered to role $policy_designer$ in organization org_admin
- Fact $Permission(org_admin, policy_designer, manage, PRA, Default)$ to specify that, in organization org_admin , role $policy_designer$ is permitted to manage view PRA in every situation (corresponding to context $Default$)
- Facts $Consider(org_admin, assert, assign)$ and $Consider(org_admin, retract, revoke)$. These two facts are specific to our implementation in Prolog. They say that the actions $assert$ (used to insert new facts in a Prolog base) and $retract$ (used to delete facts from a Prolog base) respectively correspond to $assign$ and $revoke$ activities

This initial base specifies that, in organization org_admin , the subject s is empowered in role $policy_designer$ and the role $policy_designer$ is permitted to manage the view PRA . Thus, s can define any new permissions and fully specify the security policy of org . Generally, s will start by creating roles that are relevant to organization org . This is why, it is convenient to also include in the initial base, the following permission:

- $Permission(org, policy_designer, assign, ROA, Default)$

However, notice that s may *itself* create this permission by inserting in view PRA a new object pra_1 such that:

- $issuer(pra_1) = org \wedge grantee(pra_1) = policy_designer \wedge privilege(pra_1) = assign \wedge target(pra_1) = ROA \wedge context(pra_1) = Default$

Subject s can also define the administration policy of org . AdOr-BAC is sufficiently flexible to define any type of administration policy, from completely centralized to fully distributed ones. In particular, s can create other subjects and assign them roles so that these subjects can participate to the security policy specification. Notice that AdOr-BAC is also used to control the accesses to OToKit so that only *authorized* users can update the security policy. This is because defining a given security policy actually corresponds to inserting objects in specific views. Thus we can actually control that subjects who are attempting to update the security policy are permitted to do so by simply controlling that they are permitted to insert objects in the corresponding views. This is why we say that Or-BAC is a fully auto-administered model.

6 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) and a unique predefined role *policy-designer* assigned to the creator. The role *policy-designer* has permissions to define roles to administer the 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 Permission *delegation*, when a user wants to grant another user a specific permission. We suggest two variations of the UPA component: UPA that enables a user to delegate a permission to perform a specific action on a specific object and UPA' 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 [2], there are several different characteristics related to delegation such as permanence, monotonicity, totality, levels of delegation or 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.

Consistency analysis is another important problem. In this paper, we restricted the presentation to permission and do not consider prohibition. Introducing prohibition create potential conflicts both in Or-BAC and AdOr-BAC. In Or-BAC or AdOr-BAC, a conflict exists when a subject s is both permitted and prohibited to perform an action α on an object o . Such a situation can appear if there is a permission granted to role r_1 , activity a_1 and view v_1 and there is a prohibition associated with role r_2 , activity a_2 and view v_2 . If s may be assigned to both roles r_1 and r_2 and o may be used in both views v_1 and v_2 and α may take part in both activities a_1 and a_2 , then there is a conflict. Another kind of potential conflict exists between Or-BAC and AdOr-BAC for instance when there is a prohibition associated with role r_1 , activity a_1 and view v_1 , and using PRA, there is a permission that allows a role r_2 to grant a permission to role r_1 , activity a_1 and view v_1 . We are currently investigating these different types of conflict. We plan to present our results in a forthcoming paper.

Acknowledgement

For this work, Alexandre Miège is funded by France Télécom R&D and Frédéric Cuppens is partially funded by the MP6 RNRT project of the ministry of Research. The authors thank Nora Cuppens-Boulahia for helpful comments about this paper.

References

- [1] Ezedin Barka and Ravi Sandhu. A Role-Based Delegation Model and Some Extensions. In *Proceedings of the 23rd National Information Systems Security Conference*, Baltimore, MD, October 2000.
- [2] 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.

- [3] J. Carrre, F. Cuppens, and C. Saurel. SACADDOS: a support tool to manage multilevel documents. In *Database Security, 12: Status and Prospects. Results of the IFIP WG 11.3 Workshop on Database Security*, Fairfax, Virginia, 1999. Kluwer Academic Press.
- [4] T. Chen. A linear time algorithm for deciding security. In A. K. Jones, R.J. Lipton, and L. Snyder, editors, *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, 1976.
- [5] 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.
- [6] F. Cuppens and A. Gabillon. Modelling a Multilevel Database with Temporal Downgrading Functionalities. In *Database Security, 9: Status and Prospects. Results of the IFIP WG 11.3 Workshop on Database Security*, North Holland, 1996.
- [7] F. Cuppens and A. Miège. Modelling Contexts in the Or-BAC Model. In *Proceedings of 19th Applied Computer Security Associates Conference (ACSAC 2003)*, Las Vegas, Nevada, December 2003.
- [8] 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.
- [9] David F. Ferraiolo, Dennis M. Guilber, and Nickilyn Lynch. An examination of federal and commercial access control policy needs. In *Proceedings of the 16th NIST-NSA National Computer Security Conference*, pages 107–116, Baltimore, MD, September 1993.
- [10] 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.
- [11] S. I. Gavrilu 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.
- [12] 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.
- [13] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, 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.
- [14] 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.
- [15] 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.
- [16] R. Sandhu, V. Bhamidipati, E. Coyne, S. Ganta, and C. Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedins of the 2nd ACM Workshop one Role-Based Access Control*, Fairfax, Virginia, November 1997.
- [17] 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.
- [18] 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.

- [19] 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.
- [20] 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.