CERIAS Tech Report 2001-52

Temporal Hierarchy and Inheritance Semantics for GTRBAC

James B. D. Joshi[1], Elisa Bertino[2], Arif Ghafoor[1]
Center for Education and Research in
Information Assurance and Security
&
[1]School of Electrical and Computer Engineering, Purdue University,
West Lafayette, IN 47907
[2]Dipartimento di Scienze dell' Informazione, Universitia' di Milano

Temporal Hierarchy and Inheritance Semantics for GTRBAC

James B. D. Joshi[#], Elisa Bertino[*], Arif Ghafoor[#]

CERIAS
and
[#]School of Electrical and Computer Engineering, Purdue University, USA
{*joshij, ghafoor*}*@ecn.purdue.edu,*
[*]Dipartimento di Scienze dell' Informazione, Universita' di Milano,
Milano, Italy
berino@dsi.unimi.it

Abstract

A Generalized Temporal Role Based Access Control (*GTRBAC*) *model that captures an exhaustive set of temporal constraint needs for access control has recently been proposed. GTRBAC's language constructs allow one to specify various temporal constraints on role, user-role assignments and role-permission assignments. However, the presence of 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.*

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

1 Introduction

Role based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access control (DAC and MAC) models [7, 8, 10, 14, 15, 17], which have some inherent limitations [10]. Several beneficial features such as policy neutrality, support for least privilege, efficient access control management, are associated with RBAC models [6, 9, 17]. 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 [6]. RBAC models have also been found suitable for addressing security issues in the Internet environment [1, 10, 16], and show promise for newer heterogeneous multidomain environments that raise serious concerns related to access control across domain boundaries [5, 9].

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 [3]. To address general time-based access control needs, Bertino *et al.* propose a Temporal RBAC model (TRBAC), which has been recently generalized by Joshi *et al.* [11]. The Generalized-TRBAC (GTRBAC) model [11] 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 assignment 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, but no one has done so. An *enabled* role becomes *active* when a user acquires the permissions assigned to the role in a session. An open issue in the GTRBAC model, as well as in the TRBAC model [4] is the interplay between temporal constraints and role inheritance hierarchy.

Many researchers have highlighted the importance and use of role hierarchies in RBAC models [12, 18, 19]. 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 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 the need for different inheritance semantics for role hierarchies has already been recognized by Sandhu [19]. He has proposed the ER-RBAC96 model that incorporates a distinction between two types of role hierarchy: *usage* hierarchy that applies *permission-usage* semantics and *activation* hierarchy that uses *role activation* 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. An *activation* hierarchy extends "*permission inheritance hierarchy to roles that are stipulated to have dynamic separation of duty* (SoD)" [19]. Our analysis further strengthens his arguments and shows that, in presence of timing constraints on various entities, the separation of the *permission-usage* and the *role-activation* semantics provides a basis for capturing various inheritance semantics of a hierarchy in presence of temporal constraints. We show that these hierarchies can further be divided into sub-types, to account for the subtle effects of temporal constraints.

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

2 Generalized Temporal Access Control Model (GTRBAC)

The GTRBAC model [11] is an extension of the TRBAC model [4]. The 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 allows the specification of the following constraints and events:

1. *Temporal constraints on role enabling/disabling*: These constraints allow one to specify the time intervals during which a role is enabled. 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 allow 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. The duration constraints may be on role enablings, user-role assignments or role-permission assignments.

6. *Triggers*: Triggers allow one to express dependency among GTRBAC events.

Table 1 summarizes the constraint types and expressions of the GTRBAC model. The GTRBAC model extends the safety notion of the TRBAC model to show that there exists an execution model for it. The periodic expressions (*I*, *P*) used in the constraint expressions are based on those in [2, 13]. *D* expresses the duration specified for a constraint. For more details, we refer the readers to [11].

We illustrate with an example the GTRBAC specification of an access control policy. Table 2 contains the GTRBAC specification of a hospital's access policy. Groupings labeled 1, 2, 3, 4 and a, b, c, d 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 the DayDoctor role on

Mondays, *Wednesdays* and *Fridays*, whereas *Bill* can assume the DayDoctor role 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 10 am and 3pm.

Table 1. Constraint Expressions

Constraint categories	Constraints		Expression
Periodicity Constraint	User-role assignment		(I, P, pr:$\texttt{assign}_U$/$\texttt{deassign}_U$ *r* to *u*)
	Role enabling		(I, P, pr:enable/disable *r*)
	Role-permission assignment		(I, P, pr:$\texttt{assign}_P$/$\texttt{deassign}_P$ *p* to *r*)
Duration Constraints	User-role assignment		([(I, P)| D], D_U, pr:$\texttt{assign}_U$/$\texttt{deassign}_U$ *r* to *u*)
	Role enabling		([(I, P)| D], D_R, pr:enable/disable *r*)
	Role-permission assignment		([(I, P)| D], D_P, pr:$\texttt{assign}_P$/$\texttt{deassign}_P$ *p* to *r*)
Duration Constraints on Role Activation	Total active role duration	Per-role	([(I, P)| D], D_{active}, [$D_{default}$], $\texttt{active}_{R\_total}$ *r*)
		Per-user-role	([(I, P)| D], $D_{uactive}$, *u*, $\texttt{active}_{UR\_total}$ *r*)
	Max role duration per activation	Per-role	([(I, P)| D], D_{max}, $\texttt{active}_{R\_max}$ *r*)
		Per-user-role	([(I, P)| D], D_{umax}, *u*, $\texttt{active}_{UR\_max}$ *r*)
Cardinality Constraint on Role Activation	Total no. of activations	Per-role	([(I, P)| D], N_{active}, [$N_{default}$], $\texttt{active}_{R\_n}$ *r*)
		Per-user-role	([(I, P)| D], $N_{uactive}$, *u*, $\texttt{active}_{UR\_n}$ *r*)
	Max. no. of concurrent activations	Per-role	([(I, P)| D], N_{max}, [$N_{default}$], $\texttt{active}_{R\_con}$ *r*)
		Per-user-role	([(I, P)| D], N_{umax}, *u*, $\texttt{active}_{UR\_con}$ *r*)
Trigger	E_1 ,…, E_n , C_1 ,…, C_k → *pr:E* after ? *t*		
Constraint Enabling	pr:enable/disable *c* where $c \in \{(D, D_x, pr{:}E), (C), (D, C)\}$		
Run-time Requests	*Users' activation request*		(s:(de)activate *r* for *u* after ? *t*))
	Administrator's run-time request		(pr:$\texttt{assign}_U$/de-$\texttt{assign}_U$ *r* to *u* after ? *t*)
			(pr:enable/disable *r* after ? *t*)
			(pr:$\texttt{assign}_P$/de-$\texttt{assign}_P$ *p* to *r* after ? *t*)
			(pr:enable/disable *c* after ? *t*)

Table 2. Example GTRBAC access policy for a medical information System

1	a.	(*DayTime*, enable DayDoctor), (*NightTime*, enable NightDoctor)
	b.	((M, W, F), $\texttt{assign}_U$ *Adams* to DayDoctor), ((T, Th, S, Su), $\texttt{assign}_U$ *Bill* to DayDoctor), ((M, W, F), $\texttt{assign}_U$ *Alice* to NightDoctor), ((T, Th, S, Su), $\texttt{assign}_U$ *Ben* to NightDoctor)
	c.	([10am, 3pm], $\texttt{assign}_U$ *Carol* to DayDoctor)
2	a.	($\texttt{assign}_U$ *Ami* to NurseInTraining) ($\texttt{assign}_U$ *Elizabeth* to DayNurse)
	b.	*c*1 = (6 *hours*, 2 *hours*, enable NurseInTraining)
3	a.	(enable DayNurse → enable *c*1)
	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*)
4	a.	(10, $\texttt{active}_{R\_n}$ DayNurse)
	b.	(5, $\texttt{active}_{R\_n}$ NightNurse)
	c.	(2 *hours*, $\texttt{active}_{R\_total}$ NurseInTraining)

In 2a, users *Ami* and *Elizabeth* are assigned 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, there is also an activation constraint 4c on the NurseInTraining role restricting its total activation time to 2 *hours*. The remaining triggers in 3 show that the DayNurse and NightNurse roles are enabled (disabled) 10 *min* respectively after the DayDoctor and NightDoctor roles are enabled (disabled). The constraint set 4 shows some activation constraints. 4a says that there can be at most 10 users activating DayDoctor role at a time, whereas 4b shows that there can be at most 5 users activating the NightDoctor role at a time.

3 Role Hierarchies and Temporal Constraints

In [19], Sandhu 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 Basic types of temporal hierarchy

Here, we take a slightly different approach than in [19]. We explicitly define a hierarchy allowing only permissions to be inherited as *inheritance-only hierarchy* or *I*-hierarchy and the one allowing only the activation semantics as *activation-only hierarchy* or *A*-hierarchy. We further refer to a hierarchy combining both the *inheritance-only* and *activation-only* semantics as *general inheritance hierarchy* or *IA*-hierarchy. 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 dependent time. Table 3 reports the various notations we use in the formal definitions given in what follows.

Table 3. Notations and meanings of expressions

Notation	Meaning
${}^{r}\mathsf{E}_{t}$	*Role r is enabled at time instant t*
${}^{u,r,s}A_{t}$	*Role r is active in user u's session s at time instant t*
${}^{u,r}AS_{t}$	*User u is assigned to role r at time instant t*
${}^{p,r}AS_{t}$	*Permission p is assigned to role r at time instant t*
${}^{u,r}AC_{t}$	*User u can activate role r at time instant t*
${}^{u,p}ACQ_{t}$	*User u can acquire permission p at time instant t*
${}^{u,s,p}ACQ_{t}$	*User u can acquire permission p in user session s at time instant*

Note that "*u is assigned r*" implies that *u* can activate *r*. However, we have used the "*can activate*" phrase to express the notation ${}^{u,r}AC_{t}$. This is 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 role-activation 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*. The notation ${}^{u,p}ACQ_{t}$ implies that *u* can acquire *p* at time *t*. This does not necessarily mean that at time *t*, *u* must be able to acquire *p*, because there may be some other constraints such as activation or SoD constraint that may actually prevent the acquisition. Similar semantics can be attached to the notation ${}^{u,s,p}ACQ_{t}$. The three hierarchies are defined as follows.

Definition 1 (*Inheritance-only hierarchy* or *I-hierarchy*): *Let x and y be roles. We say that x and y are related by an inheritance-only hierarchy at time t, written as* $(x \geq^t y)$, *if the following conditions hold:*

1. $\forall u, \forall p \ {}^{u,x}AS_t \wedge {}^{p,y}AS_t \rightarrow {}^{u,p}ACQ_t$ *and* (*c1.1*)

2. $\forall u, {}^{u,x}AS_t \wedge \neg({}^{u,y}AS_t) \rightarrow \neg({}^{u,y}AC_t)$ (*c1.2*)

x is said a senior role of y, and conversely y is said a junior role of x, with respect to the inheritance-only hierarchy.

In the above definition, condition (1) just says that any user *u* assigned to a senior role *x* can inherit (hence acquire) the junior role *y*'s permissions but it does not rule out the possibility that *u* may be explicitly assigned to *y* also. The second condition ensures that the hierarchical relation only allows inheriting *y*'s permissions by explicitly specifying that *u* cannot activate *y* unless he is also explicitly assigned to *y*. Hence, the hierarchical relation is restricted to the permission-inheritance semantics.

Definition 2 (*Activation hierarchy* or *A-hierarchy*): *Let x and y be roles. We say that x and y are related by activation hierarchy at time t, written as* $(x ?^t y)$, *if the following conditions hold:*

1. $\forall u, {}^{u,x}AS_t \rightarrow {}^{u,y}AC_t$ *and* (*c2.1*)

2. $\forall u, \forall s, \forall p \ {}^{u,x,s}A_t \wedge \neg({}^{u,y,s}A_t) \wedge \neg({}^{p,x}AS_t) \wedge {}^{p,y}AS_t \rightarrow \neg({}^{u,s,p}ACQ_t)$ (*c2.2*)

x is said a senior role of y, and conversely y is said a junior role of x, with respect to the activation inheritance.

In definition 2 above, the first condition states that if *u* is explicitly assigned to *x* then *u* can also activate *y*, even if there is no explicit assignment of *u* to *y*. Note, however, that an explicit assignment is possible but is redundant here. The second condition ensures that if user *u* actually activates *x* but not *y* in the same session, then *u* does not acquire *y*'s permissions that *x* does not have.

Definition 3 (*General Inheritance hierarchy* or *IA-hierarchy*): *Let x and y be roles. We say that x and y are related by (general) inheritance hierarchy at time t, written as* $(x ?^t y)$, *if the following conditions hold:*

1. $\forall u, \forall p \ {}^{u,x}AS_t \wedge {}^{p,y}AS_t \rightarrow {}^{u,p}ACQ_t$ *and* (*c1.1*)

2. $\forall u, {}^{u,x}AS_t \rightarrow {}^{u,y}AC_t$ (*c2.1*)

x is said a senior role of y, and conversely y is said a junior role of x, with respect to the inheritance hierarchy.

The *IA*-hierarchy is the most common form of hierarchy and contains both *permission-inheritance* and *role-activation* aspects of a hierarchy. The first condition states that the user *u* can acquire any permission assigned to the junior role *y*, whereas the second condition specifies that any user assigned to the senior role *x* can also activate *y*. These two conditions are the first conditions of the definitions of the *I*- and the *A*-hierarchies.

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

Property (*Consistency of hierarchies***)**: *Let* $<f> \in \{\geq^t, ?^t, ?^t\}$ *and* $<f'> \in \{\geq^t, ?^t, ?^t\}/\{<f>\}$. *Let x and y be roles such that x* $<f>$ *y; then the condition* $\neg(y <f'> x)$ *must hold*.

The main purpose of a hierarchical relation is the acquisition of permission of junior roles by a senior role by use of 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 Programmer 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), (Software Engineer, Programmer) (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 (Software Engineer, Programmer) is redundant in an *IA*-hierarchy in terms of what 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.

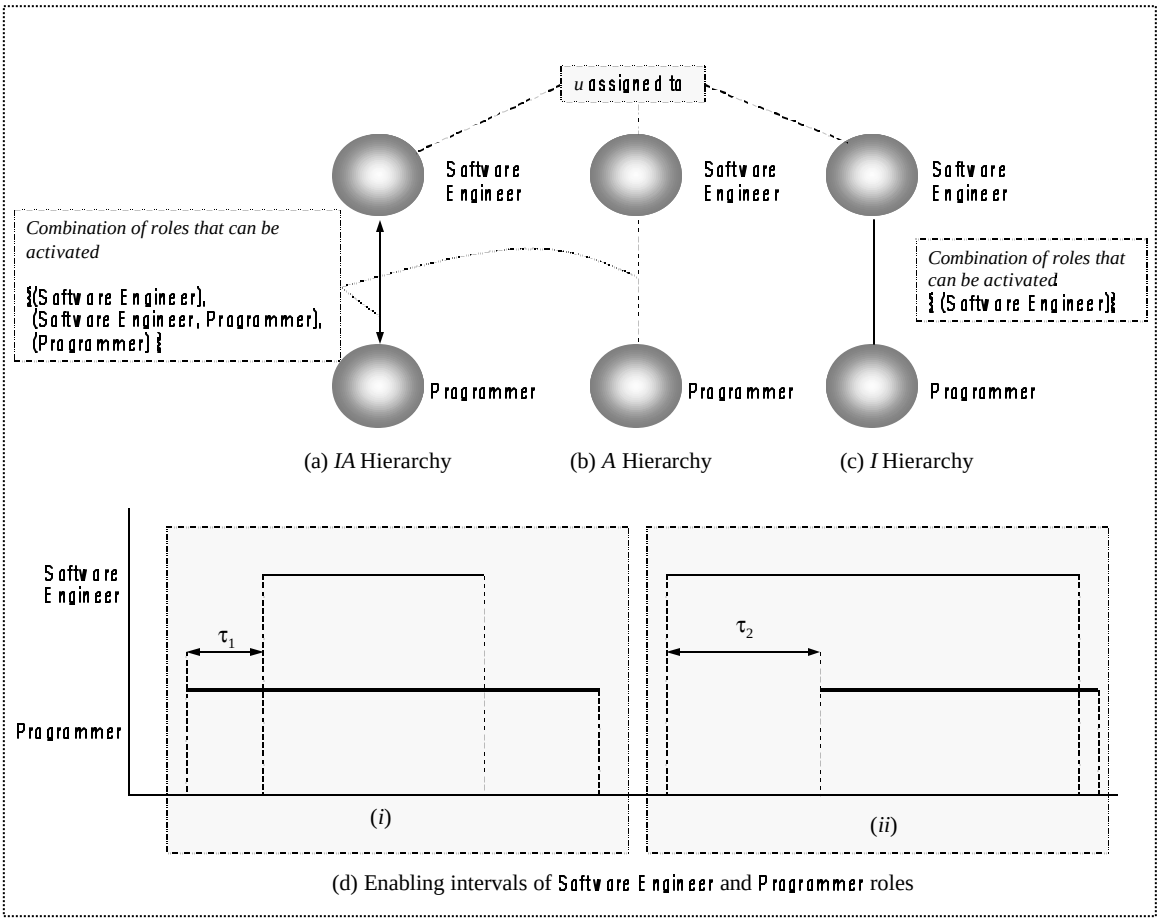

Figure 1. An example hierarchy

Table 4 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 $x_1 \geq^t x_2 \geq^t \ldots \geq^t x_n$ (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 relations over the set of roles hold using *A*-hierarchy and *IA*-hierarchy in third and fourth rows. 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.

Table 4. Hierarchies and associated features ***u* assigned to *x***

Non decreasing hierarchy relation over (x_1, x_2,…, x_n)	**Role-combinations that can be activated in a session by *u***	**Maximal permission set *P* that can be acquired by *u***	**Minimal permission set *P* that can be acquired by *u***	**No. of combinations of P_1, P_2,…, 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	$X \subseteq \{x_1, x_2,\ldots, x_n\}$ (Many redundancies)	P_{max} (By activating $\{x_1\}$)	P_{min} (By activating $\{x_n\}$)	n

The third and forth column shows 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 $P_{max} = \bigcup_{i=1}^{n} P_i$ 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 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 combination 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 $\{P_1, P_1, \ldots, P_n\}$. We note that, in an *IA*-hierarchy, by activation of role x_i, the user essentially acquires permission sets associated with all the roles x_i, x_{i+1} , ..,x_n. Hence, only n 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 to 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 [19].

3.2 Temporal Assignments and Role Hierarchy

A hierarchy in presence of various temporal constraints becomes dynamic as permissions and users can be assigned/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 both *inheritance* and *activation* hierarchies.

Inheritance-only hierarchy (*I*-hierarchy)

As we can see, in an *I*-hierarchy, the permissions of a junior role are implicitly assigned to the senior role itself. However, in 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 t_1, *u* cannot acquire any permissions of 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 t_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. *Unrestricted approach* (I_u): The permissions of the Programmer role are inherited by the Software Engineer role in interval t_2,
2. *Restricted approach* (I_r): The permissions of the Programmer role are not inherited by the Software Engineer role in interval t_2.

Under the unrestricted approach every permission assigned to a junior role is also assigned to its senior roles under an *I*-hierarchy, irrespective of whether the junior role is enabled or disabled. Under the second approach, which is more *restrictive*, each permission assigned to a junior role is also assigned to its senior roles only in intervals where the junior role is also enabled.

Table 5 below summarizes the inheritance semantics of an *I*-hierarchy in presence of temporal constraints. I_u refers to the *I*-hierarchy that adopts the *unrestricted* approach above, whereas, I_r refers to adopting the *restricted* approach. Note that the two act differently only in intervals where the senior role is enabled while the junior role is disabled.

Activation-only hierarchy (*I*-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 inheritance allowed (through the activation of the junior role) in interval t_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 t_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 an *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 t_1 in Figure 1(d)-(*i*). For such a case, we can again delineate the following two approaches:

1. *Unrestricted approach* (A_u): The user *u* is allowed to activate Programmer role in the *A*-hierarchy at any time the Programmer role is enabled.

2. *Restricted approach* (A_r): 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 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 to the hierarchy, starting from the role it is attempting to activate. The second check is required 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.

In Table 5, A_u refers to the *activation* hierarchy that adopts the *unrestricted* approach, whereas A_r refers to that adopting the *restricted* approach. We note that the two act differently only in intervals where the senior role is disabled whereas the junior role is enabled.

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 t_1, the general hierarchy can benefit from the use of role-activation semantics and activate the junior role using the *unrestricted* semantics. Similarly, in interval t_2, the *inheritance-only* semantics can be used and inheritance through the senior role using *unrestricted* semantics. This is shown in Table 5.

Table 5. Inheritance semantics

r_1 is senior of r_2→ ↓Hierarchy Type		*t* r_1 disabled r_2 enabled	*t* r_1 enabled r_2 disabled
Inheritance	I_u	No inheritance in *t*	Inheritance in *t* (*by activating* r_1)
	I_r	No inheritance in *t*	No inheritance in *t*
Activation	A_u	Inheritance in *t* (*by activating* r_2)	No inheritance in *t*
	A_r	No inheritance in *t*	No inheritance in *t*
General Inheritance	IA_u	Inheritance in *t* (*by activating* r_2)	Inheritance in *t* (*by activating* r_1)
	IA_r	No inheritance in *t*	No inheritance in *t*

3.3 Example for hierarchy subtypes

We illustrate with the examples reported in Figure 2 the practical uses of the various kinds of hierarchies listed in Table 5.

Figure 2(a) is an *I*-hierarchy of type I_u. 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_u 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_u-hierarchy instead.

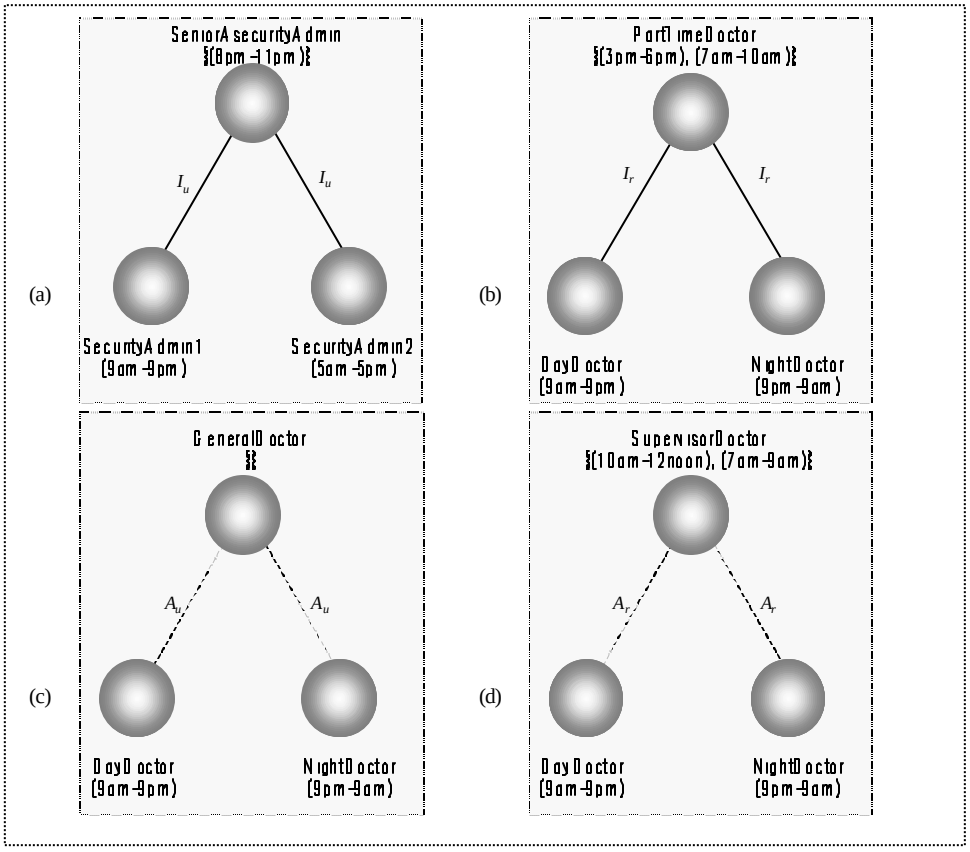

Figure 2. Hierarchy Examples

Figure 2(b), on the other hand, uses the inheritance type I_r. 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_r 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 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 DayDoctor role in the sub-interval (7am-9am) and that of only the NightDoctor 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 permissions are acquired. However if a user is also to be allowed to use the junior roles, we can use IA_r-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[1]. However, since the activation hierarchy is of type A_u, 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_r. 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.

3.4 Formal definitions of hierarchy subtypes

We now formally define the restricted and unrestricted forms of each basic hierarchy type discussed in the previous section.

Definition 4 (*Unrestricted inheritance-only hierarchy* or I_u*-hierarchy*): *Let x and y be roles. We say that x has unrestricted inheritance-only relation over y, written as* $x \geq_u y$ *iff the following holds*:

$$\forall t, {}^{x}\mathsf{E}_t \leftrightarrow (x \geq^t y)$$

Definition 5 (*Unrestricted activation hierarchy* or A_u*-hierarchy*): *Let x and y be roles. We say that x has unrestricted activation relation over y, written as* $x ?_u y$ *iff the following holds*:

$$\forall t, {}^{y}\mathsf{E}_t \leftrightarrow (x ?^t y)$$

[1] GeneralDoctor can be considered as a *virtual temporal role* that is never enabled but can be used for inheritance support. Note that the purpose of a *virtual role* in a non-temporal RBAC is to contain "*qualities that are to be inherited*" [Mof98].

Definition 6 (*Unrestricted general inheritance hierarchy* or *IA_r-hierarchy*): *Let x and y be roles. We say that x has restricted activation relation over y, written as* $x \,?_r\, y$ *iff the following holds*:

$$\forall t\,, ({}^{x}\mathsf{E}_t \vee {}^{y}\mathsf{E}_t) \leftrightarrow (x \,?^{\,t}\, y)$$

Definition 7 (*Restricted hierarchy <f>*): *Let x and y be roles. We say that x has restricted hierarchical relation <f> over y, written as x<f>y , where* $<f> \in \{\geq^t, ?^{\,t}, ?^{\,t}\}$, *iff the following holds*:

$$\forall t\,, ({}^{x}\mathsf{E}_t \wedge {}^{y}\mathsf{E}_t) \leftrightarrow (x <f> y)$$

We note that, definition 7 combines the definition for the *restricted* form of each hierarchy type. This is because, the *restricted* form of the hierarchical relation for all three types require that both the senior and junior roles be enabled at the same time, otherwise acquisition of the junior role's permission is not allowed, as shown in Table 4.

We also note that in a non-temporal RBAC, the roles are enabled at all times and hence ${}^{x}\mathsf{E}_t$ and ${}^{x}\mathsf{E}_t$ are always true. The result is that there are no distinctions among the definitions 4-7.

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 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 moreover 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, as summarized in Table 6, 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.

Table 6. Cardinality constraints and hierarchy

Hierarchy	**Activation constraints**	
	User-oriented	*Permission-oriented*
I-or IA-hierarchy	Suitable	Not suitable
A-hierarchy	Not suitable	Suitable

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 [7, 8, 12, 14, 18, 19]. However, none has addressed issues concerning the inheritance relation when temporal properties are introduced. We have used the separate notion of hierarchy using *permission-usage* and *role-activation* semantics similar to the one proposed by Sandhu [19] 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 presence of various temporal constraints. Sandhu's notion of activation hierarchy extending the inheritance hierarchy corresponds to the IA-hierarchy and our *A*-hierarchy corresponds to Sandhus' relation that relates two roles by activation hierarchy but not by inheritance semantics [19]. In [7, 8], Giuri has proposed an activation hierarchy based on AND and OR roles. However, these AND-OR roles can be easily simulated within Sandhu's ER-RBAC96 model that uses inheritance and activation hierarchies, making Giuri's model a special case of ER-RBAC96 [19].

5 Conclusions and future work

In this paper, we have addressed the key issue 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 presence of temporal constraints. This further strengthens Sandhu's [19] claim that the distinction between the two is very crucial, although the motivations he presents [19] 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 each hierarchy type could be further divided into *restricted* and *unrestricted* forms of inheritance depending upon the relation between the enabling times of *senior* and the *junior* roles. The 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 (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.

References

[1] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, and D.R. Kuhn. Role Based Access Control for the World Wide Web. In *20th National Information System Security Conference*, NIST/NSA, 1997.

[2] E. Bertino, C. Bettini, E. Ferrari, P. Samarati. An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning. *ACM Transactions on Database Systems*, 23(3):231-285, September 1998.

[3] 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, 1999.

[4] E. Bertino, P. A. Bonatti, E. Ferrari. TRBAC: A Temporal Role-based Access Control Model. *ACM Transactions on Information and System Security*, 4(4), 2001 (in print).

[5] J. Biskup, U. Flegel, and Y. Karabulut. Secure mediation: Requirements and design. In *12th Annual IFIP WG 11.3 Working Conference on Database Security*, Chalkidiki, Greece, July 1998.

[6] D. F. Ferraiolo, D. M. Gilbert, and N Lynch. An examination of Federal and commercial access control policy needs. In *Proceedings of NISTNCSC National Computer Security Conference*, pages 107-116, Baltimore, MD, September 20-23 1993.

[7] L. Giuri. A new model for role-based access control. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 249-255, New Orleans, LA, December 11-15 1995.

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

[9] J. B. D. Joshi, A. Ghafoor, W. Aref, E. H. Spafford. Digital Government Security Infrastructure Design Challenges. *IEEE Computer*, Vol. 34, No. 2, February 2001, pages 66-72.

[10] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford. Security models for web-based applications., *Communications of the ACM*, 44, 2 (Feb. 2001), pages 38-72.

[11] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor. Generalized Temporal Role Based Access Control Model (GTRBAC) (Part I)– Specification and Modeling. CERIAS TR 2001-47, Purdue University, USA, 2001.

[12] J. D. Moffet. Control Principles and Role Hierarchies. In *Proceedings of 3 rd ACM Workshop on Role-Based Access Control*, November 1998.

[13] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proc.First International Conference on Information and Knowledge Management*, 1992.

[14] 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.

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

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

[17] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman. Role-Based Access Control Models", *IEEE Computer* 29(2): 38-47, IEEE Press, 1996

[18] R. Sandhu. Role Hierarchies and Constraints for Lattice-based Access Controls. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo Eds., *Computer Security - Esorics'96*, LNCS N. 1146, Rome, Italy, 1996, pages 65-79.

[19] R. Sandhu. Role Activation Hierarchies", In *Proceedings of 2rd ACM Workshop on Role-based Access Control*, Fairfax, Virginia, October 22-23, 1998.