Temporal Hierarchies and Inheritance Semantics for GTRBAC

James B D Joshi
School of Electrical and Computer Engineering,
Purdue University,
West Lafayette, IN, USA
joshij@ecn.purdue.edu

Elisa Bertino
Dipartimento di Scienze dell'Informazione,
Universita' di Milano,
Milano, Italy
bertino@dsi.unimi.it

Arif Ghafoor
School of Electrical and Computer Engineering,
Purdue University,
West Lafayette, IN, USA
ghafoor@ecn.purdue.edu

ABSTRACT

A Generalized Temporal Role Based Access Control (GTRBAC) model that allows specification of a comprehensive set of temporal constraint for access control has recently been proposed. The model constructs allow one to specify various temporal constraints on role, user-role assignments and role-permission assignments. However, Temporal constraints on role enablings and role activations can have various implications on a role hierarchy. In this paper, we present an analysis of the effects of GTRBAC temporal constraints on a role hierarchy and introduce various kinds of temporal hierarchies. In particular, we show that there are certain distinctions that need to be made in permission inheritance and role activation semantics in order to capture all the effects of GTRBAC constraints such as role enablings and role activations on a role hierarchy.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access control; H.2.7 [**Database Administration**]: Security, integrity, and protection

General Terms

Security, Theory

Keywords

role based access control, security, role hierarchy, temporal constraints

1. INTRODUCTION

Role based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access control (DAC and MAC) models [3, 6, 8, 9, 12], which have some inherent limitations [6]. Several beneficial features such as policy neutrality, support for least privilege and efficient access control management are associated with RBAC models [6, 12]. Such features make RBAC better suited for handling access control requirements of diverse organizations. Furthermore, the concept of role is associated with the notion of functional roles in an organization, and hence RBAC models provide intuitive support for expressing organizational access control policies [12]. RBAC models have also been found suitable for addressing security issues in the Internet environment [6, 10], and show promise for newer heterogeneous multi-domain environments that raise serious concerns related to access control across domain boundaries [6].

One of the important aspects of access control is that of time constraining accesses to limit resource use. Such constraints are essential for controlling time-sensitive activities that may be present in various applications, such as workflow management systems (WFMSs), where various workflow tasks, each having some timing constraints, need to be executed in some order. Use of RBAC has been found very suitable for such workflow applications [2]. To address general time-based access control needs, Bertino *et. al.* propose a Temporal RBAC model (TRBAC) [1], which has been recently generalized by Joshi *et. al.* [5]. The Generalized-TRBAC (GTRBAC) model [5] incorporates a set of language constructs for the specification of various temporal constraints on roles, including constraints on their activations as well as on their enabling times, user-role and role-permission assignments. In particular, GTRBAC makes a clear distinction between role *enabling* and role *activation*. A role is *enabled* if a user can acquire the permissions assigned to it. An *enabled* role is said to be *activated* when a user acquires the permissions assigned to the role in a session. Related to the notions of an *enabled* or an *activated* role is that of Jaeger *et. al.'s assigned* and *activated* authorizations [4]. An open issue in the GTRBAC model, as well as in the TRBAC model [1], is the interplay between temporal constraints and role hierarchy.

Many researchers have highlighted the importance and use of role hierarchies in RBAC models [3, 7, 11]. A properly designed role hierarchy allows efficient specification and management of access control structures of a system. When two roles are hierarchically related, one is called the senior and the other the junior. The senior role inherits all the permissions assigned to the junior roles. The inheritance of permissions assigned to junior roles by a senior role significantly reduces assignment overhead, as the permissions need only be explicitly assigned to the junior roles.

Even though the notion of role hierarchy has been widely investigated, to our knowledge, no earlier work has addressed the implication of the presence of temporal constraints on role hierarchies, which is the focus of our work. In particular, in this paper, we present a detailed analysis of role hierarchy in the presence of various temporal constraints with respect to the GTRBAC model and show that there are various distinctions that need to be made about the inheritance semantics of a role hierarchy.

It is important to point out that Sandhu [11] and Moffet *et. al.* [7] have already recognized the limitations of the pure inheritance semantics proposed in the RBAC96 family of models [12]. Sandhu [11] has proposed the ER-RBAC96 model that incorporates a distinction between two types of role hierarchy: *usage* hierarchy that applies *permission-inheritance* semantics and *activation* hierarchy that uses *activation - inheritance* semantics. In a *usage* hierarchy, the activation of a *senior* role allows a user to acquire all the permissions of all of its junior roles but no users assigned only to the senior role is allowed to activate the junior roles. An *activation* hierarchy extends "*permission inheritance hierarchy to roles that are stipulated to have dynamic separation of duty* (SoD)" [11]. Our analysis further strengthens his arguments and shows that, in the presence of timing constraints on various entities, the separation of the *permission-inheritance* and the *activation-inheritance* semantics provides a basis for capturing various inheritance semantics of a hierarchy. We show that these hierarchies can be *unrestricted, weakly restricted* or *strongly restricted* depending upon the enabling times of the hierarchically related roles. In another important work related to role hierarchies, Moffet *et al.* [7] have identified the need for three types of hierarchies: *isa* hierarchy, *activity* hierarchy and *supervision* hierarchies, in order to address the needs of control principles in an organization, which include *separation of duty, decentralization* and *supervision and review* [7]. They show that the combined *permission* and *activation* inheritance within a hierarchy can limit a hierarchy from achieving organizational control needs. Clearly, our temporal hierarchies as well as Sandhu's hierarchies provide a basis for limiting such complete inheritance in a hierarchy, making it possible to support separation of duty and restricted inheritance in a hierarchy. Furthermore, Moffett *et. al.* [7] point out that the commercial organizations' demand for a *dynamic* access control model that can support dynamic authorization state as well as dynamic propagation of access rights has largely being neglected. The GTRBAC models's temporal framework and the trigger mechanism along with the temporal hierarchies that we define in this paper provide a strong basis for such dynamic features in an access control model.

The paper is organized as follows. In section two, we briefly present the constraint model of GTRBAC. In section three, we present the analysis of inheritance semantics in a role hierarchy of the GTRBAC model. We then discuss related work in section four and present some conclusions and future work in section five.

2. THE GTRBAC MODEL

The GTRBAC model [5] is an extension of the TRBAC model [1]. The GTRBAC model introduces the separate notion of role enabling and role activation and provides constraints and event expressions associated with both. An enabled role indicates that a user can activate it, whereas an activated role indicates that at least one subject has activated a role in a session. The temporal constraints in GTRBAC allow the specification of the following constraints:

1. *Temporal constraints on role enabling/disabling*: These constraints allow one to specify the time intervals during which a role is enabled or diabled. When a role is enabled, the permissions assigned to it can be acquired by a user by simply activating the role. It is also possible to specify a role duration. When such a duration is specified, the enabling/disabling event for a role is initiated by a constraint-enabling expression that may be separately specified at run-time by an administrator, or by a trigger.

2. *Temporal constraints on user-role and role-permission assignments*: These are constructs to express either a specific interval or a duration in which a user or a permission is assigned to a role.

3. *Activation constraints*: These allow one to specify how a user should be restricted in activating a role. These include, for example, specifying what is the total duration a user is allowed to activate a role, or how many users can be allowed to activate a particular role.

4. *Run-time events*: A set of run-time events allows an administrator to dynamically initiate GTRBAC events, or enable duration or activation constraints. Another set of run-time events allows users to make activation requests to the system.

5. *Constraint enabling expressions*: GTRBAC includes events that enable or disable duration constraints and role activation constraints.

6. *Triggers*: Triggers allow one to express dependency among GTRBAC events, capture past events, and define future events based on them.

The GTRBAC model extends the safety notion of the TRBAC model to show that there exists an execution model for it. For the periodicity constraints, the periodic expressions are written as (I, P), where I is an interval and P is a set of infinte number of intervals. (I, P) represents the set of all the intervals of P that is contained in I. For example, (I, P) = ([1/1/2002, 12/31/2002], *Mondays*) considers all the *Mondays* of the year 2002. D is used to express the duration specified for a duration constraint. Constraints are expressed by a generic form (X, E), where X is a periodic expression (I, P) or an expression indicating duration D, and E is an event expression such as `enable r` (event that enables role r). For more details, we refer the readers to [5]. We illustrate with an example the GTRBAC specification of an access control policy. Table 1 contains the GTRBAC specification of a hospital's access policy. Groupings labeled 1, 2, 3 and a, b, c are used simply to ease discussion.

In 1a, the enabling times of `DayDoctor` and `NightDoctor` roles are specified as a periodicity constraint. For simplicity we use *DayTime* and *NightTime* instead of their (I, P) forms. In 1b, different users are assigned to the two doctor roles. *Adams* can assume `DayDoctor` role on *Mondays,*

Table 1: Example GTRBAC access policy for a medical information system

1	a.	$(DayTime$, `enable DayDoctor`), $(NightTime$, `enable NightDoctor`)
	b.	((M, W, F), `assign`$_u$ *Adams* `to DayDoctor`), ((T, Th, S, Su), `assign`$_u$ *Bill* `to DayDoctor`), ((M, W, F), `assign`$_u$ *Alice* `to NightDoctor`), ((T, Th, S, Su), `assign`$_u$ *Ben* `to DayDoctor`),
	c.	((10am, 3pm), `assign`$_u$ *Carol* `to DayDoctor`)
2	a.	(`assign`$_u$ *Ami* `to NurseInTraining`), (`assign`$_u$ *Elizabeth* `to DayNurse`)
	b.	$c1 = (6\ hours, 2\ hours$, `enable NurseInTraining`)
3	a.	(`enable DayNurse` → `enable` $c1$)
	b.	(`activate DayNurse for` *Elizabeth* → `enable NurseInTraining after` 10 *min*)
	c.	(`enable NightDoctor` → `enable NightNurse after` 10 *min*), (`disable NightDoctor` → `disable NightNurse after` 10 *min*)
	d.	(`enable DayDoctor` → `enable DayNurse after` 10 *min*), (`disable DayDoctor` → `disable DayNurse after` 10 *min*)

Wednesdays and *Fridays*, whereas *Bill* can assume it on *Tuesdays, Thursdays, Saturdays* and *Sundays*. Similarly, *Alice* and *Ben* are assigned to the `NightDoctor` role on the different days of the week. Furthermore, in 1c, the assignment indicates that *Carol* can assume the `DayDoctor` role everyday between 10am and 3pm. In 2a, *Ami* and *Elizabeth* are assigned to roles `NurseInTraining` and `DayNurse` respectively, without any periodicity or duration constraints; that is, the assignment is valid at all times. 2b specifies a duration constraint of 2 *hours* on the enabling time of the `NurseInTraining` role, but this constraint is valid for only 6 *hours* after the constraint $c1$ is enabled. Because of this, *Ami* will be able to activate the `NurseInTraining` role at the most for two hours whenever the role is enabled.

The constraints in 3 are triggers. Trigger 3a indicates that constraint c1 is enabled once the `DayNurse` is enabled, which means now the `NurseInTraining` role can be enabled within the next 6 *hours*. Trigger 3b indicates that 10 *min* after *Elizabeth* activates the `DayNurse` role, the `NurseInTraining` role is enabled for a period of 2 *hours*. This shows that a nurse in training will have access to the system only if *Elizabeth* is present in the system, that is, she is acting as a training supervisor. It is possible that *Elizabeth* activates the `DayNurse` role a number of times within 6 hours after the `DayNurse` role is enabled, and hence the `NurseInTraining` role will be enabled as many times if these activations (by *Elizabeth*) are more than 2 hours apart. This will allow *Ami* to activate the `NurseInTraining` role a number of times. To prevent this, an activation constraint can be added. The remaining triggers in 3 show that the `DayNurse` and `NightNurse` roles are enabled (disabled) 10 *min* after the `DayDoctor` and `NightDoctor` roles are enabled (disabled).

3. ROLE HIERARCHIES AND TEMPORAL CONSTRAINTS

Sandhu [11] distinguishes a role hierarchy into two types: *usage* hierarchy and *activation* hierarchy. By defining an *activation* hierarchy as a superset of a *usage* hierarchy, Sandhu establishes an *activation* hierarchy essentially as an extension of the *usage* hierarchy. Furthermore, he shows that there are situations where the distinction between the two is very crucial. In particular, the distinction allows capturing *dynamic* SoD constraints that may exist between hierarchically related roles.

In the remainder of this section, we formally define the basic types of a temporal hierarchy and then analyze the effects of various temporal constraints on them. We show that the different types of hierarchy need to be further divided into subtypes in order to capture the complete inheritance semantics introduced due to different temporal properties associated with the roles of the hierarchy

3.1 Temporal Role Hierarchy

Here, we take a slightly different approach than in [11]. We explicitly define a hierarchy that allows only permissions to be inherited as *inheritance-only hierarchy* or *I*-hierarchy (same as the *permission-usage* hierarchy in [11]) and the one that allows only the activation semantics as *activation-only hierarchy* or *A*-hierarchy. We further refer to a hierarchy combining both the *inheritance* and *activation* semantics as *general hierarchy* or *Inheritance-Activation hierarchy*(*IA*-hierarchy for short). Finally, we extend the notion of hierarchical relations with respect to a time instant t in order to capture the fact that such semantics are time dependent. Table 2 reports the various predicate notations we use in the formal definitions presented in this paper.

Predicates $enabled(r, t)$, $u\_assigned(u, r, t)$ and $p\_assigned$ (p, r, t) refer to the status of roles, and user-role and role-permission assignments at time t. Predicate $can\_activate(u, r, t)$ implies that user u can activate role r at time t. It allows us to capture the fact that a user u may be able to activate role r without being explicitly assigned to it, as it is possible in a hierarchy that incorporates the *activation-inheritance* semantics. In other words, "*u can activate r*" implies that user u is implicitly or explicitly assigned to role r. It also does not rule out the possibility that some activation or SoD constraints may prevent the actual activation of r by u at time t. Predicate $can\_acquire(u, p, t)$ implies that *u can acquire permission p* at time t, whereas, the predicate $can\_be\_acquired(p, r, t)$ implies that permission *p can be acquired through role r* at time t. It is important to note that predicates $can\_activate(u, r, t)$, $can\_acquire(u, p, t)$ and $can\_be\_acquired(p, r, t)$ do not assume anything about the state of a role. That is, they do not say in which state role r is at time t. For example, if $can\_activate(u, r, t)$ and $enabled(r, t)$ hold then a user u's request to activate r at time t is granted provided there are no other activation or SoD constraints prohibiting it. However, if $can\_activate(u, r, t)$ holds but not $enabled(r, t)$, then u's request to activate r at time t is denied. Thus, predicates $can\_activate(u, r, t)$, $can\_acquire(u, p, t)$ and $can\_be\_acquired(p, r, t)$ indicate possibility rather than what actual occurs.

Predicates $active(u, r, s, t)$ and $acquires(u, p, s, t)$ refer to what actually occurs at time instant t. $active(u, r, s, t)$ in-

Table 2: Various status predicates

Predicate	Meaning
$enabled(r,t)$	*Role r is enabled at time t*
$u\_assigned(u,r,t)$	*User u is assigned to role r at time t*
$p\_assigned(p,r,t)$	*Permission p is assigned to role r at time t*
$can\_activate(u,r,t)$	*User u can activate role r at time t*
$can\_acquire(u,p,t)$	*User u can acquire permission p at time t*
$can\_be\_acquired(p,r,t)$	*Permission p can be acquired through role r at time t*
$active(u,r,s,t)$	*Role r is active in a user u's session s at time t*
$acquires(u,p,s,t)$	*User u acquires permission p in session s at time t*

dicates that role r is active in user u's session s at time t, whereas, $acquires(u,p,s,t)$ implies that u acquires permission p at time t in session s.

Now we introduce the following axioms to capture the key relationships among various predicates in Table 2 so that we can identify precisely the permission-acquisition and role activations that are possible or that actually occur in an RBAC system.

AXIOMS. *For all* $r \in \texttt{Roles}$, $p \in \texttt{Permissions}$, $u \in \texttt{Users}$, $s \in \texttt{Sessions}$ *and* $t \geq 0$, *the following implications hold*:

1. $p\_assigned(p,r,t) \rightarrow can\_be\_acquired(p,r,t)$
2. $u\_assigned(u,r,t) \rightarrow can\_activate(u,r,t)$
3. $can\_activate(u,r,t) \wedge can\_be\_acquired(p,r,t)$ $\rightarrow can\_acquire(u,p,t)$
4. $active(u,r,s,t) \wedge can\_be\_acquired(p,r,t)$ $\rightarrow aquires(u,p,s,t)$

Axiom (1) states that if a permission is assigned to a role, then it *can be acquired* through that role. Axiom (2) states that all users assigned to a role *can activate* that role. Axiom (3) states that if a user u *can activate* a role r, then all the permissions that *can be acquired* through r *can be acquired* by u. Thus, for the simple case where user u and permission p are assigned to r, the axioms indicate that u *can acquire* p. Similarly, axiom (4) states that if there is a user session in which a user u has activated a role r then u *acquires* all the permissions that *can be acquired* through role r.

We note that axioms (1) and (2) indicate that permission acquisition and role activation semantics is governed by explicit user-role and role permission assignments. Semantically, the use of a role hierarchy is to extend the possibility of permission acquisition and role-activation beyond the explicit assignments, as we shall show next. Below, we define the formal semantics of the time-dependent role hierarchies. The following definitions do not consider the enabling times of the hierarchically related roles, and hence are termed *unrestricted* hierarchies. The restricted forms will be introduced in later sections.

Definition 1. (*Unrestricted inheritance-only hierarchy or I-hierarchy*) Let x and y be roles such that $(x \geq_t y)$, that is, x has an *unrestricted inheritance-only* relation over y. Then the following condition ($c1$) holds:

$$\forall p, (x \geq_t y) \wedge can\_be\_acquired(p,y,t) \rightarrow can\_be\_acquired(p,x,t)$$

x is said to be a senior role of y, and conversely y is said to be a junior role of x, with respect to the *unrestricted I*-hierarchy.

The condition characterizing *inheritance-only* relation provides a new way of acquiring a permission through a role by using its relation with other roles. Its semantics indicates that a permission *can be acquired* through a role by direct inheritance of all the permissions of junior roles. Thus if $(x \geq_t y)$, then the permissions that can be acquired through x include all the permissions assigned to x (by axiom (1)) and all permissions that *can be acquired* through role y (by $c1$), which in turn include all the permissions assigned to y as well as all the permissions that *can be acquired* through y's juniors (by axiom (1) and condition $c1$). This shows that the I-hierarchy is transitive. Note that the axioms and condition $c1$ do not allow u to activate y. Hence, the hierarchical relation $(x \geq_t y)$ is restricted to the *permission-inheritance* semantics only.

Definition 2. (*Unrestricted activation-only hierarchy or A-hierarchy*) Let x and y be roles such that $(x \succeq_t y)$, that is, x has an *unrestricted activation-only* relation over y. Then the following condition ($c2$) holds:

$$\forall u, (x \succeq_t y) \wedge can\_activate(u,x,t) \rightarrow can\_activate(u,y,t)$$

x is said to be a senior role of y, and conversely y is said to be a junior role of x, with respect to the *unrestricted A*-hierarchy.

Here, the *activation-only* semantics introduces a new "*can activate*" semantics between a user and a role. Axiom (2) states that a user able to activate a role through explicit assignment, whereas the A-relation allows that through relations between roles, without a need for explicit user-role assignment. Condition ($c2$) states that if user u *can activate* role x, and x has A-relation over y, then s/he *can activate* role y too, even if u is not explicitly assigned to y. However, note that an explicit assignment of u to y is possible but will be redundant here. The set of axioms and condition $c2$ together allow a user u assigned to role x to activate all of y's juniors. However, as condition $c1$ does not apply to an A-hierarchy, if $(x \succeq_t y)$, then u cannot acquire y's permissions by just activating x. Note that the $can\_activate(u,x,t)$ predicate makes A-hierarchy transitive the same way the predicate $can\_be\_acquired(p,y,t)$ makes an I-hierarchy so.

Definition 3. (*Unrestricted general hierarchy or IA-hierarchy*) Let x and y be roles such that $(x \gg_t y)$, that is, x has an *unrestricted general inheritance* relation over y. Then the following holds:

$$(x \gg_t y) \rightarrow (x \geq_t y) \wedge (x \succeq_t y)$$

x is said to be a senior role of y, and conversely y is said to be a junior role of x, with respect to the *unrestricted* IA-hierarchy.

The IA-hierarchy is the most common form of hierarchy and contains both *permission-inheritance* and *activation-inheritance* aspects of a hierarchy. Hence, a user can acquire permissions of roles that are junior to which s/he is assigned without activating them. At the same time, s/he may activate the junior roles even though s/he is not explicitly assigned to them. Note that the definitions do not account for the enabling times of the roles that are hierarchically related.

On a given set of roles, there may be various inheritance relations. Therefore, we require that the following *consistency* property be satisfied in a role hierarchy.

Consistency Property : Let $<f> \in \{\geq_t, \succeq_t, \gg_t\}$ and $<f'> \in \{\geq_t, \succeq_t, \gg_t\}/\{<f>\}$. Let x and y be roles such that $(y <f> x)$; then the condition $\neg\ (y <f'> x)$ must hold.

The main purpose of a hierarchical relation is the acquisition of the permissions of junior roles by a senior role by using any of the three hierarchy types. The *consistency* property ensures that a senior-junior relation between two roles in one type of hierarchy is not reversed in another type of hierarchy. Due to space limitation, we do not address here other issues concerning how various hierarchies can co-exist within the same set of roles.

Examples of the three hierarchies are illustrated in Figure 1, where the `Software Engineer` role is senior to the `Progammer` role. In Figure 1(a) and 1(b), the combination of roles that a user u, assigned only to `Software Engineer` role, can activate is $\{$(`Software Engineer`),(`Programmer`), (`Software Engineer, Programmer`)$\}$. However, the permissions associated with the same combination in the two cases are not exactly the same. For example, if u activates the `Software Engineer` role, the permissions acquired by him in 1(a) is maximal, that is, both the roles' permissions are acquired, whereas it is only the permissions assigned to the `Software Engineer` role in the case of 1(b). Furthermore, the activation of the combination (`Programmer, Software Engineer`) is redundant in an IA-hierarchy in terms of which permissions are acquired, while it is significant in 1(b).

Under the role hierarchy reported in Figure 1(c), the user can activate only the `Software Engineer` role (unless of course, the user is also explicitly assigned to the `Programmer`). However, he acquires maximal permissions, that is, permissions assigned to both the roles.

Table 3 shows various features, indicated in the first row, of those types of hierarchy. The second row shows these features for the I-hierarchy over the set of roles $x_1, x_2, \ldots, x_n$ for which the inheritance relation is $x_1 \geq_t x_2 \geq_t \ldots \geq_t x_{n-1} \geq_t x_n$ (i.e., x_1 is the senior-most and x_n is the junior-most). In the table, u refers to a user assigned only to role x_1. Similar features over the set of roles characterize the A-hierarchy and IA-hierarchy; these relatess are reported in the third and the fourth rows of Table 3. The second column shows the combinations of roles that can be activated by u. In the I-hierarchy, only one role can be activated. In A and IA-hierarchies, any subset of the roles can be activated because of the role-activation semantics; however, in IA-hierarchy the activation is redundant in terms of permission acquisition. For example, consider a subset of roles that contain role x_1; activation of any other roles in this set is unnecessary as it does not add new set of permissions acquired by u.

The third and forth column show that an I-hierarchy allows the acquisition of only the maximal permission set P_{max}. Here, by considering P_i as the permission set associated with role x_i, we get $P_{max} = \cup_{i=1}^{n} P_i$ and $P_{min} = P_n$. The two columns also show that in A and IA-hierarchies, u can acquire both the maximum or minimum number of permissions. However, under an A-hierarchy, u will need to activate all the roles to acquire P_{max}, whereas under an IA-hierarchy, u will need to activate only the senior-most role.

The last column shows the number of unique combinations of these permission sets that u can acquire. Since u can activate only the senior-most role in an I-hierarchy, u can acquire only one set of combination of these permission sets, which is P_{max}. In an A-hierarchy, any subset of roles can be activated to acquire the unique combinations of the permission sets that are associated with the activated roles. Hence, it is $2^{|X|} - 1$, as there are that many non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$. We note that, in an IA-hierarchy, by activating role x_i, the user essentially acquires permission sets associated with all the roles $x_i, x_{i+1}, \ldots, x_n$. Hence, only n unique permission sets can be acquired. These values are significant from the perspective of the principle of the *least privilege*, that is, an I-hierarchy has no support for the least privilege acquisition, whereas an IA-hierarchy supports the least privilege at n levels, that is, n different combinations of permission sets are allowed. The A-hierarchy supports all combinations and hence completely supports the principle of *least privilege.*

We also note that if the constraints of the definitions are such that they hold true for all the time instants, then those hierarchies reduce to non-temporal RBAC hierarchies along the ones discussed in [11].

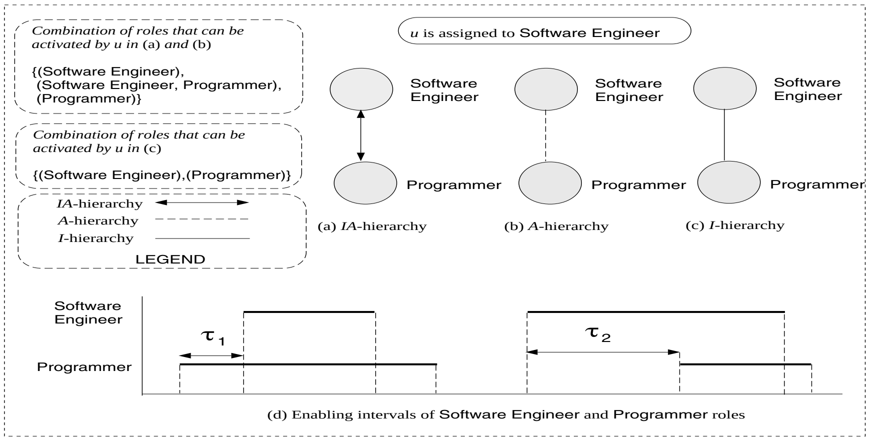
3.2 Role Enabling Constraints and Hierarchies

A hierarchy in the presence of various temporal constraints becomes dynamic as permissions and users can be assigned or de-assigned to any junior roles at times when a senior role is enabled. Furthermore, there are activation constraints that need to be accounted for when either of the hierarchy types is considered. Here, we consider the effect of the presence of temporal assignment constraints on the hierarchies.

3.2.1 Inheritance-only hierarchy (I-hierarchy)

As we can see, in an I-hierarchy, the permissions of a junior role are acquired by the senior role. However, in the presence of temporal constraints, we need to be able to capture various dynamic aspects of the hierarchy.

Let us revisit the I-hierarchy of Figure 1(c). Figure 1(d) shows two possible intervals associated with the enabling times of the two roles. In Figure 1(d)-(i), we see that the enabling interval of `Software Engineer` role is a subset of that of the `Programmer` role. In this case, the I-hierarchy has the semantics similar to the non-temporal RBAC, that is, whenever u activates the `Software Engineer` role s/he also acquires the permissions of the `Programmer` role, because at that time the `Programmer` role is also enabled. Thus, in interval τ_1, u cannot acquire any permissions of



Combination of roles that can be activated by u in (a) and (b)
{(Software Engineer), (Software Engineer, Programmer), (Programmer)}
Combination of roles that can be activated by u in (c)
{(Software Engineer),(Programmer)}
IA-hierarchy
A-hierarchy
I-hierarchy
LEGEND
u is assigned to Software Engineer
Software Engineer
Programmer
Software Engineer
Programmer
Software Engineer
Programmer
(a) IA-hierarchy
(b) A-hierarchy
(c) I-hierarchy
Software Engineer
Programmer
τ1
τ2
(d) Enabling intervals of Software Engineer and Programmer roles

Figure 1: An example hierarchy

Table 3: Hierarchies and associated features with u assigned to x_1

x_1 <f> x_2 <f> $\ldots x_{n-1}$ <f> x_n	Role combination that can be activated in a session by u	Maximal Permission set that can be acquired by u	Minimal Permission set that can be acquired by u	Number of combinations of $P_1, P_2, \ldots, P_n$ acquired by u
I-hierarchy	$\{x_1\}$	P_{max} (By activating $\{x_1\}$)	P_{max} (By activating $\{x_1\}$)	1
A-hierarchy	$X \subseteq \{x_1, x_2, \ldots, x_n\}$	P_{max} (By activating $\{x_1, x_2, \ldots, x_n\}$)	P_{min} (By activating $\{x_n\}$)	$2^{\|X\|} - 1$
IA-hierarchy	$X \subseteq \{x_1, x_2, \ldots, x_n\}$	P_{max} (By activating $\{x_1\}$)	P_{min} (By activating $\{x_n\}$)	n

the `Programmer` role even if it is enabled, as the `Software Engineer` role is disabled at that time. It is also possible that there is a temporal interval in which `Software Engineer` role is enabled but the `Programmer` role is not, as indicated by interval τ_2 in Figure 1(d)-(ii). In such a case, we can see that the following two approaches can be used to capture the inheritance semantics :

1. *Weakly restricted approach* (I_w): The permissions of the `Programmer` role are inherited by the `Software Engineer` role in interval τ_2,
2. *Strongly restricted approach* (I_s): The permissions of the `Programmer` role are not inherited by the `Software Engineer` role in interval τ_2.

Under the *weakly restricted* approach, every permission that can be acquired through a junior role can also be acquired through its senior roles under an I-hierarchy, irrespective of whether the junior role is enabled or not. In the *strongly restricted* approach, each permission that can be acquired through a junior role can also be acquired through its senior roles only in intervals where the junior role is also enabled. Table 4 below summarizes the inheritance semantics of an I-hierarchy in the presence of temporal constraints.

3.2.2 Activation-only hierarchy (A-hierarchy)

We see that when we have an A-hierarchy, it is natural to just use the second approach given above, that is, there is no *activation*-inheritance allowed in interval τ_2. This is because of an explicit need for activating a junior role by a user assigned to its senior role in order to acquire the junior role's permissions, and in τ_2, the junior role cannot be activated. If we try to also enforce the first possibility mentioned above then it will conflict with the semantics of an enabled role, as only enabled roles can be activated.

However, as *activation* hierarchy needs a user assigned to the senior role to activate a junior role in order to acquire the junior role's permissions, the issue of propagation of temporal user-role assignment down the A-hierarchy needs to be considered. For example, consider the roles `Software Engineer` and `Programmer` forming the A-hierarchy in Figure 1(b). Consider again the same enabled times of the two roles as in Figure 1(d). We need to determine whether the user is to be allowed to activate the junior role at the time when the senior role he is assigned to is not enabled, as indicated by the interval τ_1 in Figure 1(d)-(i). For such a case, we can again delineate the following two approaches:

1. *Weakly restricted approach* (A_w): The user u is allowed to activate `Programmer` role in the A-hierarchy at any time the `Programmer` role is enabled.
2. *Strongly restricted approach* (A_s): The user u is allowed to activate the `Programmer` role only if both the `Software Engineer` and `Programmer` roles are enabled (note that he does not need to activate the `Software Engineer` role).

In both the approaches, when a user tries to activate a role in an *activation* hierarchy, additional checks need to be carried out. The first check is to determine if the user is assigned to any role, up the hierarchy, starting from the role it is attempting to activate. The second check is required

Table 4: Inheritance semantics

Hierarchy ($r_1 <f> r_2$)	τ r_1 *disabled* r_2 *enabled*	τ r_1 *enabled* r_2 *disabled*
I_w	No inheritance in τ	Inheritance in τ (*by activating* r_1)
I_s	No inheritance in τ	No inheritance in τ
A_w	Inheritance in τ (*by activating* r_1)	No inheritance in τ
A_s	No inheritance in τ	No inheritance in τ
IA_w	A-Inheritance in τ (*by activating* r_2)	I-Inheritance in τ (*by activating* r_1)
IA_s	No inheritance in τ	No inheritance in τ

to determine if the senior role that a user is assigned to is also enabled. If the senior role is disabled, we then need to deactivate all activations of junior roles by the user assigned to the senior role. Table 4 summarizes the two *activation* inheritance types.

3.2.3 General inheritance hierarchy (IA-hierarchy)

As general inheritance embodies both the *permission inheritance* and *role-activation* semantics of a role hierarchy, it is simply a combination of the two. In other words, in interval τ_1, the general hierarchy can benefit from the use of role-activation semantics and allow activation of the junior role using the *weakly restricted* semantics. Similarly, in interval τ_2, the *inheritance-only* semantics can be used and inheritance through the senior role using *weakly unrestricted* semantics can be utilized. This is shown in Table 4.

3.2.4 Example of hierarchy subtypes

We illustrate with the examples reported in Figure 2 the practical uses of the various kinds of hierarchies listed in Table 4.

Consider the I_w-hierarchy in Figure 2(a). Here, we see that the SeniorSecurityAdmin role is enabled only in interval (8pm, 11pm). Neither of the junior roles is enabled in the entire interval (8pm, 11pm). But the I_w relation allows a user who activates the SeniorSecurityAdmin role to acquire all the permissions of the junior roles too. This may be desirable if SeniorSecurityAdmin role is designed to perform special security operations for checking and maintenance. In such a case, it is reasonable to think that the user assigned to the SeniorSecurityAdmin role will need all the administrative privileges of the junior roles. The temporal restrictions on SecurityAdmin1 and SecurityAdmin2 restrict the users assigned to them to carry out corresponding system administration activities only in the specified intervals. However, here, the user assigned to SeniorSecurityAdmin cannot assume the role of the junior roles SecurityAdmin1 and SecurityAdmin2. To remove this limitation, we can use IA_w-hierarchy instead.

The hierarchy in Figure 2(b), on the other hand, is of type I_s. The senior role is the PartTimeDoctor role, which has two intervals in which it can be enabled, (3pm, 6pm) and (7am, 10am). If a user activates the PartTimeDoctor role in the first interval, according to the I_s relation, he essentially gets all the privileges of only the DayDoctor role, as the NightDoctor role is disabled at that time. Now, consider the second interval. We see that it overlaps with the

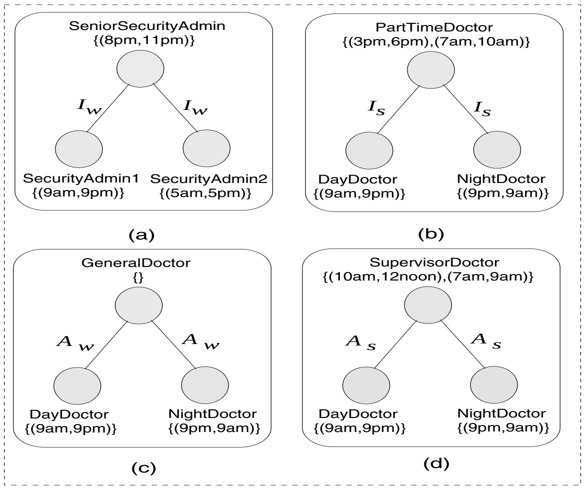


Figure 2: Hierarchy examples

enabling times of the two junior roles. Hence, if the user activates the PartTimeDoctor role in the second interval, he acquires the privileges of only the NightDoctor role in the sub-interval (7am-9am) and that of only the DayDoctor role in the interval (9am, 10am). Thus, we see that the two different semantics of an *inheritance* hierarchy can be used to achieve different needs. Again, a part time doctor cannot work as a DayDoctor or a NightDoctor, although he can acquired the permissions. However, if a user is also to be allowed to use the junior roles, we can use IA_w-hierarchy instead.

Now, let us look at Figure 2(c). Here, we see that there is no interval in which the GeneralDoctor role can be enabled. However, since the activation hierarchy is of type A_w, any user assigned to the GeneralDoctor role can activate either of the junior roles when they are enabled. In effect, any one assigned to the GeneralDoctor role can activate both the DayDoctor and the NightDoctor roles whenever they are enabled.

Figure 2(d) illustrates the use of an activation hierarchy of type A_s. Here, a doctor supervisor can assume the SupervisorDoctor role in intervals (10am, 12noon) and (7am, 9am). In the first interval, the supervisor will be able to acquire all the privileges of the DayDoctor role by activating it and in the second interval, he will be able to acquire all the privileges of the NightDoctor role by activating it along with the SupervisorDoctor role. The SupervisorDoctor role may simply contain some extra privileges that are required for the supervision task during day and night.

3.3 Formal Definitions of Restricted Hierarchies

We now formally define the *weakly restricted* and *strongly restricted* forms of each hierarchy type discussed in the previous section.

Definition 4. (*Weakly restricted inheritance hierarchy or I_w-hierarchy*) Let x and y be roles such that $(x \geq_{w,t} y)$, that is, x has a *weakly restricted inheritance-only* relation over y. Then the following holds:

$$\forall p, (x \geq_{w,t} y) \wedge enabled(x,t) \wedge can\_be\_acquired(p,y,t) \\ \rightarrow can\_be\_acquired(p,x,t)$$

We note that for a $(x \geq_{w,t} y)$ relation, only role x needs to be enabled at time t. Role y may or may not be enabled at that time. Similarly, for *weakly restricted* A-hierarchy $(x \succeq_{w,t} y)$ to hold, only role y needs to be enabled as shown in the following definition.

Definition 5. (*Weakly restricted activation hierarchy A_w-hierarchy*) Let x and y be roles such that $(x \succeq_{w,t} y)$, that is, x has a *weakly restricted activation-only* relation over y. Then the following holds:

$$\forall u, (x \succeq_{w,t} y) \wedge enabled(y,t) \wedge can\_activate(u,x,t) \\ \rightarrow can\_activate(u,y,t)$$

Definition 6. (*Weakly restricted general hierarchy or IA_w-hierarchy*) Let x and y be roles such that $(x \gg_t y)$, that is, x has a *weakly restricted general inheritance* relation over y. Then the following holds:

$$(x \gg_{w,t} y) \rightarrow (x \geq_{w,t} y) \wedge (x \succeq_{w,t} y)$$

The *strongly restricted* forms of the hierarchies allow inheritance semantics to be valid only when both the hierarchically related roles are enabled. The following definitions formalize these hierarchies.

Definition 7. (*Strongly restricted inheritance-only hierarchy or I_s-hierarchy*) Let x and y be roles such that $(x \geq_{s,t} y)$, that is, x has a *strongly restricted inheritance-only* relation over y. Then the following holds:

$$\forall p, (x \geq_{s,t} y) \wedge enabled(x,t) \wedge enabled(y,t) \\ \wedge can\_be\_acquired(p,y,t) \rightarrow can\_be\_acquired(p,x,t)$$

Definition 8. (*Strongly restricted activation hierarchy or A_s-hierarchy*) Let x and y be roles such that $(x \succeq_{s,t} y)$, that is, x has a *stronly restricted activation-only* relation over y. Then the following holds:

$$\forall u, (x \succeq_{s,t} y) \wedge enabled(x,t) \wedge enabled(y,t) \\ \wedge can\_activate(u,x,t) \rightarrow can\_activate(u,y,t)$$

Definition 9. (*Strongly restricted general hierarchy or IA_s-hierarchy*) Let x and y be roles such that $(x \gg_t y)$, that is, x has a *strongly restricted general inheritance* relation over y. Then the following holds:

$$(x \gg_{s,t} y) \rightarrow (x \geq_{s,t} y) \wedge (x \succeq_{s,t} y)$$

The *weakly restricted* and *strongly restricted* forms of hierarchies deal with the cases where atleast one of the two roles is enabled. The hierarchies defined in section 3.1 do not consider the enabling times of the related roles. In this sense, the *weakly restricted* and *strongly restricted* hierarchies are specializations of the *unestricted* hierarchy types with an additional requirement that one or both the roles be enabled for the inheritance semantics to be valid.

It is important to note that if inheritance between two roles is defined just by using one of the *unrestricted* types, the inheritance semantics applies even when the roles are not enabled. The benefit of such a case is in the propagation of the inheritance semantics along the hierarchy, as illustrated in Figure 3. Assume that the hierarchy is an *unrestricted* A-hierarchy and consider an interval τ in which only roles $r1$ and $r4$ are enabled. We can see that definition 2 applies to each pair and the result is that any user assigned

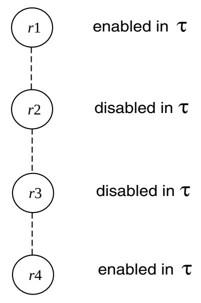


Figure 3: Inheritance through disabled roles

to $r1$ can also activate $r4$. Now suppose that it is an A_w-hierarchy as defined in 5. As $r2$ and $r3$ are both disabled, the activation-inheritance semantics does not apply between them. And hence, it blocks the activation-inheritance semantics between $r1$ and $r4$ also. Thus, no user assigned to $r1$ will be able to activate role $r4$.

3.4 Periodicity and Duration Constraint Expressions

A hierarchical relation between two roles is essentially a constraint on them. Hence, GTRBAC's constraint enabling/disabling expression can be used to *enable* or *disable* a hierarchical relation. Hence, if h is a hierarchical relation x <*f*> y, then h can be enabled/disabled by using the event expression "`enable/disable` h". This allows administrators to dynamically change the hierarchical relationships on a set of roles through predefined periodicity constraitns, run-time requests and triggers.

For specifying the periodicity constraints on a hierarchy, we simply use the GTRBAC model's periodicity expression framework, i.e., we use (*I, P,* `enable/disable` h) to mean that enabling or disabling of a hierarchical relation h is constrained by the interval expression (*I, P*) , i.e., *for all* $t \in Sol(I,P), h$ is enabled/disabled, where $Sol(I,P)$ is the set of valid times instants denoted by (I,P).

Similarly, we use the expression $c_d = (D_h,$ `enable/disable` $h)$ to define the duration constraint on a hierarchy h. D_h indicates how long the hierarchical relation h may hold. In other words, if $t_{end} - t_{start} = D_h$, where t_{start} is the time at which h becomes valid, then *for all* $t \in (t_{end} - t_{start}), h$ holds. Note that t_{start} is not known in advance and hence is determined by the firing of the event "`enable/disable` h" by a trigger or a run-time request. For example, suppose we have the following trigger and a duration constraint on a hierarchical relation:

(tr): `enable` $r \rightarrow$ `enable` $(h = r \succeq_{w,t} r_s)$ `after 10` *min*
(c): (1 *Hours*, `enable` h)

Here, only 10 minutes after role r is enabled will role rs become the senior of r_s. Furthermore, the duration constraint allows r_s to remain senior of r for 1 *hour*.

We also note that a duration constraint can also be of the forms (*I, P,* D_h, `enable/disable` h) or (D, D_h, `enable/disable` h). Constraint c_d = (*I, P,* D_h, `enable/disable` h) implies that the enabling/disabling of h can be done for duration D_h only within the intervals defined by (*I, P*). Now, suppose that constraint (c) above is replaced by ([`Mondays, Fridays`], D_h, `enable` h) , i.e., (*I, P*) indicates every `Mondays` and `Fridays`. In that case, if the trigger (tr) is fired, then, on days other than `Mondays` and `Fridays`, role r_s cannot senior of r.

If the duration constraint (c) is $(D,\ D_h,$ `enable/disable` $h)$, it needs to be first enabled by a constraint enabling expression "`enable` c". If (tr) fires after constraint (c) has been enabled, then the hierarchical relation is enabled and rs becomes the senior of r. Compared to this, constraint $c_d = (D_h,$ `enable/disable` $h)$ indicates that the duration constraint c_d is enabled at all times.

A practical use of such a dynamically changing hierarchical relation is in a case where a senior (acting as a *supervisor*) is allowed to inherit *read-only* permissions of its juniors. For example, a particular end of the week period can be specified when the *supervisor* can read all its juniors' documents, by enabling the senior-junior hierarchical relations. This will allow her/him to carry out progress review of the project as well as the weekly progress of each individual team member that he is supervising. Moffet *et. al* [7] have identified such a supervision-review capability as an important organizational control principle.

3.5 Activation Constraints and Role Hierarchy

Each individual role in a hierarchy may have its own activation constraints. These constraints provide a way of limiting resource use by limiting access to resources. In either of the *inheritance* or *activation* hierarchies, the question of whether such activation constraints have any effect on the permission inheritance becomes an issue. Next, we consider a hierarchy in the presence of cardinality constraints and then generalize the discussion to the other activation constraints.

Assume that the `Programmer` role has a permission set, say P, associated with a licensed software package. Suppose that there are 5 user licenses for the package indicating that only 5 users can concurrently execute any program of the package. Such a constraint could be directly expressed as a cardinality constraint on the `Programmer` role. `Software Engineer`, being senior to `Programmer`, can inherit P. However, at anytime the number of concurrent executions of any particular program by users assigned to the `Software Engineer` role and `Programmer` role needs to be restricted to 5. If we adopt an I or IA- hierarchy, we observe that correctly enforcing such a constraint is not straightforward:

- As the cardinality constraint is applied on the `Programmer` role, it cannot capture the use of the permission set P by the `Software Engineer` role. Hence, there may be five concurrent activations of the `Programmer` role and some activations of the `Software Engineer` role at any time, allowing more than five users to have access to the programs. In such a situation extra measures need to be taken to enforce the cardinality constraint.

- An alternative solution may be to develop a constraint expression on the combination of roles, such as the one that says *the number of concurrent activations of* `Software Engineer` *and* `Programmer` *roles should be at most 5.* However, this introduces other problems because of the fact that P could be only a subset of the permission set associated with the `Software Engineer` role. In such a case, the constraint will enforce the same cardinality constraint on all the permissions assigned to the `Software Engineer` role and not only to P. For example, six concurrent activations of the `Software Engineer` role will not be permitted and hence permissions other than P assigned to it cannot be used, which may not be what we want.

We note that, here, the cardinality control on a role is aimed at controlling the concurrent use of permissions and, hence, we say that the cardinality constraint is *permission-oriented.*

Now suppose that the role hierarchy is an A-hierarchy. As users need to explicitly activate junior roles in order to acquire its permissions, the above problems do not arise. Hence, in the example, if we use the *activation* hierarchy rather than the *inheritance* hierarchy, the intended cardinality control on the use of P is easily enforced. Furthermore, if there is another role `Programmer2` that is also a junior to the `Software Engineer` role that has a permission set P_2 and cardinality constraint (*permission-oriented* as in `Programmer`) of n, the simple overall *activation* hierarchy is an effective solution.

As another example, suppose we want at the most 5 nurses and 3 doctors on active duty at a time and we create two roles, `Doctor` and `Nurse`, such that `Doctor` is senior to `Nurse`. Here, the cardinality constraints are *user-oriented* rather than being *permission-oriented* in that, by imposing the cardinality constraint of 3 on the `Doctor` role and 5 on the `Nurse` role, we want to restrict scheduling at the most 3 doctors and 5 nurses at a time. We can assume that there is no need to control the permission distribution associated with the `Doctor` and `Nurse` roles, as in the previous case.

Now assume that we use an A-hierarchy. This means, when there are 3 doctors and 5 nurses in active duty, the doctors do not have permissions that are associated with the `Nurse` role, as they cannot activate the `Nurse` role. If we want each doctor to also be able to use permissions associated with the `Nurse` role every time s/he is active, by making her/him activate both the roles, then only two nurses will be able to activate the `Nurse` role. This is not what we intend to enforce. However, if we adopt an I-hierarchy or an IA-hierarchy, the problem does not arise, because, the permissions associated with the `Nurse` role are implicitly assigned to the `Doctor` role too and there is no need of explicitly enabling the `Nurse` role by a user assigned to the `Doctor` role.

Thus, we can see that an I-hierarchy or an IA-hierarchy can capture any activation constraint on roles when the cardinality control implies the control on the number of users, whereas an A-hierarchy captures any activation constraint on roles when the activation control implies control on the distribution of permissions.

Similar to the cases in cardinality constraint, an I-hierarchy or an IA-hierarchy is appropriate when other activation constraints imply a *user-oriented* control, whereas an A-hierarchy is appropriate when the activation constraints imply a *permission-oriented* control. Furthermore, the prevalent concept of a role as a *set of permission* implies that the *permission-oriented* activation control is a phenomenon that is closer to the RBAC concepts than the *user-oriented* activation control.

4. RELATED WORK

Several researchers have addressed issues related to inheritance semantics in RBAC [3, 7, 8, 11]. However, none has addressed issues concerning the inheritance relation when temporal properties are introduced. We have used the separate notions of hierarchy using *permission-usage* and *role-activation* semantics similar to the one proposed by Sandhu [11] and have strengthened Sandhu's argument that the distinction between the two semantics is very crucial. Sandhu's argument is based on the fact that the simple usage semantics is inadequate for expressing desired inheritance relation when certain *dynamic* SoD constraints are used between two roles that are hierarchically related, whereas, here, we emphasized the need for such distinction to capture the inheritance semantics in the presence of various temporal constraints. Sandhu's notion of activation hierarchy extending the inheritance hierarchy corresponds to the *IA*-hierarchy and our *I*-hierarchy corresponds to Sandhu's *usage*-hierarchy [11]. In [3], Giuri has proposed an activation hierarchy based on AND and OR roles. However, these AND-OR roles can be easily simulated by Sandhu's ER-RBAC96 hierarchies as well as our three basic hierarchies, making Giuri's model a special case of ER-RBAC96 and GTRBAC [11].

In order to address the needs of control principles in an organization, which include *separation of duty*, *decentralization* and *supervision and review*, Moffet *et al.* [7] have identified three types of hierarchies: *isa* hierarchy, *activity* hierarchy and *supervision* hierarchies. They show that for addressing these control principles completely , we need a dynamic access control model and a hierarchy that allows restrictive inheritance as well as dynamic propagation of access rights [7]. We believe that GTRBAC's temporal constraint framework with trigger as well as constraint enabling mechanisms, and the temporal hierarchies can provide the modeling capability to address some, if not all, of these issues. Due to space limitation, we leave that for future work.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have addressed the key issues concerning the effects of various temporal constraints of the GTRBAC model on a role hierarchy. We showed that the distinction between the *inheritance-only*, *activation-only* and *general-inheritance* hierarchies is useful in capturing the hierarchy semantics in the presence of temporal constraints. This further strengthens Sandhu's [11] claim that the distinction between the two is very crucial, although the motivations he presents [11] are very different from our motivations that essentially derive from the introduction of temporal properties. Our inheritance hierarchies also have different levels of support for the principle of least privilege, which is also considered one of the strong virtues of RBAC models. We also showed that the *unrestricted* hierarchy types augmented with the enabling times of the related roles results in the *strongly restricted* and *weakly restricted* forms of inheritance. The *strongly restricted* versions of the three cases have the same temporal requirements, that is, both the junior and the senior roles must be enabled at the time the hierarchical relations are effective. We further introduced the notions of *user-oriented* and *permission-oriented cardinality* constraint (or *activation constraints* in general), which are associated with the *inheritance* and *activation* semantics.

We plan to extend the present work in various directions. The first direction is an extensive investigation on how the various inheritance hierarchies can co-exist on the same set of roles. The possibility of establishing different inheritance relations among the roles in a given set is very promising. It would allow one to support a very large number of different constraints and application semantics. We also plan to develop an SQL-like language for specifying temporal properties for roles and the various types of inheritance relations. Finally, we plan to develop a prototype of such language on top of a relational DBMS.

6. ACKNOWLEDGMENTS

Portions of this work were supported by the sponsors of the *Center for Education and Research in Information Assurance and Security* (CERIAS) of Purdue University.

7. REFERENCES

[1] E. Bertino, P. A. Bonatti, and E. Ferrari. Trbac: A temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(4):65–104, September 2001.

[2] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, September 1999.

[3] L. Giuri. Role-based access control: A natural approach. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[4] T. Jaeger and J. E. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information System Security*, 4(2):158–190, May 2001.

[5] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. Generalized temporal role based access control model (GTRBAC) (Part I)- specification and modeling. Technical Report CERIAS TR 2001-47, Purdue University, 2001.

[6] J. B. D. Joshi, A. Ghafoor, W. Aref, and E. H. Spafford. Digital government security infrastructure design challenges. *IEEE Computer*, 34(2):66–72, February 2001.

[7] J. D. Moffet and E. C. Lupu. The uses of role hierarchies in access control. In *Proceedings of 4th ACM Workshop on Role-Based Access Control*, October 1999.

[8] M. Nyanchama and S. Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3–33, 1999.

[9] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, May 2000.

[10] J. S. Park, R. Sandhu, and G. J. Ahn. Role-based access control on the web. *ACM Transactions on Information and System Security*, 4(1):37–71, February 2001.

[11] R. Sandhu. Role activation hierarchies. In *Proceedings of 2rd ACM Workshop on Role-based Access Control*, pages 65–79, Fairfax, Virginia, October 22-23 1998.

[12] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.