A Generalized Temporal Role-Based Access Control Model

James B.D. Joshi, *Member*, *IEEE*, Elisa Bertino, *Fellow*, *IEEE*, Usman Latif, and Arif Ghafoor, *Fellow*, *IEEE*

Abstract—Role-based access control (RBAC) models have generated a great interest in the security community as a powerful and generalized approach to security management. In many practical scenarios, users may be restricted to assume roles only at predefined time periods. Furthermore, roles may only be invoked on prespecified intervals of time depending upon when certain actions are permitted. To capture such dynamic aspects of a role, a temporal RBAC (TRBAC) model has been recently proposed. However, the TRBAC model addresses the role enabling constraints only. In this paper, we propose a Generalized Temporal Role-Based Access Control (GTRBAC) model capable of expressing a wider range of temporal constraints. In particular, the model allows expressing periodic as well as duration constraints on roles, user-role assignments, and role-permission assignments. In an interval, activation of a role can further be restricted as a result of numerous activation constraints including cardinality constraints and maximum active duration constraints. The GTRBAC model extends the syntactic structure of the TRBAC model and its event and trigger expressions subsume those of TRBAC. Furthermore, GTRBAC allows expressing role hierarchies and separation of duty (SoD) constraints for specifying fine-grained temporal semantics.

Index Terms—Access control, role-based, temporal constraints, role hierarchy, separation of duty.

1 Introduction

Role-based access control (RBAC) models have generated great interest in the security community as a powerful and generalized approach to security management [4], [7], [9], [13], [16]. These models allow the assignment of users and permissions to roles. A user can acquire all the permissions of a role of which he is a member. The RBAC model is naturally suitable to organizations where users are assigned organizational roles with well-defined access control privileges [9]. RBAC models are policy neutral [16] and can express a wide range of security policies including discretionary and mandatory, as well as user-defined or organizational specific policies [14]. Major advantages of RBAC include support for security management and the principle of least privilege [16]. For example, a change in a user's responsibility or role within an organization can be managed efficiently by assigning him a new role and revoking his assignment to any previous role. In addition, role hierarchies and grouping of objects into object classes facilitate the management of permissions [9], [16].

- *J.B.D. Joshi is with the University of Pittsburgh, Room 721, SIS Building, 135 North Bellefield Avenue, Pittsburgh, PA 15260-5259. E-mail: jjoshi@mail.sis.pitt.edu.*
- *Elisa Bertino is with CERIAS Purdue University, Recitation Hall, Oval Drive 656, West Lafayette, IN 47907-2086. E-mail: bertino@cerias.purdue.edu.*
- *U. Latif is with Distributed Multimedia Systems Laboratory, MSEE 206, School of ECE, Purdue University, West Lafyaette, IN 47907. E-mail: questions@techuser.net.*
- *A. Ghafoor is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907. E-mail: ghafoor@ecn.purdue.edu.*

Manuscript received 17 Jan. 2002; revised 1 July 2003; accepted 9 Dec. 2003; published online 18 Nov. 2004.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 115728.

Because of its relevance and above-mentioned benefits, the RBAC model has been extensively investigated [1], [4], [7], [9], [13]. Although this model has attained a considerable level of maturity, there are many applications that cannot be supported by this model and its different variants. In particular, applications with temporal semantics, such as workflow-based systems, fall in this category [6]. In many organizations, processes and functions may have limited time spans or have periodic temporal durations. For instance, a part-time staff member in an organization may be authorized to work only on working days between 9:00 a.m. and 1:00 p.m. If a part-time staff member is represented by a role, enforcing such rules requires that the part-time employee assumes the role only in that interval. Such a requirement can be supported by specifying times when the role can be enabled so that a legitimate user can activate it. A part-time role may be further restricted to only two hours of active time in any given session. In addition, depending upon the organizational needs, the size of the part-time staff assuming a role during the daytime may be different from the size of the part-time staff employed during the night shift.

Bertino et al. have proposed the Temporal-RBAC (TRBAC) model that addresses some of the temporal issues related to RBAC [7]. The main features of this model include periodic enabling of roles and temporal dependencies among roles which can be expressed through triggers. A role is said to be enabled if assumed by a user. Priorities are associated with role events, which in conjunction with a set of precedence rules, are used to resolve conflicts. TRBAC also allows an administrator to issue runtime requests for enabling and disabling a role. The model, however, cannot handle several other important temporal constraints, which are elaborated as follows: First, the model does not include temporal constraints for the user-role and role-permission assignments. It assumes that only roles are enabled and disabled at different time intervals. In this paper, it is demonstrated that in some applications, roles need to be static, that is, they are enabled at all times, while users and permissions assigned to them can be transient. Second, the TRBAC model only handles the temporal constraints on the role enabling but does not support well-defined distinct notions about role enabling and role activation. A role is said to be active if there is at least one user who has assumed that role. Therefore, the TRBAC model cannot handle several constraints related to the activations of a role such as the constraints on the maximum active duration allowed to a user and the maximum number of activations of a role by a single user within a particular interval of time. Third, as the model does not consider duration constraints and constraints on the actual activations of roles, it does not support the notion of enabling and disabling of constraints. The activation constraints should be clearly defined with respect to the enabled time of a role. We, therefore, introduce the notion of constraint enabling/disabling in this paper. Finally, TRBAC does not address the time-based semantics of role hierarchies and separation of duty (SoD) constraints [11], [15], [17], [18].

In this paper, we illustrate the importance of the constraints mentioned above and propose a Generalized TRBAC (GTRBAC) model that subsumes all the essential features of the TRBAC model and can handle all the issues mentioned above. A related work on this topic is the Temporal Data Authorization Model (TDAM) [3], which can express access control policies based on the temporal characteristics of data. However, TDAM does not capture temporal characteristics of user to role assignment. Ahn et al. propose a constraint specification language called *RCL2000* [1]. However, this language does not support specification of temporal constraints.

The paper is organized as follows: In Section 2, we provide the general background on the NIST RBAC model and the notion of periodic expressions. Section 3 presents the description of the temporal constraints in the proposed GTRBAC model. Syntax and semantics of these constraints are also discussed in this section. In Section 4, we address the issue of conflict resolution in GTRBAC. In addition, we introduce the notion of safe temporal constraints and activation base (TCAB) and discuss the execution semantics of the GTRBAC model. In Section 5, we present time-based semantics of role hierarchies and separation of duty constraints. The related work is discussed in Section 6 and the conclusion of the paper is given in Section 7.

2 OVERVIEW

In this section, we briefly overview the NIST RBAC model and the periodic time expression.

2.1 The NIST RBAC Model

The NIST RBAC model as proposed by Ferraiolo et al. consists of four basic components: a set of users `Users`, a set of roles `Roles`, a set of permissions `Permissions`, and a set of sessions `Sessions` [9]. A user can be a human being or an autonomous agent. A role is a collection of permissions needed to perform a certain function within an organization. A permission refers to an access mode that can be exercised on an object in the system and a session relates a user to possibly many roles. In each session, a user can request activation of some of the roles he is authorized to assume. Such request is granted only if the corresponding role is enabled at the time of the request and the user is entitled to activate the role at that time. In the RBAC model, for four sets, namely, `Users`, `Roles`, `Permissions`, and `Sessions`, several functions are defined. The *user role assignment* (UA) and the *role permission assignment* (PA) functions model the assignment of users to roles and the assignment of permissions to roles, respectively. The *user* function maps each session to a single user, whereas the *role* function establishes a mapping between a session and a set of roles activated by the corresponding user in the session. On `Roles`, a hierarchy is denoted by $\leq$. For roles $r_i, r_j \in \texttt{Roles}$, if $r_j \leq r_i$, then r_i inherits the permissions of r_j. In such a case, r_i is a senior role and r_j a junior role.

2.2 Periodic Expression

Periodic time is represented through a symbolic formalism and can be expressed as a tuple $\langle[\texttt{begin}, \texttt{end}], \texttt{P}\rangle$, where `P` is a *periodic expression* denoting an infinite set of periodic time instants, and $[\texttt{begin}, \texttt{end}]$ is a time interval denoting the lower and upper bounds for the instants in `P` [12], [5]. The periodic time uses the notion of *calendar* defined as a countable set of contiguous intervals [7]. We assume a set of calendars containing the calendars *Hours*, *Days*, *Weeks*, *Months*, and *Years*, where *Hours* is the calendar that is assumed to have the finest granularity. A subcalendar relationship can be established among these calendars. Given two calendars C_1 and C_2, C_1 is said to be a subcalendar of C_2, written as $C_1 \sqsubseteq C_2$, if each interval of C_2 is covered by a finite number of intervals of C_1. Calendars can be combined to represent more general *periodic expressions* denoting periodic intervals such as the set of *Mondays* or the set of *the third hour of the first day of each month*. A periodic expression is defined as:

$$\texttt{P} = \sum_{i=1}^{n} O_i.C_i \triangleright x.C_d,$$

where $C_d, C_1, \ldots, C_n$ are calendars and $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$, $C_i \sqsubseteq C_{i-1}$ for $i = 2, \ldots, n$, $C_d \sqsubseteq C_n$, and $x \in \mathbb{N}$. The symbol $\triangleright$ separates the first part of the periodic expression that distinguishes the set of starting points of the intervals, from the specification of the duration of each interval in terms of calendar C_d. For example, $\{all.Years + \{3,7\}.Months \triangleright 2.Months\}$ represents the set of intervals having a duration of two months with their starting times synchronized with the same instant as the third or the seventh month of every year. In practice, O_i is omitted if its value is *all*. In case O_i is a singleton, it is represented by its unique element. Similarly, $x.C_d$ can be omitted when x is equal to 1. A set of time instants corresponding to a periodic expression `P` is denoted by $Sol(I, P)$. Similarly, the set of intervals in (I, P) is denoted by $\Pi(P)$. For simplicity, in this paper, the bounds `begin` and `end`, constraining a periodic expression, is denoted by a pair of *date expressions* of the form `mm/dd/yyyy:hh`. The end point `end` can also be ∞. For instance, [`1/1/2001, 12/31/2001`] denotes all the instants in the year 2001.

TABLE 1
GTRBAC Constraint Expressions

Constraint Categories		Constraints		Expression
Temporal constraints on role enabling, user-role and role-permission assignemtns	*Periodicity Constraint*	User-role assignment		$(I, P, pr{:}\mathtt{assign_U/deassign_U}\ r\ \mathtt{to}\ u)$
		Role enabling		$(I, P, pr{:}\mathtt{enable/disable}\ r)$
		Role-permission assignment		$(I, P, pr{:}\mathtt{assign_P/deassign_P}\ p\ \mathtt{to}\ r)$
	Duration Constraints	User-role assignment		$([(I, P) \| D], D_U, pr{:}\mathtt{assign_U/deassign_U}\ r\ \mathtt{to}\ u)$
		Role enabling		$([(I, P) \| D], D_R, pr{:}\mathtt{enable/disable}\ r)$
		Role-permission assignment		$([(I, P) \| D], D_P, pr{:}\mathtt{assign_P/deassign_P}\ p\ \mathtt{to}\ r)$
Activation constraints	*Duration Constraints on Role Activation*	Total active role duration	Per-role	$([(I, P) \| D], D_{active}, [D_{default}], pr{:}\mathtt{active_{R\_total}}\ r)$
			Per-user-role	$([(I, P) \| D], D_{uactive}, u, pr{:}\mathtt{active_{UR\_total}}\ r)$
		Max role duration per activation	Per-role	$([(I, P) \| D], D_{max}, pr{:}\mathtt{active_{R\_max}}\ r)$
			Per-user-role	$([(I, P) \| D], D_{umax}, u, pr{:}\mathtt{active_{UR\_max}}\ r)$
	Cardinality Constraint on Role Activation	Total no. of activations	Per-role	$([(I, P) \| D], N_{active}, [N_{default}], pr{:}\mathtt{active_{R\_n}}\ r)$
			Per-user-role	$([(I, P) \| D], N_{uactive}, u, pr{:}\mathtt{active_{UR\_n}}\ r)$
		Max. no. of concurrent activations	Per-role	$([(I, P) \| D], N_{max}, [N_{default}], pr{:}\mathtt{active_{R\_con}}\ r)$
			Per-user-role	$([(I, P) \| D], N_{umax}, u, pr{:}\mathtt{active_{UR\_con}}\ r)$
Constraint Enabling				$\mathtt{enable/disable}\ c$ where $c \in \{(D, D_x, pr{:}E), (C), (D, C)\}$
Run-time Requests	*Users' activation request*			$(s{:}\mathtt{(de)activate}\ r\ \mathtt{for}\ u\ \mathtt{after}\ \Delta t))$
	Administrator's run-time request			$(pr{:}\mathtt{assign_U/de\text{-}assign_U}\ r\ \mathtt{to}\ u\ \mathtt{after}\ \Delta t)$
				$(pr{:}\mathtt{enable/disable}\ r\ \mathtt{after}\ \Delta t)$
				$(pr{:}\mathtt{assign_P/de\text{-}assign_P}\ p\ \mathtt{to}\ r\ \mathtt{after}\ \Delta t)$
				$(pr{:}\mathtt{enable/disable}\ c\ \mathtt{after}\ \Delta t)$
Trigger				$E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow pr{:}E\ \mathtt{after}\ \Delta t$

3 Temporal Constraints in GTRBAC—Syntax and Semantics

This section discusses various types of temporal constraints relevant to role-based systems. In particular, we focus our discussion on numerous temporal constraints applied to RBAC components. The proposed GTRBAC model provides duration and periodicity constraints, as well as other forms of specialized activation constraints. A key aspect of the proposed model is that it distinguishes between the notions of role enabling and role activation. Such distinction leads to the notion of states of a role. In the proposed model, a role can assume one of the three states: *disabled*, *enabled*, and *active*. The *disabled* state indicates that the role cannot be used in any user session, i.e., a user cannot acquire the permissions associated with the role. A role in the *disabled* state can be enabled. The *enabled* state indicates that users who are authorized to use the role at the time of the request may activate the role. Subsequently, if a user activates the role, the state of the role becomes *active*. A role in the *active* state implies that there is at least one user who has activated the role. Once in the *active* state, reactivation of the role does not change its state. When a role is in the *active* state, upon deactivation, the role transitions to the *enabled* state provided there is only one session in which it is *active*; otherwise, the role remains in the *active* state. A role in the *enabled* or *active* state transitions to the *disabled* state if a disabling event occurs. The proposed model allows the specification of the following types of constraints:

1. *temporal constraints on role enabling*, *user-role*, and *role-permission assignments*,
2. *activation constraints*,
3. *runtime events*,
4. *constraint enabling expressions*, and
5. *triggers*.

Table 1 summarizes the constraint types and expressions of the GTRBAC model.

Basic event expressions used by the GTRBAC constraint specification language are depicted in Table 2. Priorities are associated with each event in the proposed model. We define $(\mathtt{Prios}, \preceq)$ as a totally ordered set of priorities and assume that $\mathtt{Prios}$ contains two distinct elements $\bot$ and $\top$ such that, for all $x \in \mathtt{Prios}$, $\bot \preceq x \preceq \top$. We use $x \prec y$, if $x \preceq y$ and $x \neq y$. Status predicates, listed in Table 2, are used to capture the state information associated with roles. In GTRBAC, event expressions, priorities, and status predicates are used to express the constraints listed in Table 1. Next, we present the syntax and semantics of the constraint expressions listed in Table 1 and illustrate their use in expressing an access control policy for an example in medical domain.

3.1 Periodicity and Duration Constraints on Role Enabling and Assignments

An important feature of the proposed GTRBAC model is that periodicity and duration constraints can be applied to various components of RBAC. Specifically, by constraining the role enabling or activation times, these constraints can be applied to roles as well as to user-role and role-permission assignments. Depending on the organizational requirements, role enabling, and assignments can be restricted to particular intervals or to a specified duration.

TABLE 2
Events and Status Predicates

Simple Event ($r \in$ `Roles`, $u \in$ `Users`, and p $\in$ `Permissions`)
`enable` *r or* `disable` *r*
`assign`$_U$ *r* `to` *u or* `de-assign`$_U$ *r* `to` *u*
`assign`$_P$ *p* `to` *r* or `de-assign`$_P$ *p* `to` *r*
`enable` *c or* `disable` *c*, (where c is a duration or an activation constraint)
Prioritized Events
pr:E, where pr $\in$ `Prios` *and E is a* simple

Status Predicate (*C*)	**Status Predicate with time** (C_t)	**Semantics** [*for time*]
`enabled`(*r*)	`enabled`(*r, t*)	*r is enabled* [*at time t*]
`u_assigned`(*u, r*)	`u_assigned`(*u, r, t*)	*u is assigned to r* [*at time t*]
`p_assigned`(*p, r*)	`p_assigned`(*p, r, t*)	*p is assigned to r* [*at time t*]
`active`(*r*)	`active`(*r, t*)	*r is active* [*at time t*] *in at least one session*
`u_active`(*u, r*)	`u_active`(*u, r, t*)	*r is active in u's session* [*at time t*]
`s_active`(*u, r, s*)	`s_active`(*u, r, s, t*)	*r is active in u's session s* [*at time t*]
`acquires`(*u, p*)	`acquires`(*u, p, t*)	*u acquires p* [*at time t*]

Periodicity Constraints $(I, P, pr : E)$. Periodicity constraints are used to specify the exact intervals during which a role can be enabled or disabled, and during which a user-role assignment or a role permission assignment is valid. As shown in Table 1, the periodicity constraint expressions have the general form $(I, P, pr : E)$. The pair (I, P) specifies the intervals during which an event E takes place. E can be a role enabling event: "`enable/disable` r" or either of the assignment events: "`assign`$_p$/`deassign`$_p$ p to r" or "`assign`$_U$/`deassign`$_U$ u to r." Pr indicates the priority of event, which will be elaborated in later sections.

Fig. 1 shows an example of periodicity constraints on user-role assignments. The two thick lines at the time axis represent the intervals (t_3, t_6) and (t_8, t_{11}) in which role r is enabled. The lines above the time axis indicate intervals in which users are assigned to role r. The dotted portions of these lines indicate intervals in which user-role assignments are valid, although their assignment may not be in effect because the role is disabled in these intervals. For example, when user u_1 is assigned to role r in interval (t_1, t_5), he can activate role r only in the interval (t_3, t_5), as the role is disabled in the remaining part of interval (t_1, t_5). Similarly, user u_2 is assigned to r in interval (t_4, t_{10}), but can activate the role only in intervals (t_4, t_6) and (t_8, t_{10}). User u_3 is assigned to r in interval (t_2, t_7), but can assume r only in interval (t_3, t_6).

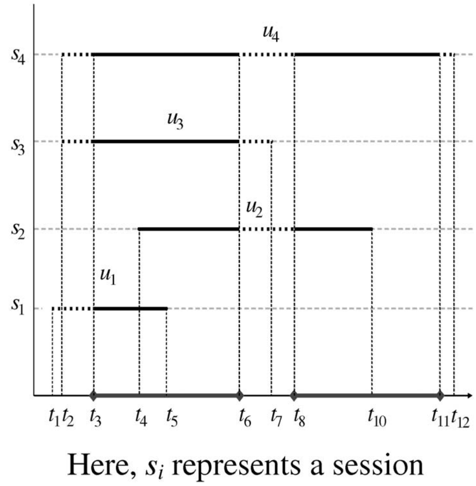

Fig. 1. Periodicity constraint on user-role assignment.

Duration Constraints $([(I, P,)|D], D_x, pr : E)$. Duration constraints are used to specify durations for which enabling or assignment of a role is valid. When an event occurs, the duration constraint associated with the event validates the event for the specified duration only. In case no duration constraint exists for the event, the event remains valid until it is disabled by some other means, e.g., by a trigger.

The general form of the duration constraint expressions for role enabling and assignment is $([(I, P,)|D], D_x, pr : E)$, where x is either `R`, `U`, or `P`, corresponding to events: "`enable/disable` r," "`assign`$_U$/`deassign`$_U$ r to u," and "`assign`$_P$/`deassign`$_P$ p to r," respectively. D and D_x refer to the durations such that $D \leq D_x$. The symbol "|" between (I, P) and D indicates that either (I, P) or D is specified. The square bracket in $[(I, P,)|D]$ implies that this parameter is optional. Accordingly, we have three types of duration constraints:

$$(I, P, D_x, pr : E), (D, D_x, pr : E), \text{ and } (D_x, pr : E).$$

The expression $(I, P, D_x, pr : E)$ indicates that event E is valid for the duration D_x within each valid periodic interval specified by (I, P). $(D_x, pr : E)$ implies that the constraint is valid at all times. Therefore, if event E happens at any time, it is restricted to duration D_x. The constraint $c = (D, D_x, pr : E)$ implies that there is a valid duration D within which the duration restriction D_x applies to event E. In other words, the constraint c is enabled for duration D. The constraint enabling expressions as shown in Table 1 can be used to enable such constraints and the activation constraints discussed later. The constraint enabling/disabling event has the expression of the form "`enable/disable` c," where c is a constraint expression $(D, D_x, pr : E)$. A constraint enabling event corresponds to either a runtime request or a triggered event. The duration constraint expression has the same general form as that of the activation constraint expression, described below. Hence, the semantics of the duration constraints on role enabling and assignments is similar to that of the activation constraints. The examples

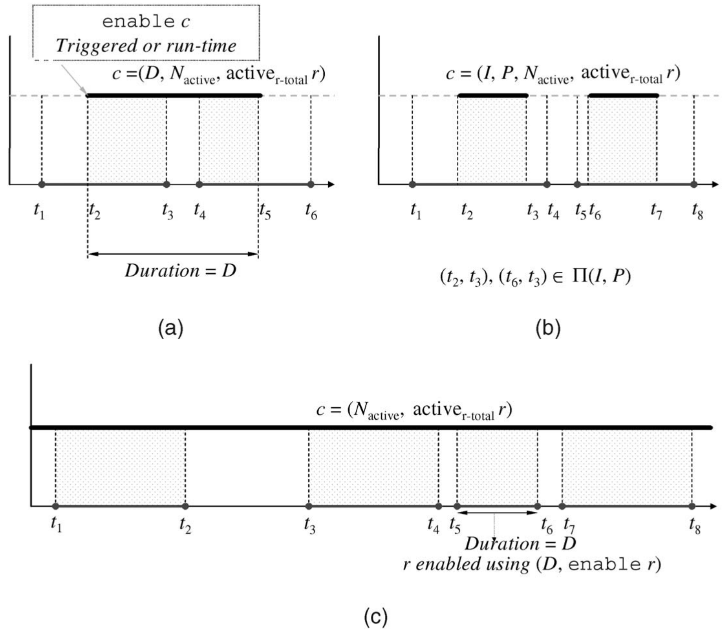

Fig. 2. Constraint enabled (a) for a specifiec duration, (b) in specified intervals, and (c) at all times.

about activation constraints in Fig. 2 also illustrate how duration constraints mentioned here are imposed.

3.2 Temporal Constraints on Role Activation

Role activation requests are made at the discretion of a user at arbitrary times and, hence, periodicity constraints on role activations should not be imposed. On the other hand, duration constraints can be imposed on role activations. In the proposed model, duration constraints on role activations can be classified into two types: *total active duration constraint* and *maximum duration per activation constraint*. The *total active duration constraint* on a role restricts the span of the role's activation duration in a given period to a specified value. After the users have utilized the specified total active duration for a role, the role cannot be activated again, even though it may still be enabled. It can be noted that the *total active duration* allowed for a role may span a number of intervals in which the role is enabled. The *total active duration* may be specified on *per-role* and *per-user-role* basis. *Per-role* constraint restricts the total active duration for a role. Once the sum of all the activation durations of a role reaches the maximum allowed value, no further activation of the role is allowed and the current activations are terminated. *Per-user-role* constraint restricts the total active duration for a role by a particular user. Once a user utilizes the total active duration of his role, he is not allowed to further activate the role, whereas other users may still activate the role.

The *maximum duration constraint per activation* restricts the maximum allowable duration for each activation of a role. Once such *duration* expires for a user, the role activation for that user becomes void. However, there may still be other activations of the same role in the system, including one by the same user in some other session. This constraint can also be specified on a *per role* or *per user role* basis. A *per role* constraint restricts the maximum active duration for each activation of a role for any user, unless there is a *per user-role* constraint specified for that user. A *per-user-role* constraint restricts the maximum active duration allowed for each activation of a role by a particular user. Activation duration can be limited within a prespecified interval.

In some applications, restrictions on the number of concurrent activations of a role may be required for controlling access to critical objects or resources. For example, we may want to ensure that a single user does not access all the resources while others are denied the access. Such cardinality restriction on role activation can be categorized into two types: *total n activations constraint* and *maximum n concurrent activations constraint*. In the first category, a role is limited to a total of n activations. This constraint may also be specified on *per-role or per-user-role* basis. The *per-role* constraint allows at most n activations of a role in a given period of time, irrespective of whether these activations occur simultaneously in different sessions or at different times. Similarly, the *per-user-role* constraint restricts a total of n activations of a role by a specified user.

In the second category, a role is restricted to n concurrent activations at any time. A constraint on a *per-role* basis may be specified to restrict the number of concurrent activations of a role to a maximum value. The activation of these roles may be associated with the same or different users. On the other hand, the *per-user-role* constraint restricts the total number of concurrent activations of a role by a particular user to a given value. Different users may have different permissible upper limits on the number of concurrent activations of the same role.

Activation constraints have the general form

$$([(I, P)|D], C),$$

where C represents the restriction applied to a role activation. For example,

$$C = (D_{\text{active}}, [D_{\text{default}}], \texttt{active}_{\text{R\_total}}\ r)$$

corresponds to the *total active role duration-per-role* constraint. $[(I, P)|D]$ is an optional temporal parameter and has the same meaning as given by the duration constraints. Therefore, similar to the duration constraints, an activation constraint assumes one of the three forms: (I, P, C), (D, C), or (C). The first two expressions are semantically identical to the expressions for the duration constraints. Constraint (C) implies that the activation restriction specified by C applies to each enabling of the associated role. If C is a *per-role* constraint, it has an optional default parameter that can be used to specify the default value corresponding to the *per-user-role* restriction. For example, if $C = (D_{\text{active}}, [D_{\text{default}}], \texttt{active}_{\text{R\_total}}\ r)$, then D_{default} indicates that the *default per-user-role* active duration is the value applied to all the users assigned to the role. In case D_{default} is not specified, it is assumed to be equal to the *per-role* value, D_{active}. Parameters of other activation constraints can be similarly interpreted.

Fig. 2 illustrates the three different forms of an activation cardinality constraint C. In Fig. 2a, the constraint c is of form (D, C). In this case, the role is enabled in the intervals (t_1, t_3) and (t_4, t_6). A trigger or a runtime request can enable this constraint at time t_2 (i.e., event "enable c" occurs). Subsequently, c becomes valid for duration D, which in this case corresponds to interval (t_2, t_5). However, within interval (t_2, t_5), a subinterval (t_3, t_4) can exist in which role r is not enabled. The cardinality constraint c implies that the total number of activations of role r in the intervals (t_2, t_3) and (t_4, t_5) combined should not exceed N_{active}.

Fig. 2b illustrates an activation constraint of the form $c = (I, P, C)$. Here, (t_2, t_3) and (t_6, t_7) are intervals in (I, P) and, hence, during each of these intervals the total number of activations of role r is restricted to N_{active}. Fig. 2c shows a constraint of the form $c = (C)$, where, for each enabling period of r, constraint (C) is valid. For example, role r is enabled by a periodicity constraint in the intervals (t_1, t_2), (t_3, t_4), and (t_7, t_8). During each of these intervals, at most N_{active} activations of role r are allowed. Furthermore, role r can also be enabled in interval (t_5, t_6) because of the duration constraint $(D, \texttt{enable}\ r)$. The activation constraint c is then also applicable to this interval, for which only N_{active} activations of role r are allowed.

3.3 Runtime Requests, Triggers, and Constraint Enabling

As mentioned earlier, a user's request to activate a role is made at his discretion. In GTRBAC, such a request is modeled as a runtime event. Similarly, the administrators' runtime requests to initiate events that may override any existing valid events are also modeled. Such events can be used to override a predefined policy to make useful changes in the policy. For example, an administrator may initiate events to disable roles detected to be in use by some malicious users. A relevant requirement in many application domains is the need of automatically executing certain actions due to the occurrence of an event, such as the enabling or disabling of a role. In GTRBAC, we model such dependencies among events by using triggers. In addition, the duration constraints on role enabling and assignments and role activation constraints can be enabled for a prespecified interval or duration. GTRBAC includes expressions to enable or disable such constraints.

As shown in Table 1, a user's runtime request to activate or deactivate a role can be expressed as: 1) s: $\texttt{activate}\ r\ \texttt{for}\ u\ \texttt{after}\ \triangle t$ and 2) s: $\texttt{deactivate}\ r\ \texttt{for}\ u\ \texttt{after}\ \triangle t$. The priority associated with this request is assumed to be the same as that of event "$\texttt{assign}\ r\ \texttt{to}\ u$" that authorizes the activation of role r by user u. Similarly, *an administrator's runtime request* expression, written as $pr : E\ \texttt{after}\ \triangle t$ is a prioritized event that occurs $\triangle t$ time units after the request. In case the priority and the delay need to be omitted, we set $pr = \top$, where $\top$ represents the highest priority and $\triangle t = 0$.

The trigger expression has the form $E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow pr : E\ \texttt{after}\ \triangle t$, where E_is are simple event expressions or runtime requests, C_is are status predicates, $pr : E$ is a prioritized event expression with $pr \prec \top$, E is a simple expression such that $E \in \{s : \texttt{activate}\ r\ \texttt{for}\ u\}$, and $\triangle t$ is a duration expression. It can be noted that because an activation request is made at a user's discretion, the event E should not be "s: $\texttt{activate}\ r\ \texttt{for}\ u$." However, event "$s$: $\texttt{activate}\ r\ \texttt{for}\ u$" can trigger other events and, hence, can be a part of the body of a trigger. Note that the event "s: $\texttt{deactivate}\ r\ \texttt{for}\ u$" is allowed to appear in the head of a trigger as it can be used to enforce access control policy. We illustrate the GTRBAC specification of an access control policy through the following example for a medical information system.

Example 3.1. Consider the GTRBAC access control policy of Table 3, from a medical information system. In row 1a, the enabling times of DayDoctor and NightDoctor roles are specified as a periodicity constraint. The (I, P) forms for $DayTime$ (9:00 a.m.-9:00 p.m.) and $NightTime$ (9:00 p.m. -9:00 a.m.) are as follows:

$$DayTime = ([12/1/2003, \infty], all.Days + 10.Hours \triangleright 12.Hours),$$

and

$$NightTime = ([12/1/2003, \infty], all.Days + 22.Hours \triangleright 12.Hours).$$

In constraint 1b in Table 3, *Adams* is assigned to the role of DayDoctor on *Mondays*, *Wednesdays*, and *Fridays*, whereas *Bill* is assigned to this role on *Tuesdays*, *Thursdays*, *Saturdays*, and *Sundays*. The assignment in constraint 1c in Table 3 indicates that *Carol* can assume the DayDoctor role everyday between 10:00 a.m. and 3:00 p.m. In constraint 2a in Table 3, users *Ami* and *Elizabeth* are assigned to the roles of NurseInTraining and DayNurse, respectively, without any periodicity or duration constraints. In other words, their assignments are valid at all the times. Constraint 2b in Table 3 specifies a duration constraint of 2 *hours* for the enabling time of the NurseInTraining role, but this constraint is valid only for 6 *hours* after the constraint *c*1 is enabled. Consequently, once the NurseInTraining role is enabled, *Ami* can activate the NurseInTraining role at the most for two hours.

Trigger 3a in Table 3 indicates that the constraint *c*1 in row 2b is enabled once the DayNurse is enabled. As a result, the NurseInTraining role can be enabled within 6 *hours*. Trigger 3b in Table 3 indicates that 10 *min* after *Elizabeth* activates the DayNurse role, the NurseInTraining role is enabled for a period of 2 *hours*. As a result, a nurse-in-training

TABLE 3
Example GTRBAC Access Policy for 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),
	c	(*Everyday between* 10am - 3pm, $\texttt{assign}_U$ *Carol* `to` DayDoctor)
2	a	($\texttt{assign}_U$ *Ami* `to` NurseInTraining); ($\texttt{assign}_U$ *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*)
4		(a) (10, $\texttt{active}_{\texttt{R\_n}}$ DayNurse); (b) (5, $\texttt{active}_{\texttt{R\_n}}$ NightNurse); (c) (2 *hours*, $\texttt{active}_{\texttt{R\_total}}$ NurseInTraining)

can have access to the system only if *Elizabeth* is present in the system. In other words, once the roles are assumed, *Elizabeth* acts as a training supervisor for a nurse-in-training. Note that *Elizabeth* can activate the DayNurse role multiple times within a duration of 6 hours after the DayNurse role is enabled. The activation constraint 4c in Table 3 limits the total activation time associated with the NurseInTraining role to 2 *hours*. The constraint set 4 shows additional activation constraints. For example, constraint 4a indicates 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.

4 GTRBAC Conflict Resolution and Execution Semantics

In this section, we address issues related to conflicts that may arise in the GTRBAC model and propose an approach for conflict resolution and generating an execution model. We define set Γ consisting of all the event expressions, constraints, and triggers in a GTRBAC system as Temporal Constraint and Activation Base (TCAB). The set Γ is essentially a set of constraints listed in Table 1. Furthermore, we assume users' and administrators' runtime requests as a sequence

$$RQ = \langle RQ(0), RQ(1), \ldots, RQ(t), \ldots \rangle.$$

Note, $RQ(t) \in RQ$ is a set of runtime requests at time t and may be empty.

4.1 Conflicts in GTRBAC

Various types of conflicts may arise in a GTRBAC system. Unambiguous semantics are needed to capture such conflicting scenarios. For example, both role enabling event caused by a periodicity constraint and role disabling event caused by the firing of a trigger can correspond to the same role and may occur at the same time. Such a scenario gives rise to conflicts. Essentially, there are three categories of conflicts that may occur for a given Γ and a request sequence RQ, as elaborated in Table 4. These include:

1. *Conflicts between events of the same category* (*type* 1 conflicts). Events in the same category are associated with the same pair of states of a role or assignment. For example, event "`enable` r" results in changing the *disabled* state of role r to an *enabled* state whereas event "`disable` r corresponds to changing the *enabled* state of a role to the *disabled* state. Similarly, events "`assign` r `for` u" and "`deassign` r `for` u" belong to the same category. The entries $(a)-(e)$ in Table 4 refer to conflicts among the events belonging to the same category. A pair of events E_1 and E_2 in a row is said to conflict (*written as* $E_2 = Conf(E_1)$) if the corresponding *condition C* holds.
2. *Conflicts between events of different categories*: (*type* 2 conflicts). Conflicts may also arise between events of different categories. For instance, an activation request "`activate` u `for` r" and a role disabling event "`disable` r" are conflicting events if they attempt to occur simultaneously, as a *disabled* role cannot be *active*. Similarly, activation event "`activate` u `for` r" and user-role deassignment event "$\texttt{deassign}_u$ r `to` u" cannot occur at the same time as a user may activate a role only if he is assigned to the role. We also note that events "`enable` r" and "s:`deactivate` r `for` u" do not conflict, even if both events occur simultaneously.
3. *Conflicts between constraints* (*type* 3 conflicts). Conflicts may also occur between two constraints defined for role enabling or role assignment (*type* 3a shown in Table 4). For example, a duration constraint on role enabling, (D_R, `enable` r) and a duration constraint on role disabling (D_R, `disable` r) may occur at the same time if both "`enable` r" and "`disable` r" events are valid at the same time. It can be noted that such conflicts occur because of the underlying conflicting events.

   A conflict can occur between the *per-user activation* constraint and the *per-role activation* constraint (*type* 3b) as shown in Table 4. For example, consider the *per-role* constraint

$$(D_{active}, [D_{default}], \texttt{active}_{\text{R\_total}}\ r)$$

and the *per-user-role* constraint

$$(D_{uactive}, u, \texttt{active}_{\text{UR\_total}}\ r).$$

The first constraint indicates that role r is allowed for an activation duration of $D_{uactive}$, whereas the second constraint specifies that user u is allowed to assume role r for a total activation duration of $D_{uactive}$. If duration $D_{default}$ is specified, then all the users are restricted to a total activation time of $D_{default}$. There

TABLE 4
Conflicting Events

Conflict Category	***Conflicting Events***		E_1	E_2 = **Conf** (E_1)	***Condition*** (C)
Conflicts between events of same category (**Type 1**)	*Role Enabling conflicts*	*a*	`enable` r	`Disable` r'	$(r = r')$
	Assignment conflicts	*b*	`assign`$_U$ r `to` u	`de-assign`$_U$ r' `to` u'	($r = r'$ and $u = u'$)
		c	`assign`$_P$ p `to` r	`de-assign`$_P$ p' `to` r'	($r = r'$ and $p = p'$)
	Activation conflicts	*d*	s`:deactivate` r `for` u	s'`:activate` r' `for` u'	($s = s'$, $r = r'$ and $u = u'$)
	Constraint enabling conflicts	*e*	`enable` c	`disable` c'	$(c = c')$
Conflicts between events of different categories (**Type 2**)	*Activation vs. role disabling*	*f*	s`:activate` r `for` u	`disable` r'	$(r = r')$
	Activation vs. deassignment	*g*	s`:activate` r `for` u	`de-assign`$_U$ r' `to` u'	($r = r'$ & $u = u'$)
Type 3 *constraints*	***Conflicting Constraints***		C_1	C_2 = **Conf** (C_1)	***Condition*** (C)
Conflicts among role enabling and assignments constraints (**Type 3a**)		*h*	(X_1, d_1, E_1)	(X_2, d_2, E_2)	E_2 = Conf (E_1) & occurrence of d_1 and d_2 overlap
Conflicts among activation time constraints (Per-role vs. Per-user-role conflicts) (**Type 3b**)		*i*	(d_a, d_d, `active`$_{\texttt{R\_total}}$ r)	(d_{ua}, u, `active`$_{\texttt{UR\_total}}$ r')	($r = r'$) and ($d_a \neq d_{ua}$ or $d_d \neq d_{ua}$)
		j	(d_{max}, `active`$_{\texttt{R\_max}}$ r)	(d_{umax}, u, `active`$_{\texttt{UR\_max}}$ r')	($r = r'$) and ($d_{max} \neq d_{umax}$)
		k	(n_a, n_d, `active`$_{\texttt{R\_n}}$ r)	(n_u, u, `active`$_{\texttt{UR\_n}}$ r')	($r = r'$) and ($n_a \neq n_{ua}$ or $n_a \neq n_{ua}$)
		l	(n_{max}, `active`$_{\texttt{R\_max}}$ r)	(n_{umax}, u, `active`$_{\texttt{UR\_max}}$ r')	($r = r'$) and ($n_{max} \neq n_{umax}$)

is an inherent ambiguity whether the user u should be allowed a total activation time of $D_{uactive}$ or $D_{default}$. Note, in the *per-user* constraint if $d_{default}$ is not specified, then we assume $D_{default} = D_{active}$. In other words, any single user may activate role r for the entire activation duration of D_{active}. Therefore, the *per-user-role constraint* will again conflict with the per-role constraint.

The GTRBAC model uses the notion of blocked events to resolve conflicts of types 1 and 2, as defined below. When priorities cannot resolve conflicts, the model uses a *negative-takes-precedence* principle to resolve the type 1 conflicts. According to this principle, disabling of a role takes precedence over enabling the role and the deactivation of a role takes precedence over the activation of the role. Similarly, for type 2 conflicts, the event corresponding to role disabling and user-role deassignment is preferred over the activation event, as an enabled role and a valid assignment are prerequisites for role activation. The following definition states these conflict resolution rules.

Definition 4.1.1 (Conflict resolution for Type 1 and Type 2). *Let S be a set of prioritized event expressions and constraints. Let $pr : E$ be a prioritized event expression, where E is an event and $pr \in$ `Prios`. $pr : E$ is said to be blocked by S, if the following conditions hold:*

1. *If there exists a $q \in$ `Prios`, such that $q : \mathrm{Conf}(E) \in S$ and the following holds:*
   a. *If $pr : E$ and $q : \mathrm{Conf}(E)$ result in a type 1 conflict, then either*
      i. *E corresponds to E_1 in Table 4, and $pr \preceq q$ or*
      ii. *E corresponds to E_2 in Table 4 and $q \prec pr$;*
   b. *If $pr : E$ and $q : \mathrm{Conf}(E)$ result in a type 2 conflict, and $E = s :$ `activate` r `for` u.*
2. *If there exists a valid constraint $([(I, P)|D], X)$ that does not permit event $pr : E$ to occur.*

Set of nonblocked events in S is denoted by `Nonblocked`(S). Furthermore, if both type 1 and type 2 conflicts occur, events blocked by type 1 conflicts are removed prior to removing events blocked by type 2 conflicts. In addition, if S has valid constraints of the form $([(I, P)|D], X)$, events blocked by these constraints are evaluated last.

In Definition 4.1.1, condition 1.a.i implies that event "q: disable r" blocks "pr: enable r" if $pr \preceq q$. If, however, $pr \prec q$, then according to condition 1.a.ii, the event "q: enable r" would block the event "pr: disable r." Condition 1.a applies to all the conflicts of *type* 1. Rule 1.b applies to type 2 conflicts depicted in Table 4. According to this rule, events associated with role disabling or user-role deassignment override the role activation events, as role activation by a user depends on both the role enabling and user-role assignments. It is important that a role disabling or user-role deassignment event is not blocked if either one aims to block an activation event. By resolving the type 1 conflicts first, we ensure that an activation event is blocked by a role disabling or user-role deassignment that has not been blocked. Parts b and c of Example 4.1 presented next illustrate the necessity of handling type 1 conflicts prior to handling type 2 conflicts. The second part of the definition indicates that an event may also be blocked by the duration constraints on role enabling and assignments, and activation constraints on roles. When several activation requests for a role are present, some of these activation requests may need to be blocked to enforce an activation constraint. For example, assume that there is a cardinality constraint that says only five activations of role r are to be allowed at a time. If, at a particular time, seven activation requests associated with role r are present, the cardinality constraint on the role will block two of these events. In such a case, a predefined selection criterion is needed to select the activation requests that are to be blocked. Such a selection criterion may depend, for example, on the priority of the activation requests, or the duration for which the activation has existed, or their combination. Furthermore, note the general form of the activation request is "activate r for u after $\triangle t$," which indicates that a user may request role activation in advance. The selection criteria can use the value of $\triangle t$ to determine activation requests that should be blocked. Furthermore, once the type 1 and type 2 conflicts have been resolved, events blocked by constraints following the resolution rule for the type 3b conflicts are selected. The following example further illustrates the notion of blocked events.

Example 4.1. Assume a system with two priorities $\mathtt{H} = \mathtt{High}$ and $\mathtt{VH} = \mathtt{VeryHigh}$ with $\mathtt{H} < \mathtt{VH}$. We consider the following three cases of increasing complexity:

1. Let

$$S = \{\mathtt{H:enable}\ r_0, \mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1, \mathtt{H:disable}\ r_1\}.$$

According to condition 1.a.i of Definition 4.1.1, $\mathtt{Nonblocked}(S) = \{\mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1\}$, since event "H:enable r_0" is blocked by event "H:disable r_0." Similarly, according to condition 1.a.i, event "H:disable r_1" is blocked by "VH:enable r_1."

2. Next, we consider a more complex case for

$$S = \{\mathtt{H:enable}\ r_0, \mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1, \mathtt{H:disable}\ r_1\ \mathtt{VH:(s:activate}\ r_1\ \mathtt{for}\ u)\}.$$

Assume we first resolve type 2 conflicts and then type 1 conflicts. In this case, event "VH: (s:activate r_1 for u)" is removed first as it is blocked by the event "H:disable r_1" as per condition 1.b.i. We then encounter the case where $\mathtt{Nonblocked}(S) = \{\mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1\}$. Note that event "H:disable r_1," that blocks event "VH:(s:activate r_1 for u)," which itself is a blocked event. Hence, blocking of event VH:(s:activate r_1 for u) by H:disable r_1 is not correct.

Alternatively, assume we first remove type 1 conflicts, which results in

$$\mathtt{Nonblocked}(S) = \{\mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1, \mathtt{H:(s:activate}\ r_1\ \mathtt{for}\ u)\}.$$

In the next step, we remove any type 2 conflicts. As event "H:(s:activate r_1 for u)" is not blocked by any event, the final result is

$$\mathtt{Nonblocked}(S) = \{\mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1, \mathtt{H:(s:activate}\ r_1\ \mathtt{for}\ u)\}.$$

3. S is further extended as follows:

$$S = \{\mathtt{H:enable}\ r_0, \mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1, \mathtt{H:disable}\ r_1, \mathtt{VH:(s:activate}\ r_1\ \mathtt{for}\ u_1), \mathtt{H:(s:activate}\ r_1\ \mathtt{for}\ u_2), \mathtt{enable}\ c\},$$

where $c = (1, \mathtt{H:active}_{\mathrm{R\_Total}}\ r_1)$. After resolving type 1 and type 2 conflicts, we generate

$$\mathtt{Nonblocked}(S) = \{\mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1, \mathtt{VH:(s:activate}\ r_1\ \mathtt{for}\ u_1), \mathtt{H:enable}\ c, \mathtt{H:(s:activate}\ r_1\ \mathtt{for}\ u_2)\}.$$

Note that constraint c implies that only one activation of r_1 is permitted. Thus, one of the activation requests must be blocked. Because of the low priority, event "H:(s:activate r_1 for u_2)" is blocked. Hence, the final set of nonblocked events generated is

$$\mathtt{Nonblocked}(S) = \{\mathtt{H:disable}\ r_0, \mathtt{VH:enable}\ r_1, \mathtt{VH:(s:activate}\ r_1\ \mathtt{for}\ u_1)\}.$$

It can be noted that *type* 3a conflicts associated with constraints are mainly due to the underlying conflicting events associated with the constraint expressions. Hence, the resolution of *type* 1 conflicts in Definition 4.1.1 is applicable to *type* 3a conflicts as well. To resolve *type* 3b conflicts, we use a combination of "*per-role*-takes-precedence over the *per-user-role* constraint" and "*more specific constraint* takes precedence" rules. These rules are formally defined below.

Definition 4.1.2: (Conflict resolution for Type 3b conflicts). *Let $(dn_a, [dn_{default}], pr : \mathtt{active}_{\mathrm{R\_x}}\ r)$ be a per-role constraint and $(dn_{ua}, u, \mathtt{active}_{\mathrm{UR\_x}}\ r)$ be a per-user-role constraint defined for the same role r and*

$$\mathrm{R\_x} \in \{\mathrm{R\_Total}, \mathrm{R\_Max}, \mathrm{R\_n}, \mathrm{R\_con}\}.$$

TABLE 5
Constraint Parameter of *u-Snapshot* and *r-Snapshot*

u-snapshot parameter		*r-snapshot parameter*	
d_{ua}	*remaining total duration for which u can activate r*	d_{ra}	*remaining total active duration for r,*
n_{ua}	*remaining number of times that u can activate r,*	n_{ra}	*remaining total number of activations of r,*
d_{m}	*maximum duration for which u can activate r at one time*	d_{rm}	*remaining maximum active duration for r,*
n_{m}	*maximum number of concurrent activations of r that u can have*	n_{rm}	*remaining maximum number of activations of r,*
S_{u}	$S_{\text{u}} = (s_1, s_2, \ldots, s_{\text{k}})$ *is the list of sessions in which u is currently using r.*	*status*	*current status of r*
		P_{r}	*is the set of permissions that are assigned to r.*
D_{u}	$D_{\text{u}} = (d_1, d_2, \ldots, d_{\text{k}})$ *is the list of durations of activations of r by u in each of these sessions.*	U_{r}	*is the set of u-snapshots such that, for all* $ut \in U_r$, *ut.r = r; where ut.r refers to the element r of the u-snapshot ut.*

Then, the following rules apply:

1. *If there are activation constraints of the same type for a role, the highest priority constraint blocks the others as per Definition 4.1.1.*
2. *With respect to the per-role parameter* dn_a *and the per-user-role parameter* dn_{ua}*, the former overrides the latter.*
3. *With respect to the default parameter* $dn_{default}$ *and the per-user-role parameter* dn_{ua}*, the* more-specific *per-user-role constraint overrides the less-specific per-role constraint. In other words, when per-role activation constraint* $(dn_a, dn_{default}, pr : \texttt{active}_{\text{R\_x}}\ r)$ *and per-user-role activation constraint*

   $$(dn_{ua}, u, \texttt{active}_{\text{UR\_x}}\ r)$$

   are both specified, user u *has constraint* dn_{ua}*, but not* $dn_{default}$*.*
4. *The following conditions hold: 1)* $d_a \geq d_{ua}$ *and 2)* $d_a = n.d_{ua}$*, for some* $n > 0$*. In other words, the value of per-user-role should not exceed the value of per-role.*

Note, in Rule 3 the more-specific per-user-role constraint overrides the less specific per-role constraint as both the parameters $dn_{default}$ and dn_{ua} refer to, for example, the number of roles that can be associated with a user at a particular time. For Rule 2, however, parameters dn_a and dn_{ua} do not refer to the same information. If parameter dn_a refers to the total number of activations of a role that is allowed, then dn_{ua} refers to the maximum number of activations of a role allowed for a particular user. Intuitively, the total number of activations of a role by a single user should not exceed the total number of activations allowed for that role. Conflict resolution Rule 2 ensures that the value specified for a role binds the value specified for a user for that role.

4.2 GTRBAC Execution Model

Based on the rules for conflict resolution defined in the previous section, we now discuss the execution semantics of the GTRBAC model. In this section, we define system states and traces, and construct an execution model for GTRBAC. We also provide a definition to capture events that are caused at each instant of time and present a state generation algorithm for constructing new states from the existing states based on the current set of valid constraints.

The dynamics of occurrences of events and various states of role enablings and activations in GTRBAC are represented as a sequence of snapshots. Each snapshot provides the current set of prioritized events and the status of role, user-role, and role-permission assignments as well as that of the activation constraints. To efficiently represent status information in form of snapshots, we first define the following two structures, called *u-snapshot* and *r-snapshot*.

Definition 4.2.1 (u-snapshot/ r-snapshot). *We define:*

1. *A u-snapshot for user* u *with respect to a role* r *as a tuple* $(u, r, d_{\text{ua}}, n_{\text{ua}}, d_{\text{m}}, n_{\text{m}}, S_{\text{u}}, D_{\text{u}})$*, where* $r \in \texttt{Roles}$*,* $u \in \texttt{Users}$ *such that* u *is assigned to* r *and the constraint parameters are as defined in Table 5.*
2. *An r-snapshot for a role* r *as a tuple*

   $$(r, d_{\text{ra}}, n_{\text{ra}}, d_{\text{rm}}, n_{\text{rm}}, status, P_{\text{r}}, U_{\text{r}}),$$

   where $r \in \texttt{Roles}$ *and the other constraint variables are as defined in Table 5.*

These snapshots are used to model events, status of various roles and assignments, and status of constraints obtained by two distinct sequences EV and ST, respectively. The model in the form of system trace is defined below.

Definition 4.2.2 (System Trace). *A* system trace*—or simply a* trace*—consists of infinite sequences of* EV *and* ST*, such that for all integers* $t \geq 0$*:*

```
Algorithm ComputeST
  Input: t, EV, ST, CT;  Output: ST(t);
/ * Initially ST(0) = (r, ∞, ∞, ∞, ∞, disabled, ∅, ∅). For each
   pair (r, u) we use the associated snapshots rt and ut ∈ U_r.
   Assume that CT (t) = {c| enable c ∈ Nonblocked(EV(t))*/
Step 1: Handle assignments */
FOR each E ∈ Nonblocked(EV(t)) DO
  Case (E): de-assign r to u    : U_r = U_r - {ut};
            de-assign p to r    : P_r = P_r - {p};
            assign p to r       : P_r = P_r ∪ {p};
            assign r to u       : U_r = U_r ∪ {(u, ∞, ∞, ∞, ∞, ∅,∅)}
            deactivate r for u: remove(s, S_u, D_u)
/* Step 2: Handle role disabling event */
FOR each (disable r) ∈ Nonblocked(EV(t)) DO
    rt.status = disabled;
    IF (C_x ∈ CT (t)) THEN
        Set per-role parameters of rt to ∞
        FOR each ut ∈ U_r DO
            Set (S_u, D_u) to (∅,∅);
            IF (C_x ∈ CT (t)) OR (C_x-1 ∈ CT (t)) THEN
                Set per-user-role parameters of rt to ∞
/* Step 3: Handle valid constraints */
FOR each ((X, C) ∈ CT (t-1) and (X, C) ∉ CT (t)) where X ∈ {(I, P), D} & C is a per-role activation
constraint DO
    IF (C = C_x) THEN Set per-role parameters of the corresponding rt to ∞
/* Step 4: Handle role enabling events */
FOR each (enable r) ∈ Nonblocked(EV(t)) DO
    IF (rt.status ≠ enabled) /* role is being enabled */
        rt.status = enabled;
        FOR each ([(I, P)|D], C) ∈ CT (t)) Set the per-role parameter of rt to per-role value specified in C
/* Step 5: Handle valid activation requests*/
FOR each (s:activate r for u) ∈ Nonblocked(EV(t)) DO /* assume rt for r and ut for u in rt * /
  rt.n_ra = rt.n_ra - 1; ut.n_ua = ut.n_ua -1;          /* decrement the values */
  FOR each ([(I, P)|D], C) ∈ CT (t)) such that C is a constraint on r DO
    IF (C is per-user-role constraint) THEN
    Set the per-user-role parameter of the corresponding ut to that in C.
    ELSE /* C is a per-role constraint */
        IF (per-user-role default value is specified in C)
            Set the per-user-role parameter of the ut to default value;
        ELSE /* per-user-role default value is not specified in C*/
            Set the per-user-role parameter of ut to the per-role value in C;
  d = min(d_ua, d_m); /* update the remaining role value */
  add(s, d, S_u, D_u);
/* Step 6: Process constraint variables for the currently active roles and user-role activation*/
FOR each r-snapshot DO
    IF status = enabled THEN
        decrement (durations(r));    d_ra = d_ra - |sessions(r)|;
    ELSE
        d_ra = d_ra -1;
        FOR each user assigned to r DO d_ua = d_ua -1
```

```
Functions used are defined as follows:
remove(s, S, D), where, s is a session id,
  S = {s1, s2, ... , sk} and D = {d1, d2, ... ,
  dk} is a procedure that computes (S, D)
  such that S = S – {s} and D = D – {d},
  where d corresponds to s.
add(s, d, S, D), where, s is a session id, d
  is the duration of activation related to s,
  S = {s1, s2, ... , sk} and D = {d1, d2, ... ,
  dk}; after processing, we get S = S ∪ {s}
  and D = D ∪ {d}.
decrement(D), where D = {d1, d2, ... ,
  dk} is a set of integers; after processing
  we get D = {d1-1, d2-1, ... , dk-1 }.
sessions (r) returns a set of sessions
  {s1, s2, ... , sk }in which role r is
  currently activated.
durations(r) returns a set of active
  durations {d1, d2, ... , dk}that
  corresponds to the sessions in
  sessions(r)
```

Fig. 3. Algorithm `ComputeST`.

- *The tth element of EV, denoted as $EV(t)$, is a set of prioritized event expressions. Intuitively, this is the set of events which occur at time t.*
- *The tth element of ST, denoted as $ST(t)$, is a set of r-snapshots corresponding to existing roles at time t. Algorithm `ComputeST` in Fig. 3 is used to compute $ST(t)$ for each t.*

A trace *is called* canonical *if $ST(0)$ = set of r-snapshots of the form $(r, \infty, \infty, \infty, \infty, \texttt{disabled}, \oslash, \oslash)$ for all roles r in the system, i.e., all r-snapshots are initialized to*

$$(r, \infty;, \infty, \infty, \infty, \texttt{disabled}, \oslash, \oslash).$$

We assume that a system starts from an initial state at time $t = 0$, where all the roles are disabled and no user-role assignments, role-permission assignments, or valid activation constraints are active. As the time progresses, the events listed in Table 4 take place, thus changing status of various roles and assignments. The notion of a GTRBAC trace with such an initial state is represented by a canonical trace.

The above definition of a trace enforces the intended semantics of events. The set $\texttt{Nonblocked}(EV(t))$ contains the maximal priority events that occur at time t. We note that Γ and RQ determine a unique state. It can also be noted that the state information contained in $ST(t)$ concerning the active state of roles depends on the activation constraints enabled at time t. A duration constraint or role activation constraint (c) is valid if event "enable c" is in $\texttt{Nonblocked}(EV(t))$. Therefore, given a previous state, event set and the valid activation constraint set, the following proposition holds.

Proposition 4.1 [7]. *Given a sequence EV, and an initial status S_0, a unique trace (EV, ST) is generated with $ST(0) = S_0$.*

The proposition implies that a procedure for generating a unique trace can be developed. Accordingly, we describe an algorithm `ComputeST`, shown in Fig. 3, which computes the next state from an existing state using a given set of events and valid constraints. Based on the unblocked events and the current set of valid constraints, the algorithm updates the state information contained in *r-snapshots* and *u-snapshots*. All the events in $\texttt{Nonblocked}(EV(t))$ happen at time t. The state information represented by the *r-snapshots* in $ST(t)$ contains the effect of the events in

$$\texttt{Nonblocked}(EV(t))$$

on state $ST(t-1)$. In Step 1 of the algorithm shown in Fig. 3, all nonblocked assignment/deassignment and deactivation events are processed. In Step 2, the role disabling events are processed. Note, when a role is disabled, the role-specific and the user-specific parameters are reset to ∞, which indicates that if there are no per-role or per-user-role constraints, then the activation duration and the number of concurrent activations are unlimited. Note, the conflict resolution rules for type 2 conflicts indicate that the role disabling and the user-role deassignment events affect the active sessions related to the corresponding roles and users. Hence, it is important to first process these events and then update the information related to active roles that remain active for the next unit duration.

In Step 3 of the algorithm (Fig. 3), the values of per-role parameters in *r*-snapshots are reverted to their initial value ∞ corresponding to those activation constraints that become invalid. In Step 4, per-role constraint variables in *r*-snapshots of the newly enabled roles are initialized. In Step 5, new activations of roles are processed. In this process, first, the cardinality variables *per-role* and *per-user-role* are decremented to find the remaining number of activations allowed after this activation request has been granted. Next, the users'constraint variables are initialized and session information is entered to the session list. In Step 6, the remaining active duration of each role is decremented. The total role duration is also adjusted accordingly. For the disabled roles, the duration constraint variables, for both roles and users assigned to them, are decremented. Decrementing the duration constraint variables take care of any activation constraint that is valid at the time the associated role is disabled. The following theorem shows that the algorithm terminates correctly. Also, the theorem provides the complexity of the algorithm.

Theorem 4.1 (Correctness and complexity of `ComputeST`): *Given $EV(t)$, $ST(t-1)$, and Γ, the algorithm `ComputeST`:*

1. *produces $ST(t)$ such that the updated status of r-snapshots and u-snapshots in $ST(t)$ satisfies all the constraints in Γ and the valid activation constraints for the interval $(t, t+1)$, and*
2. *terminates, and has complexity*

$$\mathrm{O(n_R(n_U + n_P + n_{Sm}))},$$

where $\mathrm{n_R}$, $\mathrm{n_P}$, $\mathrm{n_U}$, and $\mathrm{n_{Sm}}$ represent the number of roles, permissions, users, and the maximum allowable number of sessions, respectively, in a system.

Note that each case of Step 1 has a complexity of the order of either $n_R.n_U$ (for user-role assignment/deassignment) or $n_R.n_P$ (for role-permission assignment/deassignment). For Steps 2, 3, 4, and 6, the complexity is a constant multiple of n_R. For Step 5, the complexity is in the order of $n_R.n_S$; this is because each activation refers to a session and each session can have at most n_R roles. Hence, the complexity of the algorithm is $O(n_R(n_U + n_P + n_{Sm}))$. A detailed proof of the theorem can be found in [10].

Given a Γ and a request stream RQ, we need to identify events in EV. Intuitively, each event should be caused by some element of Γ or RQ. When a trigger causes a prioritized event, the event expressions in the body of the trigger should not be blocked. Events in EV are formally defined as follows.

Definition 4.2.3 (Caused Events). *Given a trace, a Γ and a request sequence RQ, the set of caused prioritized events at time t, is the least set $\texttt{Caused}(t, EV, ST, \Gamma, RQ)$ (in short, written as $\texttt{CSet}(t)$ below) that satisfies the following conditions:*

1. *If $(I, P, pr : E)$ and $t \in Sol(I, P)$, then $pr : E \in \texttt{CSet}(t)$. (for periodicity constraint)*
2. *If $(pr : E \texttt{ after } \triangle t) \in RQ(t - \triangle t) \triangle t \leq t)$, then $pr : E \in \texttt{CSet}(t)$; (for runtime request)*
3. *If*

$$[E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow p : E \texttt{ after } \triangle t] \in \Gamma$$

and the following conditions hold, then $pr : E \in \texttt{CSet}(t)$; (for triggers):

   a. $0 \leq \triangle t \leq t$.
   b. *$\forall C_i$, such that $(1 \leq i \leq k)$, C_i holds (C_i is C or C_t as shown in Table 2).*
   c. *$\forall E_i$, such that $(1 \leq i \leq n)$, $pr : E_i \in EV(t - \triangle t$ not blocked by $EV(t - \triangle t)$.*

4.
   a. *If $c = (I, P, X) \in \Gamma$ and $t \in Sol(I, P)$. (for duration/activation constraints)*
      i. *$0 \leq \triangle t = (t - t_1) \leq D_x$.*
      ii. *$[B \rightarrow pr : E \texttt{ after } \triangle t] \in \Gamma$ or a runtime request $pr : E \in RQ(t - t_1)$, as a result of which $pr : E \in \texttt{CSet}(t - t_1)$ not blocked $(EV(t - t_1)))))$, then*

$$pr : \texttt{enable } c \in \texttt{CSet}(t) :$$

   b. *If $c = (D, X) \in \Gamma$, where $x \in \{\texttt{U}, \texttt{R}, \texttt{P}\}$, and if there exists a pair t_1, t_2 such that*
      i. *$t_1 \leq t_2$ and $\triangle t_1 = (t - t_1) \leq D$.*
      ii. *$(\exists [B \rightarrow pr : \texttt{enable } c\texttt{after } \triangle t_1] \in \Gamma$ OR $pr : \texttt{enable } c \in RQ(t - t_1)$ as a result of which $\texttt{enable } c \in \texttt{CSet}(t - t_1)$ and is not blocked by $EV(t - t_1))$, then*

$$pr : \texttt{enable } c \in \texttt{CSet}(t);$$

Furthermore, in addition to (a) and (b), if $X = (D_x, pr : E) \in \Gamma$ is a duration constraint such that $x \in \{\mathtt{U}, \mathtt{R}, \mathtt{P}\}$, and the following condition holds

i. $\exists[B \rightarrow pr : E \texttt{ after } \triangle t_2] \in \Gamma$ *OR*

$$pr : E \in RQ(t - t_2),$$

as a result of which $pr : E \in EV(t - t_2)$ and is not blocked by $EV(t - t_2)$,

then pr : $\texttt{enable } c \in \texttt{CSet}(t)$ *and*

$$q : \texttt{enable } c \in \texttt{CSet}(t),$$

where q is the priority specified for c.

Condition C1 implies that all the events scheduled via a periodic event are added into the set

$$\texttt{Caused}(t, EV, ST, \Gamma, RQ).$$

Condition C2 indicates that all the explicit runtime requests are added into the set $\texttt{Caused}(t, EV, ST, \Gamma, RQ)$. Similarly, Condition C3 implies that all the events, scheduled through a trigger, are added to $\texttt{Caused}(t, EV, ST, \Gamma, RQ)$, provided that the conditions C_is specified in the body of the trigger are satisfied and each of the events E_is occurs at time $t - \triangle t$. Furthermore, it is necessary that events E_is are not blocked by any other concurrent event, as indicated by condition C3(c).

Condition C4 implies that all the events not blocked by valid duration or activation constraints are added to $\texttt{Caused}(t, EV, ST, \Gamma, RQ)$. C4(a) defines the condition that must be satisfied by caused events associated with either a duration or activation constraint. Note that events restricted by a duration or activation constraint are caused by either the runtime requests or by the triggers and are not activated by any periodicity constraints. Furthermore, such events must not be blocked by any concurrent event. These conditions are ensured by condition C4(a)(ii).

Condition C4(a)(i) ensures that an event is still valid only if the duration D_x associated with the event has not expired. Similarly, C4(b) implies that all the events that are associated with the duration or activation constraints of the form $c = (D, X)$ are considered. Note, as the start time of D is not known, semantically we require that c itself be enabled for a duration D. In other words, "enable c" is a caused event for D duration. Furthermore, "enable c" should not be blocked by any concurrent event at that time. The condition C4(b)(ii) ensures that these conditions hold. Condition C4(b)(iii) defines those events which are restricted by the constraint c.

It can be noted that the TCABs and request streams determine changes in a system state at each time instant. Next, we define the system behavior induced by TCABs and request streams and address the *safeness* issue. Intuitively, *safeness* implies that for each event in $EV(t)$, there is a definite and known cause.

Definition 4.2.4 (Execution Model). *A trace (EV, ST) is an execution model of a TCAB Γ and a request stream RQ, if for all $t \geq 0$, $EV(t) = \texttt{Caused}(t, EV, ST, \Gamma, RQ)$.*

It is possible that some specifications may yield no execution model, whereas some ambiguous specifications may admit two or more such models [7]. For instance, if an event in $EV(t)$, say $\texttt{enable } r$, triggers another event which in turn causes event $\texttt{disable } r$ to occur, the later one is added in $EV(t)$. According to the conflict resolution rule, event $\texttt{disable } r$ blocks $\texttt{enable } r$. Such a situation is undesirable as the event $\texttt{enable } r$ that is the cause of event $\texttt{disable } r$ is itself being blocked by the event $\texttt{disable } r$. However, if such cases are excluded, the GTRBAC specification yields exactly one model for all the possible runtime requests. There are simple syntactic conditions that prevent any undesirable behavior as a result of conflicting events. Such syntactic conditions—called *safeness*—are introduced next.

4.3 Safe TCABs

We introduce a safeness condition that can be verified in polynomial time and guarantees that a given TCAB has one and exactly one execution model. The notion of dependency graph is essential to analyze the safeness of the execution model. Each TCAB Γ can be represented as a directed labeled dependency graph $DG_{\mathrm{R}} = \langle N, ED \rangle$, where N, a set of nodes, represents the set of all prioritized event expressions $pr : E$ that occur in the head of a trigger $[B \rightarrow pr : E] \in \Gamma$, and ED *(the set of edges) consists of the following triples, for all triggers $[B \rightarrow pr : E] \in \Gamma$, for all events E' in the body B, and for all nodes $q : E' \in N$,*

1. $\langle q : E', +, pr : E \rangle$ and
2. $\langle r : \mathrm{Conf}(E'), -, pr : E \rangle$, *for all* $[r : \mathrm{Conf}(E')] \in N$ such that $q \preceq r$.

Each triple (N_1, l, N_2) represents an edge from node N_1 to N_2, labeled by l. *Given the initial status of the roles and assignments, safeness of Γ implies that the system's behavior is unambiguously determined by* Γ, and RQ. Bertino et al. [7] have shown that for the TRBAC event set, Γ is safe if its dependency graph $\mathrm{DG_R}$ contains no cycles in which some edge is labeled "-." GTRBAC event set subsumes the TRBAC set. Furthermore, the triggers do not allow event "$\texttt{activate } r \texttt{ for } u$" in the head of the trigger, as indicated in Section 4, whereas events can have dependencies expressed in a trigger exactly as specified in TRBAC. Hence, the dependency graph analysis also applies to the GTRBAC. Note that safeness is a sufficient condition for a predictable system behavior. Although it is difficult to find the necessary conditions, even if found, they offer little practical help because such syntactic properties fail to recognize that the ill-formed portions of a program may be harmless because they can never be activated [7]. Checking existence and uniqueness of a model are, in general, NP-hard problems [7]. Algorithm SafetyCheck illustrated in Fig. 4 is used for the safeness verification of a TCAB. The first part of the algorithm builds the dependency graph associated with Γ, and the second part checks for cycles with a negative edge. The correctness of the algorithm can be proven from the results reported in [7]. If Γ is found to be unsafe, then we need to remove a trigger to ensure that a cycle with a negative edge does not exist in the dependency graph of Γ.

5 GTRBAC Temporal Hierarchies and Separation of Duty Constraints

Hierarchies and Separation of Duty constraints play crucial roles in policy specification and security management in an

```
Algorithm SafetyCheck
Input: a TCAB T
Output: true if T is safe, false otherwise
/* construction of the dependency graph */
N := 0; E:= 0;
FOR all [B → pr:E] ∈ T DO
  IF (E = activate r for u) THEN return false;
  N := N ∪{pr:E};
FOR all [B → pr:E] ∈ T DO
  FOR all E' ∈ B such that ∃q; q:E' ∈ N DO
      E:= E ∪ {⟨q:E', +, pr:Ei⟩ };
  FOR all r:conf(E') ∈N such that q    r DO
      E:= E ∪{⟨r:conf(E'), - , pr:Ei⟩}
/* cycle generation and checking */
SCC :=strongly connected components of ⟨N, E⟩
FOR all ⟨N', E'⟩ ∈ SCC DO
   FOR all ⟨X, l, Y⟩ ∈ E' DO
       IF l = '-' THEN return false;
return true;
```

Fig. 4. Algorithm SafetyCheck.

organization. By allowing permission-inheritance, role hierarchies reduce overhead associated with the permission administration [9], [16]. SoDs contain useful restrictions for avoiding possible fraud that users may commit by carrying out conflicting activities [9], [15], [18]. In this section, we present formal semantics of hierarchies and SoDs in the context of time. In a temporal context, it is essential to establish unambiguous semantics of permission-inheritance and role-activation within a hierarchy when enabling and/or activation times of hierarchically related roles are considered. In a role hierarchy, permission-inheritance semantics identify the permissions that a role can inherit from its junior roles. Similarly, once a user is assigned a role, the role-activation semantics identify the set of junior roles that can be activated by that user.

Prior to presenting the temporal hierarchies and time-based SoDs, we introduce four status predicates, namely,

$$\texttt{can\_activate}(u, r, t), \texttt{can\_acquire}(u, p, t),$$
$$\texttt{can\_be\_acquired}(p, r, t), \text{ and } \texttt{acquires}(u, p, s, t)$$

as defined in Table 6. Predicate $\texttt{can\_activate}(u, r, t)$ indicates that user u can activate role r at time t, implying that user u is implicitly or explicitly assigned to role r. Similarly, $\texttt{can\_be\_acquired}(p, r, t)$ implies that permission p is implicitly or explicitly assigned to role r, whereas $\texttt{can\_acquire}(u, p, t)$ indicates that role p is implicitly or explicitly assigned to u. $\texttt{acquires}(u, p, s, t)$ implies that u acquires permission p at time t in session s. Axioms in Table 6 list the key relationships among these predicates and identify the permission-acquisition and role-activation semantics in GTRBAC.

Axiom (1) states that if a permission is assigned to a role, the permission *can be acquired* through that role. According to axiom (2), all the users assigned to a role *can activate* that role. Axiom (3) indicates 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. 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 are governed by the explicit user-role and role-permission assignments.

5.1 Temporal Role Hierarchies

A role hierarchy expands the scope of the permission-acquisition and role-activation semantics beyond the explicit assignments through the hierarchical relations among roles. We define three categories of hierarchies:

1. *unrestricted hierarchies*, in which permission-inheritance and role-activation semantics are not affected by the presence of any timing constraints on the hierarchically related roles,
2. *enabling time restricted hierarchies*, in which the permission-inheritance and role-activation semantics depend on the enabling times of the hierarchically related roles, and

TABLE 6
Extended Status Predicates

Predicate	Meaning
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*
acquires(*u, p, s, t*)	*User u' acquires permission p in session s at time t*

Axioms: *For all r* ∈ Roles, *u* ∈ Users, *p* ∈ Permissions, *s*∈ Sessions, *and time instant t* ≥ 0, *the following implications hold*:

1	assigned(*p, r, t*)→ can_be_acquired(*p, r, t*)
2	assigned(*u, r, t*) → can_activate (*u, r, t*)
3	can_activate (*u, r, t*) ∧can_be_acquired(*p, r, t*)→can_acquire (*u, p,t*)
4	active(*u, r, s, t*) ∧ can_be_acquired(*p, r, t*) → acquires(*u, p, s, t*)

TABLE 7
Role Hierarchies in GTRBAC

Category	Short form	Notation	The following condition *c* holds
Unrestricted hierarchies	*I-hierarchy*	$(x \geq_t y)$	$\forall p, (x \geq_t y) \wedge \texttt{can\_be\_acquired}(p, y, t) \rightarrow \texttt{can\_be\_acquired}(p, x, t)$
	A-hierarchy	$(x \succcurlyeq_t y)$	$\forall u, (x \succcurlyeq_t y) \wedge \texttt{can\_activate}(u, x, t) \rightarrow \texttt{can\_activate}(u, y, t)$
	IA-hierarchy	$(x \succsim_t y)$	$(x \succsim_t y) \leftrightarrow (x \geq_t y) \wedge (x \succcurlyeq_t y)$
Enabling time restricted hierarchies	*Weakly Restricted*		
	I_w-hierarchy	$(x \geq_{w,t} y)$	$\forall p, (x \geq_{w,t} y) \wedge \texttt{enabled}(x, t) \wedge \texttt{canbe\_acquired}(p, y, t) \rightarrow \texttt{can\_be\_acquired}(p, x, t)$
	A_w-hierarchy	$(x \succcurlyeq_{w,t} y)$	$\forall p, (x \succcurlyeq_{w,t} y) \wedge \texttt{enabled}(y, t) \wedge \texttt{can\_activate}(u, x, t) \rightarrow \texttt{can\_activate}(u, y, t)$
	IA_w-hierarchy	$(x \succsim_{w,t} y)$	$(x \succsim_{w,t} y) \leftrightarrow (x \geq_{w,t} y) \wedge (x \succcurlyeq_{w,t} y)$
	Strongly Restricted		
	I_s-hierarchy	$(x \geq_{s,t} y)$	$\forall p, (x \geq_{s,t} y) \wedge \texttt{enabled}(x, t) \wedge \texttt{enabled}(y, t) \wedge \texttt{can\_be\_acquired}(p, y, t) \rightarrow \texttt{can\_be\_acquired}(p, x, t)$
	A_s-hierarchy	$(x \succcurlyeq_{s,t} y)$	$\forall p, (x \succcurlyeq_{s,t} y) \wedge \texttt{enabled}(x, t) \wedge \texttt{enabled}(y, t) \wedge \texttt{can\_activate}(u, x, t) \rightarrow \texttt{can\_activate}(u, y, t)$
	IA_s-hierarchy	$(x \succsim_{s,t} y)$	$(x \succsim_{s,t} y) \leftrightarrow (x \geq_{s,t} y) \wedge (x \succcurlyeq_{s,t} y)$
Activation time restricted hierarchies (Effect of timing constraints on role activation)	*A_a-hierarchy*	$(x \succcurlyeq_{a,t} y)$	$\forall p, (x \succcurlyeq_{a,t} y) \wedge \texttt{active}(u, x, t) \wedge \rightarrow \texttt{can\_activate}(u, y, t)$
	A_{sa}-hierarchy	$(x \succcurlyeq_{sa,t} y)$	$(x \succcurlyeq_{sa,t} y) \rightarrow (x \succcurlyeq_{a,t} y)$
			$\forall u, (x \succcurlyeq_{sa,t} y) \wedge \texttt{active}(u, x, s_1, t) \wedge \texttt{active}(u, y, s_2, t) \rightarrow (s_1 = s_2)$
	A_{ssa}-hierarchy	$(x \succcurlyeq_{ssa,t} y)$	$(x \succcurlyeq_{ssa,t} y) \rightarrow (x \succcurlyeq_{a,t} y)$
			$\forall u, (x \succcurlyeq_{ssa,t} y) \wedge \texttt{active}(u, x, s, t) \rightarrow \texttt{active}(u, y, s, t)$
	A_e–hierarchy	$(x \succcurlyeq_{e,t} y)$	$\forall u, (x \succcurlyeq_{e,t} y) \wedge \texttt{can\_activate}(u, y, t) \rightarrow \texttt{can\_activate}(u, y, t)$,
			$\forall u, (x \succcurlyeq_{e,t} y) \wedge \texttt{active}(u, x, t) \rightarrow \neg\, \texttt{active}(u, y, t)$
			$\forall u, (x \succcurlyeq_{e,t} y) \wedge \texttt{active}(u, y, t) \rightarrow \neg\, \texttt{active}(u, x, t)$
	IA_e–hierarchy	$(x \succsim_{e,t} y)$	$\forall u, (x \succsim_{e,t} y) \wedge \texttt{can\_be\_acquired}(p, y, t) \rightarrow \texttt{can\_be\_acquired}(p, x, t)$
			$\forall u, (x \succsim_{e,t} y) \wedge \texttt{can\_activate}(u, y, t) \rightarrow \texttt{can\_activate}(u, y, t)$
			$\forall u, (x \succsim_{e,t} y) \wedge \texttt{active}(u, x, t) \rightarrow \neg\, \texttt{active}(u, y, t)$
			$\forall u, (x \succsim_{e,t} y) \wedge \texttt{active}(u, y, t) \rightarrow \neg\, \texttt{active}(u, x, t)$

3. *activation time restricted hierarchies*, in which the permission-inheritance and role-activation semantics depend on the active states of the hierarchically related roles.

Table 7 lists the hierarchies and the associated constraints. As shown in the table, unrestricted and enabling-time restricted hierarchies may be of three types: *inheritance-only* hierarchy (*I*-hierarchy), *activation-only* hierarchy (*A*-hierarchy), or *inheritance-activation* hierarchy (*IA*-hierarchy). Condition *c*

TABLE 8
Time-Based SSoD Constraints

Interval constraint on SSoD		The following condition holds
Weak form	$(\pi, SSoD_W(R, u))$	$\forall t \in \pi$ and $\forall r_1, r_2 \in R$ *such that* $r_1 \neq r_2$, $SSoD_W(R, u) \wedge$ `u_assigned`$(u, r_1, t) \rightarrow$ $\neg$`u_assigned`(u, r_2, t)
Strong form	$(\pi, SSoD_S(R, u))$	$\forall r_1, r_2 \in R$ *such that* $r_1 \neq r_2$, $(\exists t \in \pi, (SSoD_S(R, u) \wedge$ `u_assigned`$(u, r_1, t) \rightarrow \neg (\exists t \in \pi,$ `u_assigned`$(u, r_2, t))$
Periodicity constraints on SSoD		**The following condition holds**
Weak form	$(I, P, SSoD_W(R, u))$	$\forall t \in Sol(I, P)$ and $\forall r_1, r_2 \in R$ *such that* $r_1 \neq r_2$, $SSoD_{EW}(R, u) \wedge$ `u_assigned`$(u, r_1, t) \rightarrow$ $\neg$`u_assigned`(u, r_2, t) *Furthermore, we see that* $(I, P, SSoD_W(R, u)) \leftrightarrow \forall \pi \in \prod(I, P), (\pi, SSoD_W(R, u))$
Strong form	$(I, P, SSoD_S(R, u))$	$(I, P, SSoD_S(R, u)) \leftrightarrow \forall \pi \in \prod(I, P), (\pi, SSoD_S(R, u))$
Extended Strong form	$(I, P, SSoD_{ES}(R, u))$	$\forall r_1, r_2 \in R$ *such that* $r_1 \neq r_2$, and $(\exists t \in Sol(I, P), SSoD_{EW}(R, u) \wedge$ `u_assigned`$(u, r_1, t) \rightarrow \neg (\exists t \in Sol(I, P),$ `u_assigned`$(u, r_2, t))$

for the *I*-hierarchy implies that if $(x \geq t_y)$, then according to Axiom (1), the permissions that can be acquired through x include all the permissions assigned to x and all the permissions that *can be acquired through* role y, as shown by condition (*c*) in Table 7. Condition c corresponding to A-hierarchy implies that if user u *can activate* role x, and $x \succeq_t y$, then he *can* also *activate* role y, even if u is not explicitly assigned to y. Implicitly, u cannot acquire y's permissions by merely activating x. The *IA*-hierarchy includes both permission-inheritance and role-activation semantics.

When the enabling intervals associated with hierarchically related roles partially overlap, we need to consider the issue of how inheritance and activation semantics apply in intervals where only one of the roles is enabled. In order to capture the inheritance and activation semantics when the enabling times of the hierarchically related roles partially overlap, we introduce the concept of *weakly restricted* and *strongly restricted* hierarchies. The *weakly restricted* hierarchies allow inheritance or activation semantics in the nonoverlapping intervals, whereas the *strongly restricted* hierarchies allow inheritance and activation semantics only in the overlapping intervals. According to the condition of *weakly restricted I*-hierarchy, if $(x \geq_{w,t} y)$, only role x needs to be enabled at time t for the inheritance semantics to apply. Role y may or may not be enabled at that time. Similarly, for the A_w-hierarchy, $x \succeq_{w,t} y$, only role y needs to be enabled.

In *activation-time restricted* hierarchies, inheritance depends on the activation states of the hierarchically related roles. In an *activation-time* hierarchy (A_a-*hierarchy*) a user can activate the junior role only if he has already activated the senior role. Note that the A_a-hierarchy relation allows activation of the junior and senior roles in the same or different sessions. A *session-specific activation-time* hierarchy (A_{sa}-*hierarchy*) is a more restrictive form of A_a-*hierarchy*, where simultaneous activation of both the senior and junior roles is allowed only within the same session. Another level of restriction is also implied by the *strong session-specific activation time* hierarchy (A_{ssa}-*hierarchy*). In A_{ssa}-*hierarchy*, the additional condition implies that both the roles must be active in the same user session. It can be noted that A_a, A_{sa}, and A_{ssa}-*hierarchies* have mutually inclusive semantics in that they allow juniors to be activated only if the senior is in the active state.

The exclusive-activation-time hierarchy, represented as A_e-*hierarchy*, defines a mutually exclusive semantics to a hierarchy relation. The three conditions for A_e-*hierarchy* imply that only one of the hierarchically related roles can be activated at a time. Furthermore, when a role is activated the permissions of its juniors are not inherited. The IA_e-*hierarchy* extends A_e-*hierarchy* with an additional condition that if a role is activated, permissions that can be acquired through its junior are also acquired.

In a given set of roles, various inheritance relations may exist. Therefore, in order to ensure that the senior-junior relation between two roles existing in one type of hierarchy is not reversed in another, the following *consistency* property needs to be satisfied in a role hierarchy.

Consistency Property 5.1. *Let* $< f >$ *and* $< f' >$ *be hierarchies such that* $< f' > \neq < f >$, *and* x *and* y *be distinct roles such that* $x < f > y$, *then the condition* $\neg(y < f' > x)$ *must hold.*

5.2 Time-Based Separation of Duty Constraints

RBAC models allow *static* and *dynamic* SoD constraints (SSoD and DSoD). We can bind an SoD constraint to be applied in a specific set of intervals by using periodicity constraints of the form (I, P, SoD). Similarly, a duration constraint can be specified for an SoD as $([I, P|D,]D_x, SoD)$. However, different semantic interpretations of the constraint (I, P, SoD) or $([I, P|D,]SoD)$ can exist. Before presenting such interpretations of a periodicity constraint (I, P, SoD), we first observe that for a single interval, say π, the constraint expression (π, SoD) can be interpreted in two ways, as defined for *weak* and *strong* forms of time-based SSoD in Table 8.

The *weak form* $(\pi, SSoD_W)$ implies that within the specified interval there does not exist a time instant in which conflicting roles are assigned to the same user. $(\pi, SSoD_W)$ does not, however, restrict conflicting roles

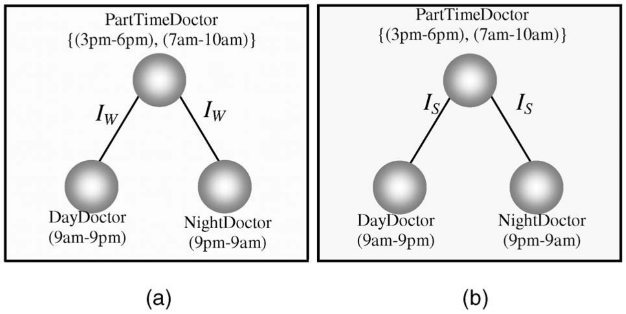

Fig. 5. Example I_w and I_s hierarchies.

from being assigned to the same user at different time instants. The *strong* form $(\pi, SSoD_S)$ implies that within the specified interval, if there is an instant in which a role, say r, is assigned to a user, then at no other instant in π can the user be assigned to a role that conflicts with r. By using these two forms, we obtain three semantic interpretations of periodicity constraint $(I, P, SSoD)$, as listed in the Table 8. The *weak* form $(I, P, SSoD_W)$ implies that at each time instant in (I, P), a user should not be assigned to conflicting roles. $(I, P, SSoD_W)$, however, allows a user to be assigned to two conflicting roles at different time instants. The *strong* form $(I, P, SSoD_S)$ implies that for each recurring intervals in (I, P), the *strong* form of interval constraint $(\pi, SSoD_S)$ applies. The *extended strong* form $(I, P, SSoD_{ES})$ implies that there do not exist two or more time instants in (I, P) for which a user is assigned to conflicting roles. The *weak*, *strong*, and *extended strong* forms also exist for the duration constraints of the form $([I, P|D], D_x, SSoD)$.

Note that Table 8 defines time-based semantics of the SSoD constraint only. The *weak*, *strong*, and *extended* strong forms also exist for periodicity and duration constraints of the forms $(I, P, DSoD)$ and $([I, P|D], D_x\ DSoD)$ on DSoD constraints. These forms capture all the possible ways to express the needed semantics of SoDs with intervals associated with them.

5.3 Examples of Temporal Hierarchies and SoD Constraints

In this section, we present a few examples to illustrate the use of temporal hierarchies and SoD constraints. Examples 5.1 and 5.2 show applications of temporal hierarchies, whereas Example 5.3 illustrates the application of time-based SoD constraints.

Example 5.1. Consider three roles PartTimeDoctor, DayDoctor, and NightDoctor forming a hierarchy as shown in Fig. 5a. The senior role PartTimeDoctor is enabled in intervals (3:00 p.m., 6:00 p.m.) and (7:00 a.m., 10:00 a.m.). As PartTimeDoctor is related to the DayDoctor and NightDoctor through I_W-hierarchy, all the permissions of roles DayDoctor and NightDoctor are inherited by the PartTimeDoctor role in both the intervals (3:00 p.m. - 6:00 p.m.) and (7:00 a.m. - 10:00 a.m.).

Next, we replace the I_W-hierarchy with I_S-hierarchy as shown in Fig. 5b. According to the definition of the I_S-hierarchy, only the permissions of role DayDoctor are inherited by the PartTimeDoctor role in the first interval (3:00 p.m. - 6:00 p.m.), as both the roles are enabled during this interval. The second enabling interval (7:00 a.m. - 10:00 a.m.) of the PartTimeDoctor role overlaps with the enabling times of the two junior roles. The subinterval of interval (7:00 a.m. - 10:00 a.m.) that overlaps with the enabling interval of role DayDoctor is (9:00 a.m., 10:00 a.m.). Hence, in interval (9:00 a.m., 10:00 a.m.), the PartTimeDoctor role inherits the permissions of the DayDoctor role only. Similarly, the subinterval of interval of (7:00 a.m. - 10:00 a.m.) associated with of the PartTimeDoctor that overlaps with the enabling interval of the NightDoctor role is (7:00 a.m. - 9:00 a.m.). Hence, according to the definition of I_S-hierarchy, role PartTimeDoctor inherits only the NightDoctor role's permissions in interval (7:00 a.m. - 9:00 a.m.). Note, the use of I_S-hierarchy in the second case captures the fact that the permissions related to roles DayDoctor and NightDoctor are mutually exclusive.

Example 5.2. As an application of A_{ssa} and A_e-hierarchies, we consider the Bell-LaPadula's model of multilevel security, which assigns subjects and objects security levels that form a lattice and defines two rules to restrict information flow [14]. The first rule, called *simple security property*, states that a subject s can *read* an object o if and only if $l(s) \geq l(o)$, where $l(s)$ and $l(o)$ denote the security levels of the subject and object, respectively. The second rule, called * *property*, states that that a subject s can *write* an object o if and only if $l(o) \geq l(s)$. It has been shown that these Bell-LaPadula rules can be expressed using RBAC constraints on user-role assignment, sessions, and hierarchies [14]. Consider a multilevel system consisting of security levels L_a, L_b, L_c, and L_d forming a lattice as shown in Fig. 6. Fig. 6a shows the GTRBAC hierarchy that represents the two rules. Here, W_x and R_x represent the *write* and *read* roles that correspond to security level L_x. A user with a clearance of L_x is assigned the role R_x. Because of the A_{ssa}-hierarchy between the *write* and *read* role pairs, a user at a particular level can activate only the associated *read-write* role pairs. However, the *I*-hierarchy among write roles and read roles allow the *simple* and **properties* of the BLP model. In some systems, a user may be allowed to use his assigned clearance level or levels below it. Such a case can be captured by the GTRBAC hierarchy shown in Fig. 6b. A user with a clearance of L_x is assigned to role R_x. The IA_e-hierarchy allows a user at a particular level to activate a read-role at that level or a role below the user's clearance level. As mentioned earlier, the A_{ssa}-hierarchy relations allow a read-write role pair to be acquired at the given level. Osborn et al. provide such transformation for different variations of the BLP model by defining various constraints [14]. The GTRBAC hierarchies shown in Fig. 6 provide a more straightforward representation.

Example 5.3. Suppose that a doctor can assume either DayDoctor or NightDoctor role on a given day, but not both. Consider the *strong* SSoD:

$$((([1.1.2003, \infty], WorkingDaysOfWeek), SSoD_S$$
$$(\{\text{DayDoctor}, \text{NightDoctor}\}, "Smith")).$$

According to this condition, starting on 1.1.2003, the $SSoD_W$ constraint applies every five working days of a week. In other words, for a particular week, if Dr. Smith is

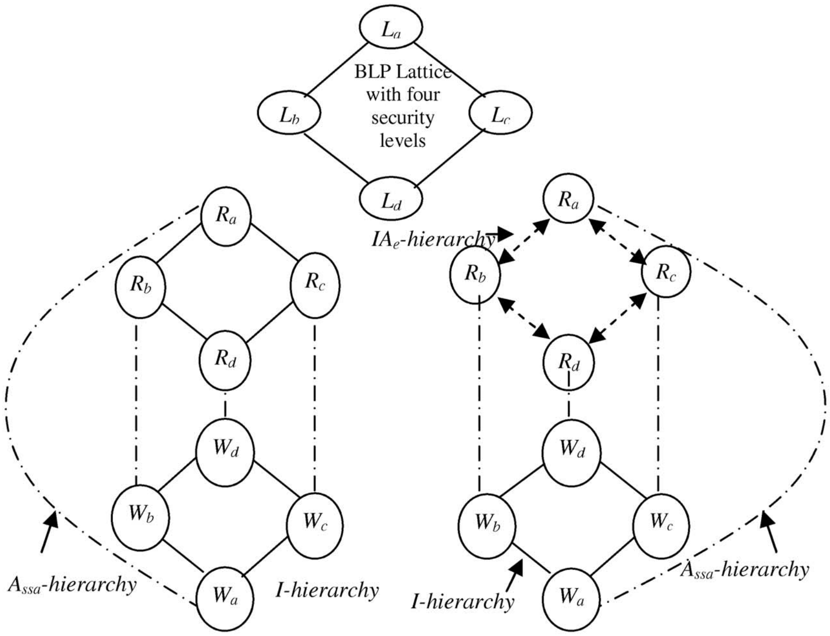

Fig. 6. Examples of A_{ssa} and IA_e-hierarchies.

assigned to role DayDoctor, he cannot be assigned to the NightDoctor role on any of the working days in that week. Next, consider the *extended strong SSoD*: (([1.1.2003, ∞], *WorkingDaysOfWeek*, $SSoD_{ES}$ ({DayDoctor, NightDoctor}, "*Smith*")). It implies that Dr. Smith is assigned to only one role for all the working days after 1.1.2003.

5.4 Safety of GTRBAC with Temporal Hierarchies and SoD Constraints

The addition of SoD constraints and temporal hierarchies to the list of constraints in Table 1 requires extending the notion of blocked events and TCAB safety as they introduce new scenarios in which events may be blocked or unsafe conditions may occur.

In particular, in order to enforce specified SoD constraints, certain events may need to be blocked. Ahn et al. show that both SSoD and DSoD constraints can be expressed as cardinality constraints with respect to given user and role sets [1]. For example, given a conflicting role set R and a user u, the DSoD implies that the number of roles from set R that user u can activate at a particular time is restricted to one. Thus, by using a condition similar to condition C4 of Definition 4.2.3 associated with the activation cardinality constraint, the events added to $\texttt{Caused}(t, EV, ST, \Gamma, RQ)$ in the presence of the SoD constraints can be easily expressed.

It can be noted that only the addition of A_{ssa}-hierarchy needs to be evaluated with respect to the safeness of Γ. For example, algorithm `SafetyCheck` can detect unsafe situations such as the presence of a trigger pair $(\texttt{enable}\ x \rightarrow pr : E; \quad pr : E \rightarrow \texttt{disable}\ x)$ in Γ. However,

$$\Gamma = \{\texttt{activate}\ x\ \texttt{for}\ u \rightarrow pr : E; \\ pr : E \rightarrow s : \texttt{deactivate}\ y\ \texttt{for}\ u\}$$

is considered safe by algorithm `SafetyCheck` as the events in triggers are of different categories for which there is no conflict. However, if we add A_{ssa}-hierarchy between roles x and y, i.e., if

$$\Gamma = \{\texttt{activatio}\ x \rightarrow pr : E; \\ pr : E \rightarrow s : \texttt{deactivate}\ y\ \texttt{for}\ u, (x \succeq_{ssa,t} y)\},$$

then Γ becomes unsafe. To illustrate this point, suppose that initially

$$EV(t) = \{s : \texttt{activate}\ x\ \texttt{for}\ u, s : \texttt{activate}\ y\ \texttt{for}\ u\}.$$

As the events are not blocked, the pair of triggers in Γ generate

$$EV(t) = \{s : \texttt{activate}\ x\ \texttt{for}\ u, s : \texttt{activate}\ y\ \texttt{for}\ u, \\ s : \texttt{deactivate}\ y\ for\ u, pr : E\}.$$

Note, event "s:`activate` y `for` u" is now blocked by the event "s:`deactivate` y for u," resulting in

$$\texttt{Nonblocked}(EV(t)) = \{s : \texttt{activate}\ x\ \texttt{for}\ u, \\ s : \texttt{deactivate}\ y\ for\ u, pr : E\}.$$

As A_{ssa}-hierarchy requires that both the roles x and y be active simultaneously in a session, the hierarchy constraint will block the event "s:`activate` x `for` u." Hence, event "s:`activate` x `for` u" causes event "s:`deactivate` y `for` u" which blocks the former event. Conflicting events due to A_{ssa}-hierarchy are shown in Table 9. Note that these events are essentially type 2, as the conflicting events are of different categories. Algorithm `SafetyCheck` needs to be extended to include a check for cycles containing events E_1 and E_2 with label "-." Note that these conflicting scenarios are introduced because an A_{ssa}-hierarchy, in addition to the role-activation semantics, defines a session-based constraint. Except for the A_{ssa}, A_e, and IA_e-hierarchies, the other hierarchies define only the permission-inheritance and role-activation semantics and, hence, they do not

TABLE 9
Conflicts Associated with A_{ssa}-Hierarchy

E_1	E_2=Conf(E_1)	*Condition*
s:activate x for u	de-activate y for u	$(x \geqslant_{ssa,t} y)$
s:activate x for u	disable y	$(x \geqslant_{ssa,t} y)$
s:activate x for u	s:de-assign y to u	$(x \geqslant_{ssa,t} y)$

introduce such conflicting scenarios. Although A_e and IA_e-hierarchies are constraints, they do not create any unsafe conditions. If $(x \succeq_{e,t} y)$ is present in Γ and events "s:activate x for u" and "s:activate y for u" both occur, one of the events is blocked. Furthermore, an activation event cannot cause another activation event to occur that blocks the former event because a trigger head cannot contain an activation event. Hence, no unsafe condition is introduced.

6 Related Work

The need for supporting constraints in an RBAC model has been addressed by many researchers. In particular, the attention has been focused on supporting *separation of duties* (SoD) constraints [1], [6], [11], [13], [15], [18]. Ferrariolo et al. [8] propose an RBAC model that supports the cardinality constraints. Sandhu et al. present a framework of four RBAC models [16]. In [1], Ahn et al. propose *RCL2000*—a role-based constraint specification language. Bertino et al. have proposed a logic-based constraint specification language that can be used to specify constraint on roles and users and their assignments to workflow tasks [6]. However, none of these models address temporal constraints. Bacon et al. have proposed the OASIS model for active security and have addressed some context-based access control requirements of large-scale systems [4]. It allows evaluation of dynamic user credentials and context conditions and uses preconditions to capture dependencies. The OASIS model, however, does not address temporal constraints and simply assumes that an implicit support for capturing events is available in the implementation platform. GTRBAC triggers provide a more general framework for capturing timing context and system events. With the additions of predicates to capture context information, all the functionalities provided by the OASIS model can be easily captured. The TRBAC model proposed by Bertino et al. [7] is the first known extension to an RBAC model that addresses the temporal constraints. Bertino et al. propose time-based access control model in [5] that supports temporal authorization and derivation rules in a non-RBAC environment. Atluri et al. [3] propose a *Temporal Data Authorization Model* (TDAM) that can express access control policies based on the temporal characteristic of data, such as valid and transaction time. Furthermore, TDAM does not support constraints on roles. Hence, temporal constraints that can be expressed in TDAM are different from those that can be expressed in the proposed GTRBAC model. The GTRBAC model can capture temporal characteristic of data only at the level of permission by using time-constrained role-permission assignments and triggers only. TDAM can, hence, augment the capabilities of the GTRBAC model. Unlike TDAM, GTRBAC also captures temporal characteristics of users and system/organizational functions represented by roles. Work related to hierarchies and separation of duty constraints can be found in [11], [13], [15], [16], [17], [18]. To the best of our knowledge, hierarchies and separation of duty constraints with temporal semantics have not been addressed in the literature.

7 Conclusions

We have proposed a generalized temporal role-based access control model that allows specification of a comprehensive set of temporal constraints. In particular, constraints on role enabling and activation and various temporal restrictions on user-role and role-permission assignments can be specified through the GTRBAC model. We have also presented time-based semantics of hierarchies and SoD constraints. A notion of safeness has been introduced to generate a safe execution model for a GTRBAC system. Although, overlapping intervals along the line of temporal work by Allen with regards to various entities of RBAC [2] have not been discussed, the semantics of overlapping intervals are elaborated for temporal hierarchies. The interval constraints along the line of work in [2] can be considered as dependency constraints where temporal intervals associated with a role are dependent on the intervals associated with some other roles. Depending upon the type of system deploying the proposed model, further extensions to the semantics of the constraints in the model may be needed. For example, in transaction/workflow types of systems, one crucial issue is to determine the timing constraints related to the execution of a transaction. A relevant question is what happens if a user's role is suddenly disabled by some event while the user is in the middle of executing a transaction permitted by a user-role assignment. Should the user's transaction be terminated at that moment or should it be allowed to complete? We leave such application specific issues for future work.

Acknowledgments

This research was supported by CERIAS (Purdue University) and through a US National Science Foundation grant IIS-0209111.

References

[1] G. Ahn and R. Sandhu, "Role-Based Authorization Constraints Specification," *ACM Trans. Information and System Security,* vol. 3, no. 4, Nov. 2000.
[2] J.F. Allen, "Maintaining Knowledge about Temporal Intervals," *Comm. ACM,* vol. 26, no. 11, pp. 832-843, Nov. 1983.
[3] V. Atluri and A. Gal, "An Authorization Model for Temporal and Derived Data: Securing Information Portals," *ACM Trans. Information and System Security,* vol. 5, no. 1, pp. 62-94, Feb. 2002.

[4] J. Bacon, K. Moody, and W. Yao, "A Model of OASIS Role-Based Access Control and Its Support for Active Security," *ACM Trans. Information and System Security,* vol. 5, no. 4, Nov. 2002.

[5] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning," *ACM Trans. Database Systems,* vol. 23, no. 3, pp. 231-285, Sept. 1998.

[6] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Trans. Information and System Security,* vol. 2, no. 1, pp. 65-104, Feb. 1999.

[7] E. Bertino, P.A. Bonatti, and E. Ferrari, "TRBAC: A Temporal Role-Based Access Control Model," *ACM Trans. Information and System Security*, vol. 4, no. 3, pp. 191-233, Aug. 2001.

[8] D. Ferrariolo, J.F. Barkley, and D.R. Kuhn, A Role-Based Access" Control Model and Reference Implementation within a Corporate Intranet," *ACM Trans. Information and System Security,* vol. 2, no. 1, pp. 34-64, 1999.

[9] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Trans. Information and System Security*, vol. 4, no. 3, pp. 224-274, Aug. 2001.

[10] J.B.D. Joshi, "A Generalized Temporal Role Based Access Control Model for Developing Secure Systems," PhD thesis, CERIAS TR 2003-23, Purdue Univ., 2003.

[11] D.R. Kuhn, "Mutual Exclusion of Roles as a Means of Implementing Separation of Duties in a Role-Based Access Control System," *ACM Trans. Information and System Security,* vol. 2, no. 2, pp. 177-228, 1999.

[12] M. Niezette and J. Stevenne, "An Efficient Symbolic Representation of Periodic Time," *Proc. First Int'l Conf. Information and Knowledge Management*, 1992.

[13] M. Nyanchama and S. Osborn, "The Role Graph Model and Conflict of Interest," *ACM Trans. Information and System Security,* vol. 2, no. 1, pp. 3-33, 1999.

[14] S.L. Osborn, R. Sandhu, and Q. Munawer, "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM Trans. Information and System Security,* vol. 3, no. 2, Feb. 2000.

[15] R. Sandhu, "Separation of Duties in Computerized Information Systems," *Database Security IV: Status and Prospects,* pp. 179-189, North Holland, 1991.

[16] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *Computer,* vol. 29, no. 2, Feb. 1996.

[17] R. Sandhu, "Role Activation Hierarchies," *Proc. Second ACM Workshop Role-Based Access Control,* 1998.

[18] R. Simon and M.E. Zurko, "Separation of Duty in Role-Based Environments," *Proc. 10th IEEE Computer Security Foundations Workshop,* June 1997.

James B.D. Joshi received the BE degree in computer science and engineering from Motilal Nehru Regional Engineering College, Allahabad, India, in 1993, the MS degree in computer science from Purdue University in 1998, and the PhD degree in computer engineering from the School of Electrical and Computer Engineering at Purdue University in 2003. He is an assistant professor in the Department of Information Sciences and Telecommunications, School of Information Sciences, at the University of Pittsburgh. Dr. Joshi is a coordinator of the Laboratory of Education and Research in Security Assured Information Systems (LERSAIS) at the University of Pittsburgh. From 1993 to 1996, he was a lecturer of computer science and Engineering at Kathmandu University, Nepal. His research interests are information systems security, database security, distributed systems and multimedia systems. He is a member of the ACM, the IEEE, and the IEEE Computer Society. He has served as a program committee member of the Eighth and Ninth ACM Symposium of Access Control Models and Technologies (SACMAT 2003, SACMAT 2004). He was a program cochair of the IEEE Workshop on Information Assurance held in conjunction with the 23rd IEEE International Performance Computing and Communications Conference (IPCCC).

Elisa Bertino is a professor in the Department of Computer Sciences at Purdue University and research director at CERIAS. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation, at Rutgers University, and at Telcordia Technologies. Her main research interests include security, privacy, database systems, object-oriented technology, and multimedia systems. In those areas, Dr. Bertino has published more than 250 papers in all major refereed journals and in proceedings of international conferences and symposia. She is a coauthor of the books *Object-Oriented Database Systems—Concepts and Architectures* (Addison-Wesley International Publishers, 1993), *Indexing Techniques for Advanced Database Systems* (Kluwer Academic Publishers, 1997), and *Intelligent Database Systems* (Addison-Wesley International Publishers, 2001). She is a coeditor in chief of the *Very Large Database Systems (VLDB) Journal* and a member of the advisory board of the *IEEE Transactions on Knowledge and Data Engineering*. She serves also on the editorial boards of several scientific journals, incuding *IEEE Internet Computing*, *ACM Transactions on Information and System Security*, *Acta Informatica*, *the Parallel and Distributed Database Journal*, *the Journal of Computer Security*, *Data and Knowledge Engineering*, *the International Journal of Cooperative Information Systems*, and *Science of Computer Programming*. She has been a consultant to several Italian companies on data management systems and applications and has given several courses to industries. She is involved in several projects sponsored by the EU. Dr. Bertino is a fellow of the IEEE and the ACM, and has been named a Golden Core Member for her service to the IEEE Computer Society. She has been a program committee member of several international conferences, such as ACM SIGMOD, VLDB, ACM OOPSLA, as program cochair of the 1998 IEEE International Conference on Data Engineering (ICDE), as program chair of 2000 European Conference on Object-Oriented Programming (ECOOP 2000), and as program chair of the Seventh ACM Symposium of Access Control Models and Technologies (SACMAT 2002). She is currently serving as program chair of the 2004 EDBT Conference.

Usman Latif's bio and photo are not available.

Arif Ghafoor is a professor in the School of Electrical and Computer Engineering, at Purdue University, West Lafayette, Indiana, and is the director of the Distributed Multimedia Systems Laboratory. He has been actively engaged in research areas related to database security, parallel and distributed computing, and multimedia information systems. He has published numerous technical papers in leading journals and conferences. Dr. Ghafoor has served on the editorial boards of various journals, including *ACM/Springer Multimedia Systems Journal*, the *Journal of Parallel and Distributed Databases*, and the *International Journal on Computer Networks*. He has served as a guest/coguest editor for various special issues of numerous journals, including *ACM/Springer Multimedia Systems Journal*, the *Journal of Parallel and Distributed Computing*, the *International Journal on Multimedia Tools and Applications*, the *IEEE Journal on the Selected Areas in Communications*, and *IEEE Transactions on Knowledge and Data Engineering*. He has coedited a book entitled *Multimedia Document Systems in Perspectives* and has coauthored a book entitled *Semantic Models for Multimedia Database Searching and Browsing* (Kluwer Academic Publishers, 2000). Dr. Ghafoor is a fellow of the IEEE. He has received the IEEE Computer Society 2000 Technical Achievement Award.