Multi-Granular Licences to Decentralize Security Administration

Frédéric Cuppens and Nora Cuppens-Boulahia and Céline Coma

GET/ENST Bretagne, 2 rue de la chataîgneraie,
35576 Cesson-Sévigné Cedex, France

There has been for several years a growing interest in defining new access control models and administration facilities for these models. Several models have observed that only structuring the model using the concept of roles as in RBAC is not sufficient to administer decentralized enterprises. These models have suggested to consider new concepts such as *organization* (as in OrBAC) or *domain* (as in GTRBAC) that make it easier to establish secured entreprise communications. In this paper, we suggest a self and decentralized object-oriented administration model built on the concept of *organization*. This model is merely based on two mechanisms: confinement and licence. Confinement restricts the authority of a subject to the organization (or sub-organizations) to which this subject has been assigned administration privileges. Licence introduces *multi grain* and *contextual* administration facility. It is used to define administration and delegation privileges, to whom they are assigned, in which context they became effective and the scope of each of them.

Keywords: Administration, Access Control, OrBAC, Authorization, Confinement, Licence

1 Introduction

An administration policy defines who is permitted to manage the security policy, i.e. who is permitted to create new security rules, or update or revoke existing security rules. Classical security models generally assume that administration is centralized: there is a single role, for instance called *security administrator* or *super user*, and subjects empowered in this role have full capacities to administer the security policy.

However, information systems tend to become more complex and more distributed. In this context there is a real need for a distributed administration model that is close to the security model it administers. Our objective in this paper is to define a security model that provides means to specify such kinds of decentralized administration policies. Our approach is based on the following four principles: (1) Organizational unit, (2) Confinement, (3) Multi-grained administration and (4) Self administration.

Our proposal for a decentralized administration model is structured around the concept of *organizational unit*. In the ARBAC administration model [22, 21, 20] defined for the RBAC model [23, 15], administration was based on role hierarchy but this approach has several shortcomings (see section 2.1). Some of these problems have been raised by previous models such as ERBAC (Enterprise RBAC) [18] or X-GTRBAC Admin (an administration model for the Generalized Temporal RBAC Model) [6] which respectively introduce the concepts of *scope* and *domain* for administration purpose. In our model, we suggest using organization as a basic concept to define the authority scope of administrators.

Notice that the concept of role is also used in our model, since each organization can be associated with a set of roles. However, capacities of a subject assigned to a role in a given organization is restricted by the organizational *confinement* principle. This principle provides a solution to a problem highlighted in [22] of undesired flow of permissions through the role hierarchy.

Another characteristic of the security model we suggest is that it can be used to define multi-grained administration. For instance, in RBAC, the only way to grant a permission to a subject is by granting this permission to a

given role and then assigning this role to the subject. This rigid scenario does not answer the need of multi-grained administration. In a multi-grained administration, an administrator can be permitted to grant a permission to a role or also directly to a given subject. We claim that this facility is necessary to get more flexible administration policies, in particular *delegation* facilities. Multi-grained administration also makes it possible to restrict the authority of some administrator so that he or she will be allowed to grant only permissions having some specific properties (a requirement stated in [22]).

The last principle of our proposal is that our security model must be self administrated, that is the concepts used to define an administration policy are similar to the ones used to define the remainder of the security policy. Thus, a self administrated security model is based on a unique set of concepts. The net advantage is that it is not necessary to consider administrative roles or permissions that are separated from other regular roles or permissions as suggested in ARBAC or X-GTRBAC. In our approach, some role may cumulate both regular and administrative permissions. For instance, one can specify that a role R_1 holds an administrative permission that allows R_1 to grant some regular permission to another role R_2 if R_1 holds this regular permission.

The administration model we present in this paper follows the four above principles and is defined as an extension of OrBAC (Organization Based Access Control) [3]. The OrBAC structure provides some facilities for designing a flexible decentralized administration model. This model is based on the concept of organization (principle 1) and is associated with a self administrated model (principle 4) called AdOrBAC [13]. In this paper, we actually show how to extend OrBAC to provide means for multi-grained administration (principle 2) and formally define the administration confinement constraint (principle 3).

The main extension we specify in OrBAC to enable multi-grained administration is the concept of *licence*. Licence is a special class of objects used in several policy-based languages for DRM such as MPEG-REL [1]. Basically, a licence is an object similar to a capability [19] whose existence is interpreted as granting a permission to a subject to execute an action on an object. In our model, an administration capacity corresponds to specifying who is permitted to manage licences. This approach provides a fully self-administrated model. Moreover, to obtain an expressive and flexible administration model, we extend the concept of licence to manage *contextual* and *multi-grained* permissions.

In this paper, we shall only consider *positive* licences, namely licences interpreted as permissions. We may also consider negative licences that correspond to prohibitions. This will raise the problem of managing possible conflicts between positive and negative licences. For the sake of simplicity and due to space limitation, we leave this problem for a forthcoming paper.

This paper is organized as follows. Section 2 further elaborates on organizational unit, confinement, multi-grained administration and self-administrated principles and provides a complete comparison with other related works. Section 3 presents our self administration model. Section 4 shows how to apply our model to specify various types of administration requirements. Finally, section 5 concludes the paper.

2 Motivation

2.1 Need of an Organizational Structure

Several access control models have been developed for about twenty years. These models are not completely satisfactory. Moreover, their administrative models, if any, are often restrictive and centralized and not always formally defined. The RBAC model is currently the most commonly used security model. In ARBAC, the administrative model of RBAC, the authority scope of an administrative role is only restricted by the role hierarchy. This leads to several shortcomings. In particular, the RRA (role-role assignment) component requires to specify complex constraints to manage effects of modifying the hierarchy of administrative roles [22]. Moreover, the security administration provided by ARBAC is not really decentralized. Actually, there is always a unique root security administrator who manages the administrative roles, administrative hierarchy of roles and instantiates all the administrative information.

A-ERBAC is another proposal of role-based administrative approach [18]. A-ERBAC introduces the concept of *scope* of authority for administrative roles. More recently, GTRBAC [17] includes an administration model that

aims at enabling policy administration within a large entreprise. This administration model [6] uses the concept of *domain* to restrict the authority of the administrators. This provides a mechanism to distribute the administrative authority over multiple domains within the entreprise.

Both scopes and domains have some analogy with our concept of *organization* which is central in the OrBAC model and useful to model administration capabilities. On the contrary, our concept of organization is different from that of *organizational pool* suggested in [22]. We use *organization* to mean any entity who defines and/or manages a security policy. In an organizational structure, each administrator has some authority to assign permissions and possibly create new administrators in the organization. Hence, the administration is independent of the hierarchy of roles. The organization structure greatly simplifies management of both roles and role hierarchies. Compared with the ARBAC model, we get the following advantages: (1) In ARBAC, many roles would combine an organizational dimension with a functional dimension whereas our proposal clearly separates these two dimensions, (2) Thanks to this separation, the number of roles to be managed is smaller than in ARBAC and management of role hierarchy is simplified, (3) Organizations provides a natural mean to manage authority scope of administrators.

2.2 Need of an object oriented approach

Basically, the main objective of an access control policy is to define which subjects are permitted to perform which actions on which objects. Our approach to define an administration policy is essentially the same. To specify administrative security policies, some subjects will be permitted to create, update or delete special objects. These objects, called licences, will have a particular structure and specific meaning, namely the existence of a valid licence will be interpreted as the assignment of some permission.

To further explain our approach, we have simply to think to a flight ticket. There are several information printed on a flight ticket: (1) the *authority* which has issued the ticket, an airline company, (2) the *grantee* of the flight ticket, namely a user identity, (3) the *target* which corresponds to a given flight, (4) the *privilege* granted by the ticket, namely taking the flight described in the target, and (5) *contextual* conditions to be satisfied, for instance the ticket is valid if this user is travelling with her husband.

A valid flight ticket is interpreted as a permission, namely a permission to take the flight described on the ticket. Thus, a flight ticket may be viewed as a special case of licence. By analogy, we shall consider that a licence has five attributes called *authority*, *grantee*, *privilege*, *target* and *context*.

The administration policy will then specify who is permitted to manage flight tickets. For instance, the employees of some travel agency will be permitted to create flight tickets and possibly to update them in specific situations.

We also need to enforce the restriction that administrative permissions granted to these employees do not encompass any licences but only those dealing with flight tickets. For this purpose, we use the concept of *view* to cluster licences to which the same security rules apply (see section 3.3). This provides means to precisely define the administrative privileges granted to roles. Since licences include a contextual attribute, it is also possible to specify dynamic administration policies (see [14] for a taxonomy of contexts).

By contrast, ARBAC is not appropriate to specify security rules that depend on contextual conditions and its PRA (Permission-role assignment) component only allows unrestricted assignment of permissions to roles. A-ERBAC also only consider static security rules and X-GTRBAC is designed to manage temporal contexts which is a specific case of context. It also does not seem possible to specify in X-GTRBAC nested contextual administrative rules such as: By night (context 1), physicians are permitted to grant nurses the permission to consult medical records in a context of urgency (context 2). Notice that this rule is different from the following rule: By night and in a context of urgency, physicians are permitted to grant nurses the permission to consult medical records.

2.3 Need of multi-granular administration privileges

A flight ticket specifies precisely which subject is permitted to take a flight (performing an action) on a given flight from a specific departure city to a destination city at a given departure time (on a specific object). This is an example of fine grained licence. However, in many situations, it is more convenient to use more global licence.

For instance, let us consider an open flight ticket which permits to take any flight from a given departure city to some destination city. In this case, we need a mean to represent a set of flights that share the same departure and destination cities. For this purpose, a *view* can be used as a target value to define more global licences. For instance, a valid open flight ticket whose target value is equal to the view *Flight_Toulouse_to_Paris* allows the grantee to take any flight from Toulouse to Paris.

Similarly, we can consider licences whose grantee attribute is a *role* instead of a specific subject. In this case, every subject assigned to the role specified in the grantee attribute will get the privilege defined in the licence. Applying the same approach, it is also useful to manage licence whose privilege attribute is not a specific action but a set of actions. In the following, this is called an *activity*.

Thus, a licence is a polymorphic object used to manage multi-granular privileges. This is used to specify both "classical" administrative requirements (where a permission is assigned to a role) with *delegation* requirements (where a permission is assigned to a specific subject). In previous proposals, delegation is modelled separately from other administrative needs (see [4] for instance).

2.4 Need of a self administrated model

Licence is the central concept to define a fully self administrated model. Using this concept, it is straightforward to specify that a same role may cumulate both regular and administrative permissions.

By contrast, ARBAC, A-ERBAC and X-GTRBAC are all based on a strict separation between administrative and regular roles. As mentioned in the introduction, this is restrictive since some roles (for instance *physician*) combines regular permissions (for instance the permission to consult a medical record) and administrative permissions (for instance the permission to grant a permission to a nurse to consult some medical records in a context of urgency).

2.5 Need of Confinement Properties

The concept of organization is used to structure the security policy definition. It is also used to define *confinement* properties. The authority range of a role must be restricted by a confinement constraint. The objective of such a confinement constraint is to prevent a malicious administrator of an organization from creating licences that would illegally apply to another organization.

As suggested in [18], a confinement constraint is *strict* if a licence created by a subject who is empowered in an administrative role in a given organization can only apply inside this organization. However, the strict confinement constraint is generally too restrictive. Instead, in the following, we shall consider the hierarchical confinement constraint. In this case, the authority range of an administrative role is limited to its organization or to one of its sub-organizations.

Notice also that the hierarchical confinement constraint applies as a general constraint to restrict the authority of administrators. In our model, it will be possible to explicitly specify that the authority range of a given administrative role does not include some sub-organizations.

3 Our administration Model

Our administration model is defined as an extension of the OrBAC model [3]. One objective of OrBAC is to define a security policy independently of its implementation. In order to reach this goal, *subjects*, *actions* and *objects* are structured round the organization where these entities evolve. This organization conforms to a given security policy. OrBAC has two abstraction levels, say concrete and abstract levels linked by the organization where they are defined. The concrete level is based on subjects, actions and objects. The abstract level contains *roles*, *activities* and *views*. Basically, a role (activity, view respectively) is a set of subjects (actions, objects respectively) on which the same security rules apply. These security rules may depend on temporal aspects, prerequisite events and so on. In this case, the *context* entity of OrBAC model brings this flexibility [14]. In this section, we see how we manage these entities to extend the OrBAC model in order to deal with a decentralized security administration that complies with our four principles.

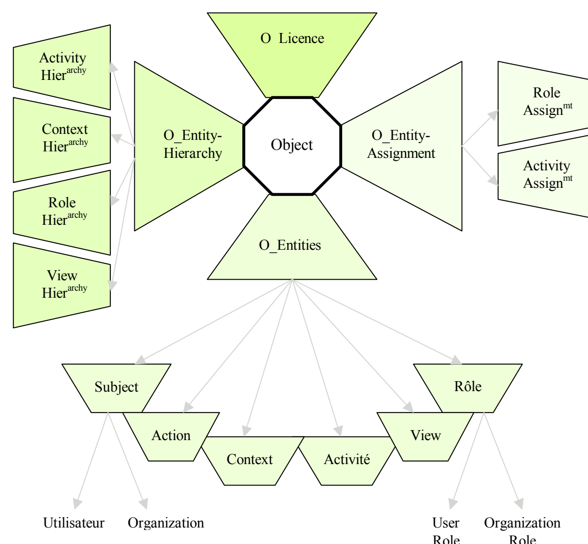

Figure 1: Administrative Views: All is Object

We use a formalism based on first order logic with negation. However, since first order logic is generally undecidable, we restrict our model to be compatible with a stratified Datalog program [24]. A stratified Datalog program is computable in polynomial time. To express rules and facts, we shall actually use a prolog-like notation[†].

The remainder of this section is organized as follows. Section 3.1 introduces the basic entities of object, subject, action and organization. Section 3.2 shows how to structure these basic entities using the concept of *view*. Section 3.3 defines the administrative concept of *licence*, shows how to model *contextual* licences, gives an interpretation of a licence in terms of permission and formally specifies the confinement principle associated with licence management. Sections 3.4 and 3.5 respectively introduce the concepts of *role* and *activity* and define two administrative concepts to respectively manage *subject-role assignment* and *action-activity assignment*. Section 3.6 shows how to use the concepts of view, role and activity to define *multi-granular* licences. Finally, section 3.7 deals with hierarchies and discusses the problem of illegal flow of permissions within the hierarchies.

3.1 Object, subject, action and organization

Our model is object-oriented. Thus, every entity corresponds to an object (see Figure 1). Each object has an *identifier* that uniquely identifies the object and a set of *attributes* to describe the object. Following a logical formalism, attributes are modelled as binary predicates. For instance, fact age(s1,20). says that the object whose identifier is s1 has an attribute age whose value is 20. Objects belong to classes. This is modelled using a binary predicate class. For instance, the fact class(s1,student). says that s1 belongs to the student class.

We consider three special classes of objects called subject, action and organization. Subjects correspond to active entities and can perform actions on objects. Actions are concrete programs. An organization corresponds to any entity that is responsible for managing a security policy. Organizations may have sub-organizations. This is modelled by a binary predicate sub_organization which defines a partial order relation on organizations.

Objects may be associated with *constraints* that correspond to special rules whose conclusion is the error predicate (as suggested in [16]). For instance:

```
error :- class(X,student), teach(X,Y), class(Y,course),level(Y,master_thesis).
```

This constraint specifies that a student cannot[‡] teach a master thesis course.

[†] Terms starting with a capital letter, such as Subject, corresponds to variables and terms starting with a lower case letter, such as peter, corresponds to constants. A fact, such as parent(peter,john)., says that peter is a parent of john. A rule, such as, grand_parent(X,Z) :- parent(X,Y),parent(Y,Z)., says that X is a grand-parent of Z if there is a subject Y such that X is a parent of Y and Y is a parent of Z

[‡] In the following, we assume that no action can violate a constraint. If executing a given action violates a constraint, then this action is rejected.

3.2 View

The concept of View is different from the concept of Class. A class is a taxonomical concept used to structure the object descriptions, i.e. a class groups objects that have similar characteristics. By contrast, a view is an organizational concept used to structure the policy specification, i.e. a view groups objects on which the same security rules apply.

In our model, each organization manages its views. For this purpose we introduce a ternary predicate use, then use(org,o,v) means that org uses object o in view v. For instance, use(enst_bretagne,c1,course)., says that organization enst_bretagne uses object c1 in view course.

A view may be defined (1) by enumerating facts specifying which objects are used in this view by a given organization or (2) by a logical rule called a *view definition*, e.g:

use(Org,C,graduate_course) :- use(Org,C,course),level(C,graduate).

This rule says that any organization Org uses object C in view graduate_course if Org uses C in view course and the attribute level of C is equal to graduate. Thus, a view definition corresponds to a condition which is used to automatically derive that some objets belong to a given view, say v. As in SQL, a view may be defined *with check option*. When the check option is used, any modification of the view will be rejected if it violates the view definition. In this case, the following additional constraint is inserted:

error :- use(Org,Obj,v), not (condition).

For instance, if the above view graduate_course is created *with check option*, then the following constraint will be inserted: error :- use(Org,C,graduate_course), not (use(Org,C,course),level(C,graduate)).

When a subject creates an object obj, then a view v and an organization org must be specified. If the security policy of organization org permits that this subject creates this object in this view, then use(org,obj,v). is inserted in the theory.

3.3 Licence and confinement principle

The licence class. The class licence is used to specify and manage the security policy. In this section, we only consider fine-grained licences that apply to subjects, objects and actions. In section 3.6, we shall model more global licences.

Objects belonging to the licence class have the following attributes: (1) authority: organization in which the licence applies, (2) grantee: subject to which the licence is granted, (3) privilege: action permitted by the licence, (4) target: object to which the licence grants an access and (5) context: specific conditions that must be satisfied to use the licence.

One can notice that the licence class is associated with an attribute called context we did not discuss so far. Contexts are used to specify conditions, for example working_hours or urgency. To model contexts, we introduce a class of objects called context and a 5-place predicate hold. If org is an organization, s is a subject, a an action, o an object and c a context, then hold(org,s,a,o,c) means that, within organization org, subject s executes action a on object o in context c.

Conditions that must be satisfied to derive that a context is active are modelled by a logical rule called *context definition*. We have defined five kinds of contexts: Temporal, Spatial, Provisional, Prerequisite, user-declared (see [14] for more details). In the following, we shall also consider a context called nominal that is always active for any organization, subject, action and object.

Licence interpretation and confinement. The following constraint enforces the *strict confinement principle*:
error :- use(Org,L,licence), authority(L,Auth), Org != Auth.

It specifies that a licence is only valid if it applies to the same organization Auth as the organization Org that uses this licence. The strict confinement principle is generally too restrictive. Within an organization, it must be also possible to create licences that apply to some of its sub-organizations. This is modelled by the following constraint called *hierarchical confinement principle*:

error :- use(Org,L,licence), authority(L,Auth), not (sub_organization(Org,Auth)).

We interpret the existence of a licence as a permission. For this purpose, we introduce the predicate permission and define the following rule:

permission(Auth,Subject,Action,Object,Context) :-
use(Org,L,licence), authority(L,Auth), grantee(L,Subject), privilege(L,Action), target(L,Object), context(L,Context), sub_organization(Org,Auth).

We use the following rule to derive actual permission for some subject, action and object when the context is active:
is_permitted(Subject,Action,Object) :- permission(Auth,Subject,Action,Object,Context), hold(Auth,Subject,Action,Object,Context).

3.4 Role and role assignment

Licences modelled in the previous section can only be used to grant a fine grained permission that applies to a given subject, action and object. As suggested in the RBAC model, it is convenient to grant permissions to roles and then assign subject to roles. For this purpose, we first consider a view role. Roles are managed as other objects. So, roles can be created. For instance, if the role professor is created in organization enst_bretagne, then the following fact is inserted: use(enst_bretagne,professor,role).

Assigning subject to roles is a part of the administration policy. To manage assignment of subject to roles, we introduce a class of objects role_assignment. This class is associated with the three following attributes: (1) authority: organization in which the role assignment applies, (2) assignee: subject to which the role is assigned and (3) assignment: role assigned by the role assignment.

Within an organization, we assume that a role assignment is valid if it applies to the organization itself or to some of its sub-organizations. Thus, the hierarchical confinement principle, defined in section 3.3, applies to role assignments:
error :- use(Org,RA,role_assignment), authority(RA,Auth), not (sub_organization(Org,Auth)).

To interpret the existence of a role assignment, we introduce the predicate empower: if org is an organization, s a subject and r a role, then empower(org,s,r) means that s is empowered in role r in organization org. We then define the following logical rule to interpret objects that belong to the role_assignment view in terms of the empower predicate:
empower(Auth,Subject,Role) :-
use(Org,RA,role_assignment), authority(RA,Auth),
assignee(RA,Subject), assignment(RA,Role), sub_organization(Org,Auth).

3.5 Activity and activity assignment

Roles provide a convenient abstraction of subjects to define a security policy. In a similar way, we suggest abstracting actions into another concept called *activity*. Actions correspond to application dependant programs. By contrast, activities is an application independent concept which, in some sense, represents the role of actions in the system.

To model activities, we first consider a view activity and each organization will have to create its activities. In the following, we assume that these activities include two atomic activities called create and delete to be used to create or delete objects, including licences. However and by contrast with other administration models such as [20, 18, 6], we do not restrict the administration activities to only creation and deletion. Thus an organization may define other administration activities. Some of these activities could be non atomic. Management of security policies that include non atomic activities is outside the scope of this paper but is discussed in [11].

When implementing the policy, it is then necessary to assign concrete actions to activities, including administrative activities. For instance, in a relational database, create and delete activities will respectively correspond to INSERT and DELETE actions.

Assigning actions to activities is clearly another part of the administration policy. To manage assignment of actions to activities, we introduce a new class called activity_assignment which is associated with the same three following attributes as role_assignment: (1) authority, (2) assignee and (3) assignment.

We assume that objects in the activity_assignment class also satisfies the hierarchical confinement principle. To interpret the existence of an activity assignment, we introduce the predicate consider: if org is an organization, act an action and a an activity, then consider(org,act,a) means that act is an implementation of activity a in organization

org. We define the following logical rule to interpret objects that belong to the activity_assignment view in terms of the consider predicate:

```
consider(Auth,Action,Activity) :-
    use(Org,AA,activity_assignment), authority(AA,Auth),
    assignee(AA,Action), assignment(AA,Activity), sub_organization(Org,Auth).
```

3.6 Multi-granular administration privileges

We can now generalize licences of section 3.3 by considering that the attribute grantee of a licence can be a subject or a role, the attribute privilege can be an action or an activity and the attribute target can be an object or a view.

If the grantee is a role, we need an additional rule to derive that a given permission is granted to a subject, when a permission is granted to a role and this subject is assigned to this role:

```
permission(Auth,Subject,Action,Object,Context) :-
    permission(Auth,Role,Action,Object,Context),
    use(Auth,Role,role), empower(Auth,Subject,Role).
```

There are two other similar rules to respectively interpret licences when the attribute privilege is an activity and the attribute target is a view.

We can further generalize the licence structure by considering that attributes authority can be an organization view (instead of an organization). Generalizing organization by organization view in the attribute authority is done by the following rule:

```
permission(Auth,Subject,Action,Object,Context) :-
    use(Org,L,licence), authority(L,View_Auth),
    use(Org,View_Auth,view), use(Org,Auth,View_Auth),
    class(Auth,organization), grantee(L,Subject),
    privilege(L,Action), target(L,Object), context(L,Context).
```

With this generalization, we can consider licences that grant an access to every sub-organization of a given organization that belongs to a given view specified in the licence. For instance, the organization enst_bretagne can create licences that apply to all its campus.

Thus, with this model, we can specify a large variety of permissions, from very specific ones that apply, in an organization, to some subject, action and object, to very general ones that apply to some role, activity and view.

3.7 Hierarchies

Hierarchy of roles has been investigated in many security models, especially RBAC. This hierarchy is generally associated with inheritance of permissions. It is an effective concept to simplify the security policy specification. In [10], we consider other hierarchies that apply to activity, view and context and associate these hierarchies with inheritance of permissions through these hierarchies. We have investigated how to administer these hierarchies.

As in the previous sections, our approach is based on managing a special class of objects called role_hierarchy. This class has the three following attributes: (1) authority, (2) senior_role and (3) junior_role. Objects belonging to class role_hierarchy are interpreted as follows: in a given organization, a given senior role will inherit permissions from a given junior role. We then consider a 3-place predicate role_inheritance defined as the following: if org is an organization, srole a role and jrole another role, then role_hierarchy(org,srole,jrole) means that role srole inherits permissions from role jrole in organization org. We define the following logical rules to interpret objects that belong to role_hierarchy:

```
role_inheritance(Auth,Srole,Jrole) :-
    use(Org,RH,role_hierarchy), authority(RH,Auth),
    senior_role(RH,Srole), junior_role(RH,Jrole), sub_organization(Org,Auth).
```

We have actually defined two different constraints respectively: Constraint associated with creation of a role hierarchy and Constraint associated with creation of a licence. But due to space limitation, we do not go in further details in this paper.

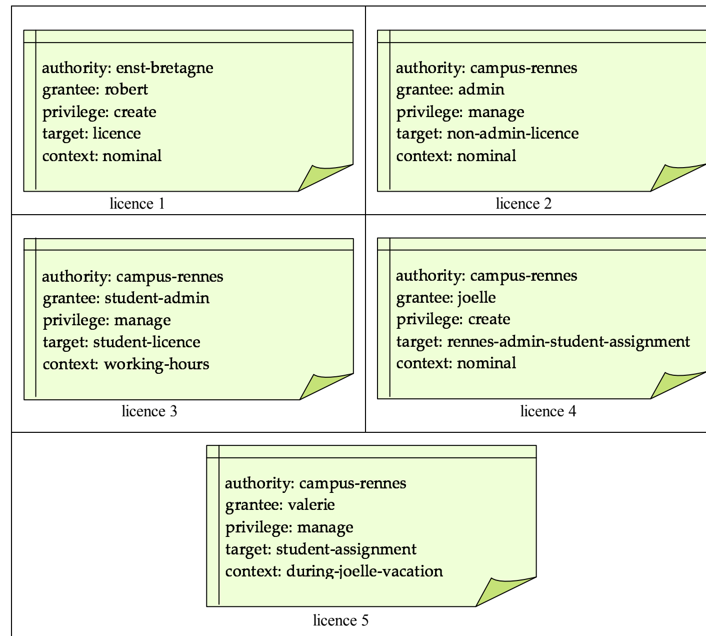

Figure 2: Licence examples

4 Policy administration examples

4.1 Organizational administration

In this section, we show how to use our model to specify some administration security requirements of an organization such as the ENST Bretagne.

Let us consider a subject, called robert, who is granted the permission to design the security policy of the ENST Bretagne. For this purpose, robert has first to create a new organizational security policy for enst-bretagne. A new root organization identifier is created and the licence 1 (see figure 2) is inserted in the policy base. This licence specifies that robert can create any licences in enst_bretagne. This is a very powerful statement which actually allows robert to do anything in the organization. Thus, robert can create licences, with authority equals to enst_bretagne that will allow him to create new organizations, views, subjects, roles, role assignments, etc.

As a first example, let us assume that robert wants to create a role admin which is responsible for administering the non administrative roles of the sub-organization campus_rennes of the enst_bretagne. For this purpose, robert must first create the following view definition *with check option*:

use(Org,L,non_admin_licence) :- use(Org,L,licence), grantee(L,R), not (role_inheritance(Org,R,admin)).

The view non_admin_licence only contains licences whose grantee is a role that does not inherit from role admin. robert has then to create the licence 2 of figure 2. Finally, robert can create a subject called marc and assign him to role admin by creating the role assignment 1 of figure 3.

As a second example, robert wants to create another role student_admin for administering the campus_rennes students during working hours. For this purpose, robert first creates the following view definition *with check option*:

use(Org,L,student_licence) :- use(Org,L,licence), grantee(L,R), role_inheritance(Org,R,student).

Thus, the view student_licence only contains licences whose grantee is a role that inherits from role student (for instance role student itself or role graduate_student). To assign to the role student_admin the permission to manage the students of campus_rennes, robert must then create the licence 3 of figure 2. Finally, robert can create a subject called joelle and assign her to role student_admin by creating the role assignment 2 of figure 3.

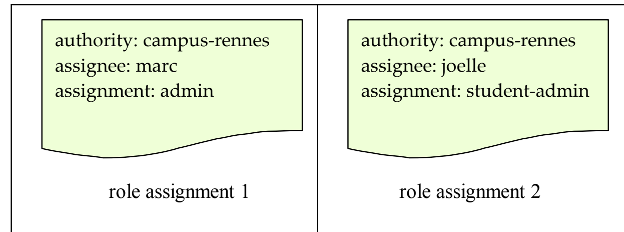

Figure 3: Role assignment examples

4.2 Delegation examples

Previous works on delegation [4, 25] showed that it is a complex problem to solve. It is generally modelled separately from other administration requirements. In this section, we show how our approach makes it possible to manage all administration requirements, including delegation requirements, in a unique model. As an example, let us assume that there is another subject, called valerie, also assigned to role student_admin, but in campus_brest. Let us also assume that joelle wants to delegate to valerie the permission to manage student assignments in campus_rennes in the context during-joelle-vacation. To model this delegation, we have first to create a view student_assignment *with check option*:

use(Org,RA,student_assignment) :-
  use(Org,RA,role_assignment), assignment(RA,Role),
  role_inheritance(Org,Role,student).

This view only contains assignments to a role that inherits from role student.

We need a second view called rennes_admin_student_assignment. This view contains licences that give permission to delegate to subject assigned to role student_admin in campus_brest the permission to manage view student_assignment of campus_rennes. This view is defined *with check option* as follows:

use(Org,L,rennes_admin_student_assignment) :-
  use(Org,L,licence), authority(L,campus_rennes),
  grantee(L,S), empower(campus_brest,S,student_admin),
  privilege(L,manage), target(L,student_assignment).

robert can then create the licence 4 of figure 2 that will allow joelle to create licences belonging to view rennes_admin_student_assignment (and thus to delegate the permission to manage student assignments to a subject who is empowered in role student_admin in campus_brest). Using this licence, joelle can delegate to valerie the permission to manage student assignments in campus_rennes in context during-joelle-vacation by creating the licence 5 of figure 2.

As suggested in [4], many other cases of delegation are possible that, due to space limitation, we cannot further develop in this paper. However, we show in [5] that our administration model actually provides a complete framework for delegation.

5 Conclusion

In this paper, we use a specific object called *licence*, a concept largely used in digital rights management, to deal with the hard problem of security policy administration. When someone obtains an administrative licence, it obtains a permission to create new *licences* that is to define new permissions. These licences can in turn be administrative licences. This technique can be used to completely decentralize the security policy administration.

Moreover, licences are multi-granular: they can be assigned either to an individual subject (an administrator in the case of an administrative licence) or to a role, to do either an action or activity (may be administrative ones), on either an object or view (that can be administrative). Permission derived from a given *licence* is constrained by five attributes (authority, grantee, privilege, target and context) (section 3.3). These constraints define the permission's scope. In this way, we avoid negative side effects of decentralizing security policy administration by

finely managing permissions obtained by delegation and controlling inheritance of permissions. ARBAC fails to manage properly these aspects. The reason is: the range of the authority in ARBAC is based on the role hierarchy which, for instance, may require complex integrity constraints to prevent some leakage of administrative rights, in particular when this role hierarchy is modified. Our model is more expressive (fine grained licences, global licences and nested contexts) than X-GTRBAC (see sections 2.2, 2.3, 2.4).

Multi-granular licences jointly with the *check option* make it possible to nicely modelled delegation (section 4.2). Work is in progress about other kinds of delegation, in particular, problem of cascading or restricting the revocation of delegation. This applies to a security administrator SA who attempts to revoke a licence to some subject SS whereas SS has delegated in turn this licence.

Our model is an extension of the OrBAC model. Concepts of *licence* and *confinement* are fully integrated in this model. OrBAC is self-administrated. This avoids to distinguish the definition of regular permissions from administrative permissions and allows some subjects or roles to cumulate both of them.

The model presented in this paper is currently implemented in MotOrBAC [8]. MotOrBAC can be used to specify various kinds of decentralized administration policies. The policy designer and other policy administrators are authenticated by the prototype so that they can only create or/and manage entities permitted by the administration policy. MotOrBAC implements high-level strategies to manage conflits and rules redundancies as defined in [7]. All security policies in MotOrBAC are saved in XML to comply with the format of information exchanged via Internet (see [12] for a presentation of the XML schema that implements the OrBAC model). The MotOrBAC prototype has been tested to specify administration security policies of an health care system (*Sillage* project) and e-voting project (*Expérience Démocratique* Project). Finally, the OrBAC model and administration extensions discussed in this paper are good bases to define an interoperability infrastructure. This point is further developed in [9] and the corresponding XaCML profile is presented in [2].

References

[1] MPEG REL. *ISO/IEC FDIS 21000-5: Information Technology Multimedia Framework Part 5: Rights Expression Language*, August 2003.

[2] D. AbiHaidar, N. Cuppens-Boulahia, F. Cuppens, and H. Debar. An extended RBAC profile of XACML. In *ACM Workshop on Secure Web Services (SWS)*, Alexandria, VA, November 2006.

[3] A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *Policy'03*, Como, Italy, June 2003.

[4] E. Barka and R. Sandhu. Framework for Role-Based Delegation Models. In *16th Annual Computer Security Applications Conference (ACSAC '00)*, Washington, DC, USA, 2000.

[5] M. Benghorbel, F. Cuppens, N. Cuppens-Boulahia, and A. Bouhoula. Managing Delegation in Access Control Models. In *15th International Conference on Advanced Computing and Communication (ADCOM)*, Guwahati, India, December 2007.

[6] R. Bhatti, J. B. D. Joshi, E. Bertino, and A. Ghafoor. X-GTRBAC Admin: A Decentralized Administration Model for Enterprise Wide Access Control. 8(4):224–274, November 2005.

[7] F. Cuppens, N. Cuppens-Boulahia, and M. Benghorbel. High-level conflict management strategies in advanced access control models. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 186:3–26, July 2007.

[8] F. Cuppens, N. Cuppens-Boulahia, and C. Coma. MotOrBAC : un outil d'administration et de simulation de politiques de sécurité. In *First Joint Conference on Security in Network Architectures (SAR) and Security of Information Systems (SSI)*, Seignosse, France, June 2006. In French.

[9] F. Cuppens, N. Cuppens-Boulahia, and C. Coma. O2O: Virtual Private Organizations to Manage Security Policy Interoperability. In *2nd International Conference on Information Systems Security (ICISS)*, Kolkata, India, December 2006.

[10] F. Cuppens, N. Cuppens-Boulahia, and A. Miège. Inheritance hierarchies in the Or-BAC Model and Application in a network environnement. In *2nd Foundation of Computer Security Workshop*, Turku, Finlande, July 2004.

[11] F. Cuppens, N. Cuppens-Boulahia, and T. Sans. Nomad: A Security Model with Non Atomic Actions and Deadlines. In *18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 186–196, Aix-en-Provence, France, 2005.

[12] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A formal approach to specify and deploy a network security policy. In *Second Workshop on Formal Aspects in Security and Trust (FAST)*, Toulouse, France, 26-27 August, 2004.

[13] F. Cuppens and A. Miège. Administration Model for Or-BAC. *Computer Systems Science and Engineering (CSSE'04)*, 19(3), May, 2004.

[14] F. Cuppens and A. Miège. Modelling Contexts in the Or-BAC Model. In *19th Annual Computer Security Applications Conference (ACSAC '03)*, 2003.

[15] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. 4(3):224–274, August 2001.

[16] S. Jajodia, S. Samarati, and V. S. Subrahmanian. A logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.

[17] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. Generalized Temporal Role-Based Access Control Model. 17(1):4–23, January 2005.

[18] A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, Como, Italie, June 2003.

[19] H. M. Levy. *Capability-Based Computer Systems*. Digital Press, 1964.

[20] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1997.

[21] R. Sandhu and Q. Munawer. The ARBAC99 Model for Administration of Roles. In *15th Annual Computer Security Applications Conference (ACSAC '99)*, page 229. IEEE Computer Society, 1999.

[22] R. Sandhu and S. Oh. A Model for Role Administration Using Organization Structure Roles. In *7th ACM symposium on Access control models and technologies (SACMAT '02)*, Monterey, California, 2002.

[23] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996.

[24] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.

[25] X. Zhang, S. Oh, and R. Sandhu. PBDM: A flexible delegation model in RBAC. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, Como, Italie, June 2003.