Fine-grained Role-based Delegation in Presence of the Hybrid Role Hierarchy

James B. D. Joshi
LERSAIS & Department of Information Science and Telecommunications, University of Pittsburgh
Pittsburgh, PA 15212
jjoshi@mail.sis.pitt.edu

Elisa Bertino
CERIAS and Department of Computer Science, Purdue University
West Lafayatte, IN 47907
bertino@cs.purdue.edu

ABSTRACT
Delegation of authority is an important process that needs to be captured by any access control model. In role-based access control models, delegation of authority involves delegating roles that a user can assume or the set of permissions that he can acquire, to other users. Several role-based delegation models have been proposed in the literature. However, these models consider delegation in presence of the general hierarchy type. Multiple hierarchy types have been proposed in the context of Generalized Temporal Role-based Access Control (GTRBAC) model, where it has been shown that multiple hierarchy semantics is desirable to express fine-grained access control policies. In this paper, we address role-based delegation schemes in the of hybrid hierarchies and elaborate on fine-grained delegation schemes. In particular, we show that upward delegation, which has been considered as having no practical use, is a desirable feature. Furthermore, we show that accountability must be considered as an important factor during the delegation process. The delegation framework proposed subsumes delegations schemes proposed in earlier role-based delegation models and provide much more fine-grained control of delegation semantics.

Categories and Subject Descriptors
D.4.6 [Security and Protection]: Access controls; H2.7 [Database Administration]: Security, integrity, and protection.

General Terms
Security, Management, Theory.

Keywords
Role based, access control, delegation, hybrid hierarchy

1. INTRODUCTION
Role based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access control (DAC and MAC) models [Jos01b, Osb00, San96a, Giu97], which have some inherent limitations. Several beneficial features make RBAC better suited for handling access control requirements of diverse organizations [Jos01b, San96a, Giu97]. One important organizational process that affects the access control privilege distribution among the users is *delegation*. Delegation involves a subject passing its authority to other subjects. Zhang *et al.* [Zha03a] identify three cases in which delegation takes place. The first case, termed as *backup of role*, corresponds to when someone is not in a position to perform the tasks that he is supposed to do. In such a case, he should be allowed to have someone else perform his job functions by delegating his authority to do the job to someone else. Secondly, delegation may be used to achieve *decentralization of authority*. Lastly, delegation is useful when individuals *collaborate* on some work – they may need to delegate their authorities to ease the collaboration.

Existing role-based delegation models address delegation in presence of the traditional single hierarchy type. Recently, Joshi *et al* [Jos05] have identified three different types of hierarchical relations within the Generalized Temporal RBAC (GTRBAC) framework [Jos05] that can be applied between roles, namely *inheritance*-only hierarchy (*I*-hierarchy), *activation*-only hierarchy (*A*-hierarchy) and *inheritance*-and-*activation* hierarchy (*IA*-hierarchy). A hybrid hierarchy in which these different hierarchy types coexist can capture fine-grained inheritance semantics [Jos05 Jos02a, Jos03, San98]. In particular, when various separation of duty (SoD) as well as user-centric and permission centric cardinality constraints need to be applied on roles in a hierarchy, *A*-hierarchy can be used [San98, Jos02a]. The use of these hierarchy types also facilitate efficient integration of multiple RBAC policies employing hierarchical as well as SoD constraints [Sha03].

In presence of the multiple hierarchy types, more fine-grained delegation semantics with practical applications is possible. In this paper, we address delegation in presence of the hybrid hierarchy within the GTRBAC framework. Because of space limitation, we do not discuss *cross-sectional* delegation that does not involve hierarchies. The novelty of the paper is as follows:

- We provide a more complete delegation framework that subsumes all role-based delegation schemes proposed earlier in the literature. In addition, the proposed delegation framework also allows delegating or preventing delegation of dynamically assigned permissions during the delegation period. We introduce a novel concept of *filter* roles to support this feature, which, to the best of our knowledge, has not been addressed earlier.

SACMAT'06, June 7-9, 2006, Lake Tahoe, CA, USA.

- The proposed delegation framework introduces upward delegation that allows a user assigned to a junior role to delegate his authority to a user assigned to a senior role. Contrary to earlier belief, we show that upward delegation is practically relevant, particularly when hybrid hierarchy is used. Further, we argue that upward delegation is also important to enhance support for accountability.

While different combinations of delegation and revocation schemes are possible, because of the space limitation, we focus on single step delegation and revocation only. For such simple cases, the revocation details are straightforward and hence, for the sake of space, are presented in the tables and only briefly discussed.

The paper is organized as follows. We overview related work in Section 2 and the GTRBAC model in Section 3. We present the proposed delegation framework in Section 4 and conclusions in Section 5.

2. RELATED WORK

Considerable work on different aspects of delegation has been reported in the literature. Gasser *et al*. address user-to-machine delegation [Gas90], while Stein explores delegation and inheritance in the object-oriented environment [Ste87]. Process to process delegation in distributed object environment has been introduced by Nagaratnam *et al*. [Nag98]. Sandhu *et al.* address delegation related to role administrators in ARBAC97 model [San99]. Goh *et al.* treat delegation as an attribute of role [Goh98]. RBDM0 allows user-to-user delegation based on roles [Bar00a, Bar00b]. That is, a user (*delegator*) assigned to a role (*delegator role*) delegates his role membership to another user (*delegatee*) assigned to another role (*delegatee role*). In RBDM0, a role is delegated entirely, *i.e*., all the permissions associated with the delegated roles are available to the user to whom the role has been delegated. RDM2000 extends RBDM0 and supports delegation in presence of a role hierarchy and multi-step delegation. RDM2000 provides a rule based declarative language to specify and enforce delegation and revocation policies. It introduces the *can_delegate* condition with prerequisite roles to restrict the scope of delegation. In both RBDM0 and RDM2000, the unit of delegation is a role. PBDM is a family of models that extend RDM2000 with newer features. PBDM0, the first model of PBDM, supports permission level user-to-user delegation, i.e., a subset of permissions is allowed to be delegated. In particular, a new "*delegation*" role is created with the set of permissions to be delegated explicitly assigned to it. PBDM1 extends PBDM0 by allowing the security administrator to control the delegated permissions. Here, the delegated role, which is a replica of an original role is created by the security administrator in order to control the flow of the delegated permissions. PBDM2 extends PBDM0 by allowing role-to-role delegation. Delegation based on conditions on time, workload and other task attributes have been considered in [Atl05]. Wainer *et al*. focus on user to user delegation in [Wai05]. Some work, e.g., [Tho97], have dealt with adding permissions to a session dynamically to facilitate collaborative work.

The framework we propose in this paper is closely related to the RBDM0 [Bar00a, Bar00b], RDM2000 [Zha03b] and PBDM [Zha03a] models and extends the features they provide with more fine grained delegation semantics aligned with the fine-grained semantics of hybrid hierarchies [Jos02a].

3. HYBRID HIERARCHY IN GTRBAC

The GTRBAC model introduces the separate notion of role enabling and role activation, and provides constraints and event expressions associated with both [Jos05]. An *enabled* role indicates that a valid user can activate it, whereas an *activated* role indicates that at the least one user has activated the role. The basic GTRBAC model proposed in [Jos05], allows specification of the following set of constraints: (*i*) *temporal constraints on role enabling/disabling* that allow specification of intervals and durations in which a role is enabled; (*ii*) *temporal constraints on user-role and role-permission assignments* that allow specifying intervals and durations in which a user or permission is assigned to a role; (*iii*) *activation constraints* that allow specification of restrictions on the activation of a role, such as, specifying the total duration for which a user may activate a role, or the number of concurrent activations of the role at a particular time; (*iv*) *run-time events* allow an administrator and users to dynamically initiate the various role events, or enable the duration or activation constraints; (*v*) *constraint enabling* events that enable or disable duration and role activation constraints mentioned earlier; and (*vi*) *triggers* that allow expressing dependencies among events and conditions

Semantically, a role hierarchy expands the scope of the permission-acquisition and role-activation semantics beyond the explicit assignments through the hierarchical relations defined among roles. Within the GTRBAC framework, the following three hierarchy types have been identified: *permission-inheritance-only* hierarchy (*I*-hierarchy), *role-activation-only* hierarchy (*A*-hierarchy) and the combined *inheritance-activation* hierarchy (*IA*-hierarchy) [Jos05]. Table 3.1 captures the predicate notations used in defining the semantics of these hierarchies [Jos05]. Predicates *enabled*(*r, t*), *assigned*(*u, r, t*) and *assigned*(*p, r, t*) refer to the status of roles, user-role and role-permission assignments at time *t*. Predicate *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*. *active*(*u, r, s, t*) indicates that role *r* is active in user *u*'s session *s* at time *t* whereas, *acquires*(*u, p, s, t*) implies that *u* acquires permission *p* at time *t* in session *s*. The axioms in Table 3.1 capture the key relationships among these predicates and identify precisely the permission-acquisition and role-activation semantics in GTRBAC [Jos05]. Axiom (1) states that if a permission is assigned to a role, then it *can be acquired* through that role. Axiom (2) states that all users assigned to a role *can activate* that role. Axiom (3) states that if a user *u can activate* a role *r*, then all the permissions that *can be acquired* through *r can be acquired* by *u*. 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 explicit user-role and role-permission assignments.

In Table 3.2, the semantics of each hierarchy type is defined by its corresponding implication rule in the shaded box. The rule for the *I*-hierarchy, ($x \geq_I y$), implies that if ($x \geq_I y$) holds, then the permissions that can be acquired through role *x* include all the permissions that *can be acquired through* role *y*. In other words, permissions of the junior roles are inherited by the senior role. Similarly, the rule for the *A*-hierarchy implies that if user *u* can *activate* role *x*, and $x \succeq_I y$ is defined, then user *u can* also *activate*

Table 3.1 Status predicates

Predicate	Meaning	Axioms
enabled(*r, t*)	*Role r is enabled at time t*	*For all r* ∈ Roles, *u* ∈ Users, *p* ∈ Permissions, *s* ∈ Sessions, *and time instant t* ≥ 0, *the following implications hold*:
u_assigned(*u, r, t*)	*User u is assigned to role r at time t*	
p_assigned(*p, r, t*)	*Permission p is assigned to role r at time t*	*1. assigned*(*p, r, t*)→ *can_be_acquired*(*p, r, t*)
can_activate (*u, r, t*)	*User u can activate role r at time t*	*2. assigned*(*u, r, t*) → *can_activate* (*u, r, t*)
can_acquire (*u, p, t*)	*User u can acquire permission p at time t*	*3. can_activate* (*u, r, t*) ∧ *can_be_acquired*(*p, r, t*) → *can_acquire* (*u, p,t*)
can_be_acquired(*p, r, t*)	*Permission p can be acquired through role r at time t*	
active(*u, r, s, t*)	*Role r is active in user u's session s at time t*	*4. active*(*u, r, s, t*) ∧ *can_be_acquired*(*p, r, t*) → *acquires*(*u, p, s, t*)
acquires(*u, p, s, t*)	*User u' acquires permission p in session s at time t*	

Table 3.2. Role hierarchies in GTRBAC

Short form	Notation	Formal Semantics
I-hierarchy	$(x \geq_t y)$	$\forall p, (x \geq_t y) \wedge can\_be\_acquired(p, y, t) \rightarrow can\_be\_acquired(p, x, t)$
A-hierarchy	$(x \succcurlyeq_t y)$	$\forall u, (x \succcurlyeq_t y) \wedge can\_activate(u, x, t) \rightarrow can\_activate(u, y, t)$
IA-hierarchy	$(x \gtrsim_t y)$	$(x \gtrsim_t y) \leftrightarrow (x \geq_t y) \wedge (x \succcurlyeq_t y)$
Consistency Property: *Let* $<f_1><f_2> \in \{\geq_t, \succcurlyeq_t, \gtrsim_t\}$, and *x and y be distinct roles such that* $(x <f_1> y)$ *then* $\neg(y <f_2> x)$ *must hold*		

Table 4.1 Notational Conventions and New GTRBAC

Terms/notation	Description
delegator	A user who delegates his authority
delegatee	A user who receives the delegated authority
delegator role	Role that the *delegatee* is assigned to
delegatee role	Role that the *delegatee* is assigned to
delegation role	A copy of *delegator role* that the delegatee is assigned to achieve delegation
delegation sub-role	A copy of role to which there is a hierarchical path from the *delegator role*.
$x <f> y, <f> \in \{\geq, \succcurlyeq, \gtrsim\}$	*x* is direct senior of *y* with hierarchy type <*f*>
$x <f>_g y$, $<f> \in \{\geq, \succcurlyeq, \gtrsim\}$	*y* is *direct* or *derived* junior of *x*

role *y* even if he is not explicitly assigned to *y*. Note that it does not imply that user *u* can acquire *y*'s permissions by merely activating *x*. In other words, permission-inheritance is not implied in an *A*-hierarchy. The *IA*-hierarchy is the most general form and includes both permission-inheritance and role-activation semantics. In the remaining sections we do not use the time parameter *t* in any expression.

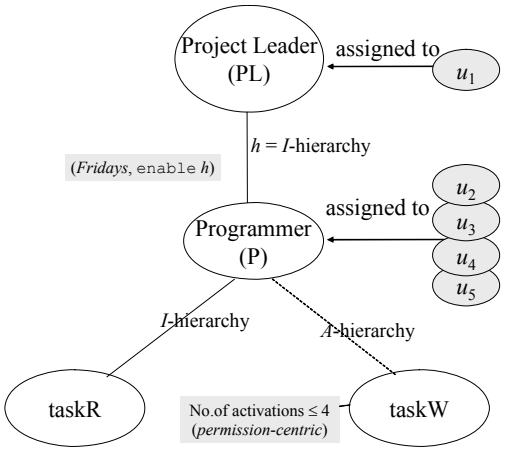

Figure 3.1 Hierarchy example

Example 3.1: Assume that a programming tool is used for a programming project and has a licensing restriction preventing more than four users using it at any given time. The project leader mainly supervises the programming tasks. Only the programmers do the coding. The project leader can only look at the tasks the programmers have carried out in a weekly basis, say on *Fridays*. Figure 3.1 depicts the hierarchy that can be generated for achieving the goal[1]. Role TaskR contains the *read*-only permissions whereas role TaskW contains all the write/modify permissions related to the programming task. The Project Leader role becomes the senior of Programmer role only on *Fridays*. Note that the users assigned to the Project Leader only inherit TaskR permissions and cannot acquire any permissions of TaskW.

4. GTRBAC DELEGATION FRAMEWORK

In this section, we extend the GTRBAC model to capture various delegation schemes in presence of different hierarchies. Table 4.1 introduces some terminologies that we use in this paper. In particular, *delegator*, *delegatee*, *delegator role, delegatee role* and *delegation role/sub-role* have specific meanings. If *x* is the *delegator role* then we use *x*' to represent the *delegation role*.

4.1 Role-based Delegation Schemes

Figure 4.1 shows the delegation schemes allowed in the proposed GTRBAC framework.

[1] *I, A* and *IA*-hierarchies are represented by a simple line, a dotted line and a line with arrows on both ends, respectively.

4.1.1 U2U, R2R , U2R, and R2U Delegation

Broadly speaking, in an RBAC model, delegation can be *user-to-user* (U2U), *user-to-role* (U2R), *role-to-role* (R2R) or *role-to-user* (R2U). In U2U delegation, a *delegator* delegates a *delegator role* or a part of it to the *delegatee* based on the relationship between the *delegator role* and the *delegatee role*. In R2R, the delegation is not user specific - the *delegator role* is delegated to the *delegatee role*. This means, anyone who can activate the *delegatee* role can also activate the *delegation* role. The key difference is, in U2U, a user primarily decides (although not always) who he wants to delegate his role to, while in R2R, a role is delegated to another role (hence to all its authorized users) by the system based on some pre-specified rule. R2R can be viewed as a special case of the U2U delegation in which the *delegator role* is delegated to all the users authorized for the *delegatee role*. U2R and R2U delegations can be considered as in the middle of the two extremes U2U and R2R. That is, in U2R, an individual user delegates his role to the *delegatee* role, while in R2U, the *delegator* role is delegated to an individual *delegatee*.

4.1.2 Downward, Cross-sectional and Upward Delegation

Each delegation scheme may further be categorized as *downward*, *cross-sectional* or *upward*. In *downward* and *upward* delegation schemes, the *delegator role* and the *delegatee role* are hierarchically related by direct or derived *I*, *A*, or *IA* relations. In *downward* delegation, the *delegator role* is a senior of the *delegatee role*, whereas, in the *upward* delegation, the *delegator role* is a junior of the *delegatee role*. In *cross-sectional* delegation, the *delegator role* and the *delegatee role* are not hierarchically related. Both *downward* and *cross-sectional* delegation schemes have been the main focus of earlier role-based delegation models. *Upward* delegation, however, has been dismissed as not being of any practical use [Zha03a]. In this paper, we introduce *upward* delegation primarily for two purposes

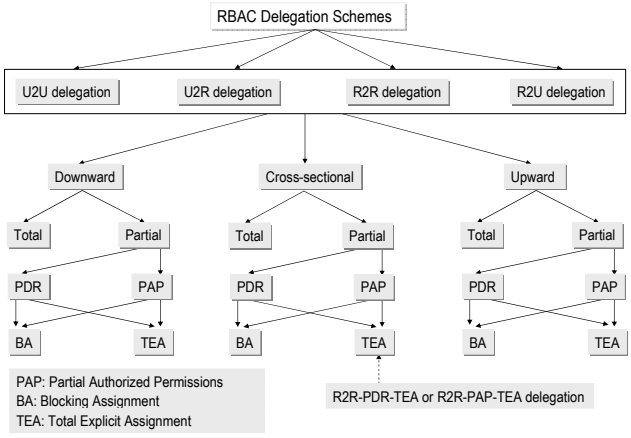

Figure 4.1 GTRBAC delegation schemes

(1) to facilitate fine-grained delegation in presence of the hybrid hierarchy and (2) to provide more fine-grained support for accountability of delegated authority.

Presence of hybrid hierarchy: In a hybrid hierarchy, a complex inheritance and activation semantics between an arbitrary pair of roles may exist. In such a case, the *upward* delegation comes as a desirable mechanism to transfer user's access authority to others who are assigned to higher roles. One such situation in hybrid hierarchies is when there is an *I*-relation followed by an *A*-relation from the senior *delegatee role* (say x) to the junior *delegator role* (say z), *i.e.*, when $x \geq_g y$ and $y \geqslant z$. In such a case, a user assigned to x cannot acquire any permission that can be acquired through role z. The user cannot activate y either, because of the *I*-relation. Hence, some form of delegation needs to be utilized if the junior *delegator role* is to be delegated to the *delegatee* assigned to the senior *delegatee role*. Example 4.2 illustrates such a case.

`Example 4.1`: Let us revisit example 3.1 and refer to Figure 4.2. Here, a user assigned to the PL role cannot acquire the permissions of the taskW role. Assume that one of the users, say John, assigned to P gets sick and his tasks need to be carried out by a user authorized for the PL role. The delegation involves giving the project leader the ability to activate taskW on John's behalf. This can be achieved in several ways for this particular RBAC hierarchy and set of assignments. Figure 4.2(b) shows one way in which the *delegation* roles P' and taskW' are created as *I*-seniors of P and taskW. PL is made *A*-senior of P'. Hence, anyone assigned to PL can activate both P' and taskW'. It could be an R2R delegation or a U2R delegation depending upon whether the system or John authorizes the delegation. Figure 4.2(c) shows the second case of U2U delegation (or R2U if the system rather than John is responsible for authorizing delegation), where user u_1 is explicitly assigned to the senior role P'. Figure 4.2(d) shows the *delegation role* P' assigned to the user u_1 (hence it is U2U or R2U delegation). Here, P' is assumed to have been explicitly assigned the set of permissions to be delegated that can be acquired through roles P and taskW.

The example shows that delegation in hybrid hierarchies becomes complex and more fine-grained control of delegated permission can be imposed. For instance, in figures (b) and (c), if we do not include taskW', the delegation would include only permissions authorized for P. We could also simply delegate taskW by not including P', and making taskW' an *A*-junior of PL in (b) or by assigning u_1 to taskW' in (d).

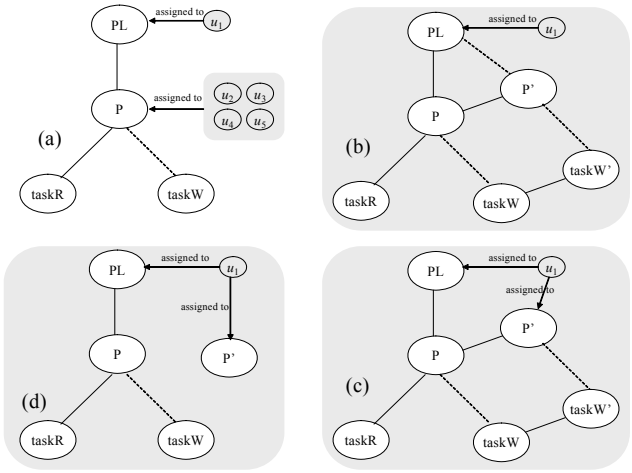

Figure 4.2 Upward delegation example

Accountability: As shown in example 4.1, it may be necessary for a user who is assigned to a junior role to delegate his authority to a user assigned to a senior role. However, as a *delegatee*, the project leader should be made accountable for any work that he does on behalf of the *delegator*. Note that as a user assigned to a

Table 4.2 *Can delegate* policy statement forms

***Can delegate* statement (*cd*)**		**Meaning**
cd_1	$can\_delegate(r_d, r_e)$	*Role r_d can be delegated to r_e* (**R2R**)
cd_2	$can\_delegate(r_d, u_e, r_e)$	*Role r_d can be delegated to user u_e assigned to role r_e* (**R2U**)
cd_3	$can\_delegate(u_d, r_d, u_e, r_e)$	*User u_d can delegate role r_d to user u_e assigned to role r_e* (**U2U**)
cd_4	$can\_delegate(u_d, P_d, u_e, r_e)$	*User u_d can delegate permission set P_d to user u_e assigned to role r_e if the following holds: $\forall\, p \in P_d$, $can\_acquire(u_d, P_d)$* (**U2U**)
cd_5	$can\_delegate(u_d, \overline{P_d}, u_e, r_e)$	*User u_d can delegate permission set P- $\overline{P_d}$ to user u_e assigned to role r_e if the following holds: $\overline{P_d} \subseteq P \wedge \forall p \in P$, $can\_acquire(u_d, P_d)$.* (**U2U**)

senior role, the *delegatee* has access to the *delegator* role prior to the delegation as well. However, after delegation, accountability becomes a key issue, as the *delegatee* will be exercising permission on behalf of the *delegator*. Key to achieving accountability is to record the session information that indicates roles that are active in a *delegatee*'s session and the permissions that have been acquired through the active roles. Accountability has not been considered as a factor in earlier delegation models.

4.1.3 Total and Partial Delegation

Each of the delegation schemes can further be categorized as *total delegation* (TD) or *partial delegation* (PD). In TD, unlike in PD, the entire authority that the *delegator role* embodies is delegated. That is, all the permissions that he can acquire (through assignments and inheritance) are made available to the *delegatee*. Such an approach will not be able to provide fine-grained control on delegation. In particular, the *delegator role* may represent authority to carry out several different tasks, particularly, if it is at the higher level of a role hierarchy. In such a case, the *delegatee* may simply want to delegate a part of the role that represents a particular subset of the delegated permissions. In earlier models, delegation schemes have only considered the permissions explicitly assigned to the *delegation role* instead of all the permissions that can be acquired through a role (using *I*-relations) or through the activation of authorized roles (using *A*-hierarchy). Our framework allows delegating the entire set of permissions that the *delegatee* can acquire by virtue of his membership to the *delegator* role.

Partial delegation may be enforced using different schemes. In our framework, we allow it in two ways: (1) *blocking assignment* (BA), and (2) *total explicit assignment* (TEA). In the BA scheme, permissions that are not to be delegated are blocked from being acquired by the *delegatee*. The TEA scheme, on the other hand, involves explicitly assigning the set of permissions that are to be delegated. Exisitng role-based delegation models use the TEA scheme for partial delegation and do not support the BA scheme. It is to be noted that the TEA scheme is static in nature and hence cannot be used in dynamic environments where role-permission assignments can continually change over time as is possible in the GTRBAC model. However, the TEA scheme is useful in cases where the *delegator* wants to ensure that in the delegation period, the *delegatee* does not have access rights that may be dynamically assigned/authorized to the *delegation role*. The advantage of the BA approach is that during the delegation period, if new permissions are available to the *delegation* role through new permission assignments to the *delegator* role or sub-roles, they will also be available to the *delegatee*. Furthermore, it also allows blocking sensitive permissions that exist prior to delegation or that may be available during the delegation period.

Table 4.3 Extended GTRBAC Events

Existing GTRBAC events	**New GTRBAC events**
`enable` *r*	[u_d]:`delegate` r_d `to` u_e
`disable` *r*	[u_d]:`revoke` r_d' `from` u_2
`assign`$_U$ *r* `to` *u*	[u_d]:`delegate` r_d `to` r_e
`deassign`$_U$ *r* `to` *u*	[u_d]:`revoke` r_d'
`assign`$_P$ *p* `to` *r*	`block_assign`$_P$ *p* `to` *r*
`deassign`$_P$ *p* `to` *r*	`block_deassign`$_P$ *p* `to` *r*
s: `activate` *r* `for` *u*	
s: `deactivate` *r* `for` *u*	

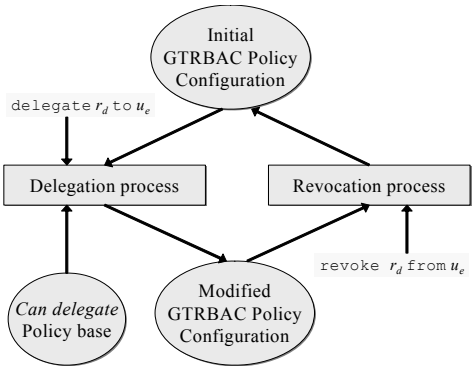

Figure 4.3 Delegation and revocation process

4.2 Delegation and revocation in presence of the *I*, *A* and *IA*-hierarchies

Here, we detail fine-grained delegation semantics for each of the hierarchy types mentioned earlier. We differentiate several cases based on the hierarchical relations between the *delegator role* and the *delegatee role*. For each delegation scheme, we present the revocation scheme that follows the reverse steps. We focus on single step delegation and revocation only. For such simple cases, the revocation details are straightforward. We assume that a *can delegate* policy is specified to indicate who can delegate or which role can be delegated to which entities (users or roles). Formally, a *can delegate* policy base is defined as follows:

Definition 4.1 (***can delegate*** policy): *A can delegate* policy base (CDPB) *is a set of expressions of the forms listed in* Table 4.2.

We extend the existing event set of GTRBAC with new events listed in Table 4.3 to facilitate the delegation process. Figure 4.3 depicts the general delegation process using an informal petri-net diagram. Delegation process is initiated by a delegation event provided there is a *can delegate* policy statement authorizing the event. For example, if the "u_d: `delegate` r_d `to` u_e" event occurs, the system has to first ensure that there a role r_e for which $cd_3 \in$ CDPB, $u\_assigned(u_d, r_d) = true$, and $u\_assigned(u_e, r_e) = true$. The delegation process includes creation of new roles and assignments as well as hierarchical structures to facilitate the intended delegation. The result hence is a new GTRBAC policy base. Given the modified policy base, a revoke event initiates the revocation process to undo the changes done by the delegation process. For our purpose, it involves removing newly created roles, new assignments and hierarchy structures to bring the policy back to the initial configuration. In actual implementation, it may simply be disabling the new entities created by the delegation process.

To support the blocking of a specified set of permissions from being delegated in BA schemes, we introduce *filter roles*, which are indicated by a line above the role name. A *filter* role is associated with a set of permissions, through what we call *blocking assignments* that are *not* to be made available through that role. In essence, a filter role may also have normal permission assignments. To define the *filter* roles, we introduce special blocking assignment event "$\texttt{block\_assign}_p$ p to $\bar{r}$" and an associated status predicate $block\_p\_assigned(p, \bar{r}, t)$ which implies that p is associated with role $\bar{r}$ through *blocking assignment* at time t. The semantics of the filter role is captured by the following definition:

Definition 4.2 (*filter role*): *A filter role* $\bar{r}$ *is a special role for which the following rules apply*:

- $block\_p\_assigned(p, \bar{r}, t) \rightarrow \neg\, can\_be\_acquired(p, \bar{r}, t)$ (**axiom 5**)
- $\forall p, (x \geq \bar{r}) \wedge can\_be\_acquired(p, \bar{r}, t) \wedge \neg block\_p\_assigned(p, \bar{r}, t) \rightarrow can\_be\_acquired(p, x, t)$ (*where y is a normal or a filter role*) (**new definition of *I*-hierarchy**)

Furthermore, we assume that for all normal permission-role assignments for which p_assigned(*p, r, t*) *is* true, *block_p_assigned*(*p, r, t*) *is* false.

We use the first condition in the definition as a new axiom. The effect of axiom 5 is that permissions associated with the filter role through the *blocking assignment* cannot be acquired in a session. This is because, axiom 4 states that only those permissions that can be acquired through a role are acquired by a user who has activated that role – and axiom 5 essentially ensures that the blocking-assigned permissions cannot be acquired through that role. This condition, however, does not prevent the *blocking*-assigned permissions from being acquired through a senior role, if the permissions that are to be blocked are inherited by the filter role from other roles. The second condition blocks the permission acquisition through hierarchy relation. This ensures that only permissions that can be acquired through a role and not blocked by it can be acquired through its *I*-senior role. We use the second condition as the new definition of the *I*-hierarchy as it now allows filter roles to be a part of the role hierarchies. We use a filter role to support the BA scheme.

4.2.1 Downward Delegation in I-hierarchy

Figure 4.4 depicts the delegation schemes when the *delegator role* r_d is *I*-senior of the *delegatee role* r_e *i.e.,* ($r_d \geq_g r_e$). Table 4.4 shows the delegation processes for each of them. The *delegatee role* may be related to junior roles through any of the three hierarchy types. To generalize the construction, the hierarchy includes three juniors (roles *a*, *b* and *c*) each related to the *delegatee role* with one hierarchy type.

***Total delegation (TD)*:** Here, the *delegatee* needs to totally delegate the *delegator role* to the *delegatee*. As shown in Figure 4.4(b), the *delegation role* r_d' is created and the *delegatee* is assigned to it. r_d' is an empty role which is made *I*-senior of the *delegator role*. The *delegatee* can now activate r_d'and acquire all the permissions that can be acquired through role r_d because of the *I*-relation. However, the *delegatee* cannot activate r_d and any roles junior to r_d. Accountability is easily facilitated if we maintain an audit log of the sessions in which the *delegation role* is activated by the *delegatee*. Note that any of the permissions that can be acquired through the *delegatee role (including those of* r_e as as ($r_d \geq_g r_e$) may be used by the *delegatee*.

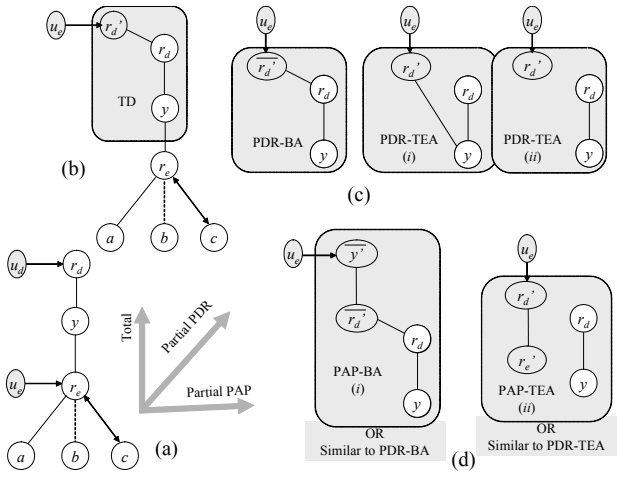

Figure 4.4 Downward delegation in *I*-hierarchy

Partial Delegator Role (PDR): As indicated earlier, this scheme allows a subset of permissions explicitly assigned to the *delegator role* to be delegated. Inherited permissions are, however, passed on in totality. This can be achieved in two different ways as shown in Figure 4.4 (c). The first scheme, PDR-BA, involves specifying the permissions $\bar{P}_d$ of r_d that are not to be delegated. As shown in the table, the delegation role is $\bar{r}_d'$ a filter role to which $\bar{P}_d$ *is blocking assigned.* However, other permissions can be acquired through $\bar{r}_d'$ because of the *I*-inheritance and are hence delegated to the *delegatee*. The second scheme, PDR-TEA can be implemented in two ways. In TDR-TEA(*i*), only the permissions assigned to r_d that are to be delegated are assigned to r_d', which is made senior of r_d's junior role so that the permissions below the hierarchy are inherited. In TDR-TEA(*ii*), however, all the

Table 4.4. Semantics for the downward delegation in I-hierarchy[2]

Scheme	*Delegation process*	*Revocation Process*
TD $cd_3 \in$ CDPB; *u_assigned*(u_d, r_d) = true *u_assigned*(u_e, r_e) = true	`delegate` r_d `to` u_e	`revoke` r_d' `from` u_e
	(1) `create` r_d'; (2) `add` ($r_d' \geq r_d$) `to` H; (3) `assign`$_U$ r_d' `to` u_e	(1) `deassign`$_U$ r_d' `to` u_e (2) `remove` ($r_d' \geq r_d$) `from` H; (3) `delete` r_d'
PDR-BA $cd_5 \in$ CDPB; *u_assigned*(u_d, r_d) = true *u_assigned*(u_e, r_e) = true	(1) `Create` $\overline{r_d'}$ (2) $\forall p \in \overline{P_d}$, `block_assign`$_P$ p `to` $\overline{r_d'}$ (3) $\forall r$, *s.t.* ($r_d \geq r$) $\in H$, `add` ($\overline{r_d'} \geq r$) `to` H (4) `assign`$_U$ $\overline{r_d'}$ `to` u_e	(1) `deassign`$_U$ $\overline{r_d'}$ `to` u_e (2) $\forall r$, *s.t.* ($r_d' \geq r$) $\in H$, `remove` ($\overline{r_d'} \geq r$) `from` H (3) `delete` $\overline{r_d'}$
PDR-TEA(*i*) $Cd_4 \in$ CDPB; *u_assigned*(u_e, r_e) = true	(1) `create` r_d' (2) $\forall p \in P_d$, `assign`$_P$ p `to` r_d' (3) $\forall r$, *s.t.* ($r_d \geq r$) $\in H$, `add` ($r_d' \geq r$) `to` H (4) `assign`$_U$ r_d' `to` u_e	(1) `deassign`$_U$ x' `to` u_2 (2) $\forall r$, *s.t.* ($x' \geq r$) $\in H$, `remove` ($x' \geq r$) `from` H (3) $\forall p \in$ *assigned(p, x')*, `deassign`$_P$ p `to` x' (4) `delete` x'
PDR-TEA (*ii*) $Cd_4 \in$ CDPB; *u_assigned*(u_e, r_e) = true	(1) `create` r_d' (2) $\forall p \in P_d$, `assign`$_P$ p `to` r_d' (3) `assign`$_U$ r_d' `to` u_e	(1) `deassign`$_U$ r_d' `to` u_e (2) $\forall p \in$ *assigned* (p, r_d'), `deassign`$_P$ p `to` r_d' (3) `delete` r_d'
PAP-BA $cd_5 \in$ CDPB; *u_assigned*(u_d, r_d) = true *u_assigned*(u_e, r_e) = true *X': new filter delegation roles/sub-roles created* (*Or similar to* ***PDR-BA***)	(1) `Create` $\overline{r_d'}$ (2) $\forall p \in \overline{P_d}$, *assigned*($p$, r_d) = true, `block_assign`$_P$ p `to` $\overline{r_d'}$ (2) $\forall r$, *s.t.* ($r_d \geq r$) $\in H$, $\exists p \in \overline{P_d}$, *assigned*($p$, r) = true (a) `create` $\overline{r'}$; (b) `add` ($\overline{r'} \geq \overline{r_d'}$) `to` H; (c) `block_assign`$_P$ p `to` $\overline{r'}$ (3) $\forall r_1, r_2$, *s.t.* ($r_d \geq_g r_1$) and ($r_1 \geq r_2$) $\in H$, $\exists p \in \overline{P_d}$, *assigned*($p$, r_2) = true (a) `create` $\overline{r_2'}$; (b) `add` ($\overline{r_2'} \geq \overline{r_1'}$) `to` H (c) `block_assign`$_P$ p `to` $\overline{r_2'}$ (4) `assign`$_U$ $\overline{r_d'}$ `to` u_e	(1) `deassign`$_U$ $\overline{r_d'}$ `to` u_e (2) $\forall \overline{r_1'}, \overline{r_2'} \in X'$, *s.t.* ($\overline{r_1'} \geq \overline{r_2'}$) $\in H$, (a) `remove` ($\overline{r_1'} \geq \overline{r_2'}$) `from` H (b) `delete` $\overline{r_1'}$ (c) `delete` $\overline{r_2'}$
PAP-TEA $Cd_4 \in$ CDPB; *u_assigned*(u_e, r_e) = true *X': new delegation roles/sub-roles created* (*Or similar to* ***PDR-TEA***)	(1) `create` r_d' (2) $\forall r$, *s.t.* ($r_d \geq r$) $\in H$, $\exists p \in P_d$, *assigned*(p, r) = true (a) `create` r'; (b) `add` ($r_d' \geq r'$) `to` H (3) $\forall r_1, r_2$, *s.t.* ($r_d \geq_g r_1$) and ($r_1 \geq r_2$) $\in H$, $\exists p \in P_d$, *assigned*(p, r_2) = true (a) `create` r_2'; (b) `add` ($r_1' \geq r_2'$) `to` H (4) $\forall r' \in X'$, $\forall p \in P_d$, *assigned*(p, r) $\rightarrow$ `assign`$_P$ p `to` r' (5) `assign`$_U$ r_d' `to` u_e	(1) `deassign`$_U$ r_d' `to` u_e (2) $\forall r'_1, r'_2 \in X'$, *s.t.* ($r'_1 \geq r'_2$) $\in H$, (a) `remove` ($r'_1 \geq r'_2$) `from` H (b) `delete` r'_1 (c) `delete` r'_2

permissions that can be acquired through r_d are assigned to the delegation role r_d'.

Partial Authorized Permission (PAP): PAP allows only a subset of permissions that can be acquired through the *delegator role* to be delegated to the *delegatee*. The BA and TEA approaches to achieving this are shown in Figure 4.4(d). PAP-BA can be implemented similar to the PDR-BA (hence, the steps for PDR-BA can be reused with a minor modification). Here, all the permissions that can be acquired through r_d (not just those assigned to r_d, as in PDR-BA) but not to be delegated are blocked. Another way to achieve it is by creating each *delegation sub-role* and explicitly filtering from each *delegation sub-role* the permissions of the *delegator sub-role* that are *not* to be delegated. However, this would require reversing the hierarchical relations among the filter roles as shown in Figure 4.4(d) (see table for the details). The PAP-TEA scheme can be implemented similar to that of PDR-TEA method. Again the difference would be that the delegation role now is assigned all the permissions that can be acquired through the *delegator* role, while in PDR-TEA, only the subset of permission that are directly assigned to the *delegator* role is assigned to the *delegation* role. Figure 4.4(d) shows another way to implement this scheme by creating the *delegation role* and *sub-roles*.

It is to be noted that, besides the difference between BA and TEA schemes (i.e., *dynamic* vs. *static*), the BA method is preferable when the permissions that are to be blocked from being delegated is relatively small. The TEA scheme is more appropriate when only a small subset of permissions is to be delegated. Furthermore, although the PAP-BA and PAP-TEA scheme is easily implemented as in PDR-BA and PDR-TEA, the alternatives shown in Figure 4.4(d) will be required when the *delegator roles/subroles* have different temporal properties (e.g., *enabling times*). In such cases, employing the PAP-BA and PAP-TEA schemes similar to the PDR counterparts will not maintain these

[2] The tables only show U2U delegation. R2R delegation can be easily derived from these.

temporal properties. Because of space limitation, how temporal hierarchies affect delegation semantics is not discussed in this paper[3].

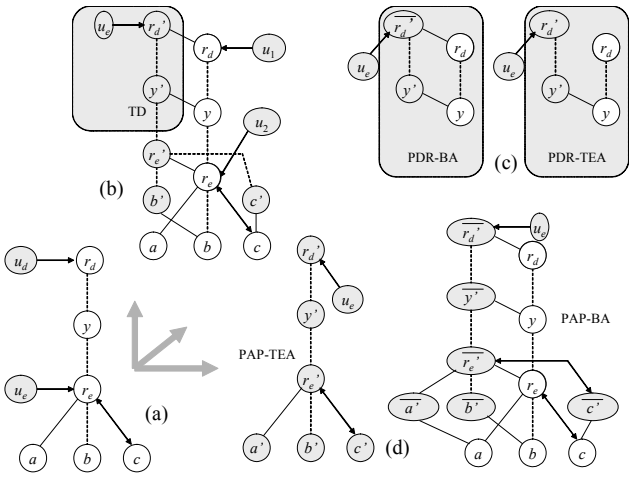

Figure 4.5 Downward delegation in A-hierarchy

4.2.2 Downward Delegation in A-hierarchy

Figure 4.5 shows various downward delegation policies when the *delegator role* and the *delegatee role* are related by an *A*-hierarchy, *i.e.* the case ($r_d \geqslant_g y$) and ($y \geqslant r_e$).

***Total delegation (TD)*:** The *delegatee* needs to delegate in totality the *delegator* role to the *delegatee*. As shown in Figure 4.5(b), the *delegation role* r_d' and the *delegation sub-roles* are created and the *delegator* is assigned to r_d'. Each of the *delegation role/sub-roles* created is an empty role and is made *I*-senior of its corresponding *delegator role/sub-role*. The *delegatee* can activate the main *delegation role* but to acquire the permissions of the roles junior to the *delegator role*, the *delegatee* has to activate the corresponding *delegation sub-role*. This is in par with the role activation capability of the *delegator* in the original hierarchy – the *delegatee* should be able to activate juniors but he does not acquire all the permissions simply by activating the main *delegation role*. Accountability is facilitated if we maintain log of the sessions in which the *delegatee* activates a *delegation role/sub-role*.

Note that there is even a *delegation sub-role* r_e*'* corresponding to the *delegatee role* r_e. Because of the separate *A*-path, the accountability can be easily achieved by logging the session information. The steps needed for enforcing this scheme is shown in Table 4.5. It is to be noted that if the *delegatee role* is related to its junior by an *I*-hierarchy, the *delegation sub-role* corresponding to that junior need not be created because the *delegatee* can acquire its permissions through the *delegation role.*

Partial Delegator Role (PDR): PDR-BA scheme involves employing a *filter delegation role* as shown in Figure 4.5(c). The remaining part of the hierarchy structure below *delegator role* r_d is recreated as in TD. PDR-TEA can be implemented similar to PDR-BA as shown in Figure 4.5 (c). Only difference will be (1) the *delegation role* r_d' is assigned all of r_d's permissions that need to be delegated; (2) unlike in PDR-BA, no hierarchical relation from r_d' to r_d is created. Table 4.5 shows the details.

Partial Authorized Permission (PAP): The first PAP scheme, PAP-BA is almost the same as TD. The difference is: (1) unlike in the TD, even the *filter delegation sub-role* for all roles in the sub-hierarchy below r_d (inclusive of r_d) is created; e.g., even filter role $\overline{a'}$ for role *a* is created as shown in Figure 4.5(d); (2) the *delegation sub-role* r_e' for the *delegatee role* r_e (note that it is also the *delegator* role) are related to the junior *delegation* sub-roles by the same hierarchical relations that relate the corresponding *delegatee* role with its corresponding juniors. The PAP-TEA scheme can be implemented similar to that of the PAP-BA scheme. Only difference is that the *delegation role* and all the *delegation sub-roles* are not hierarchically related to their original roles.

4.2.3 Downward Delegation in IA-hierarchy

Note that the *IA*-hierarchy is one that allows both permission-inheritance and role-activation semantics. This implies that when the *delegator role* is the *IA*-senior of the *delegatee role*, we can capture the delegation semantics by using the delegation schemes for *I* and *A*-hierarchies. Hence, we only illustrate the schemes in figures and do not provide details of each step. We categorize the following cases:

***Permission-only (or activation-only) delegation*:** Here only authorized permissions (or role activations) are delegated. In permissions-only (activations-only) delegation the role activation (permission-inheritance) semantics is not delegated. Hence, for permissions-only (activations-only) delegation, we apply the *I*-hierarchy (*A*-hierarchy) semantics described earlier. In particular, in *permissions*-only delegation only the *delegation role* is created and the *delegatee* assigned to it, as shown in Figure 4.6. The authorized permissions are made available to the *delegatee* when he activates the *delegation role*. Note that in the original role hierarchy, the *delegator* has authority to activate junior roles as well. This capability is not delegated.

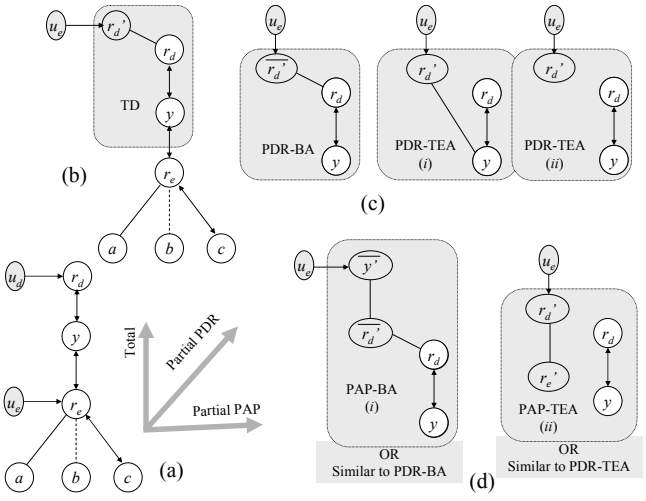

Figure 4.6. Permission-only delegation

***Permission-and-activation delegation*:** This feature allows both the authorized permissions and role activation capabilities to be

[3] We refer the readers to reference [Jos02a] for relevant information on temporal hierarchies.

delegated. The downward delegation in *IA*-hierarchy from the *delegator role* to a *delegatee* can be obtained by making simple modifications to the *A*-hierarchy, shown in Figure 4.7, as the structural changes are similar.

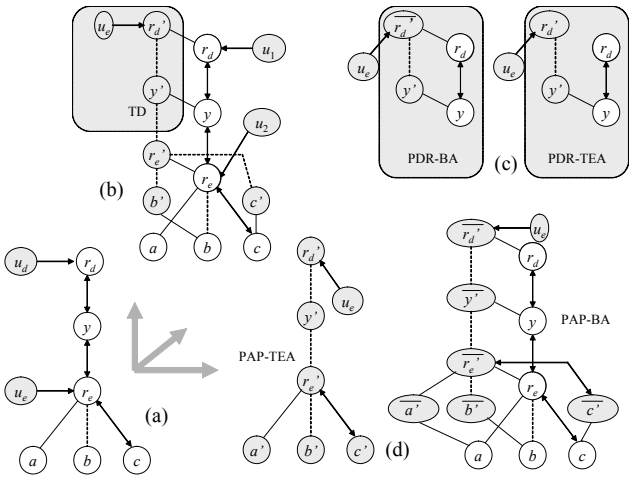

Figure 4.7 Permission and activation delegation

4.2.4 Upward Delegation in hybrid hierarchy

For upward delegation the *delegator role* is junior to the *delegatee role*. Hence, as in downward delegation, we do not have to worry about the various hierarchical relations that may exist between the two roles. As the steps can be easily derived as in tables 4.4 and 4.5, and because of the lack of space, we illustrate various U2U delegation schemes through the transformations illustrated in Figure 4.8.

Total delegation (TD): For the TD, we create the *delegation role* (r_d') and make it the *I*-seniors of the *delegator role* r_d. For each of the roles that is *I* or *IA*-junior of the *delegation role* (r_d), we create a *delegation sub-role* and make it *A*-junior of the *delegation role* r_d'. The delegation role and the delegation sub-roles are empty roles but are *I*-senior of the corresponding original roles. The *delegatee* is assigned to the *delegator role* as usual. The *delegatee* can activate each role separately, conforming to the original semantics. Note that we do not create a delegation sub-role for a role that is *I*-junior to the *delegator role* (role *a* in the figure) - the *I*-hierarchy between the delegation role and the *delegator* role allows the inheritance and hence the junior's permissions (role *a*' permissions in the figure) are acquired through r_d' by the *delegatee*.

***Partial Delegator Role (PDR)*:** The PDR-BA scheme is similar to TD scheme. Only difference is that here the *delegation role* $\bar{r}_d'$ is related to *delegation sub-roles* by the same hierarchical relations as those of their corresponding *delegator role* and *sub-roles* (*filter*) respectively. As in previous cases, the permissions assigned to the *delegator role* r_d and any *I*-junior of r_d not to be delegated are filtered. The PDR-TEA is similar to PDR-BA – the only difference is, here, no *delegation role/sub-role* is hierarchically related to its corresponding *delegator role/sub-role*. The permissions to be delegated corresponding to each *delegation role/sub-role* are explicitly assigned to them.

Partial Authorized Permissions *(PAP):* For PAP-BA scheme, the newly created delegation role $\bar{r}_d'$ is made *I*-senior of the *delegator role* r_d. As shown in Figure 4.8, the *filter delegation sub-roles* for *A* and *IA*-juniors (*b* and *c*) of the *delegator role* r_d are created and made *A*-juniors of the *delegation* role ($\bar{r}_d'$) and the *I*-seniors of their corresponding roles, respectively. For the PAP-TEA, the sub-hierarchy with the *delegator role* r_d as the senior-most role is recreated and the original hierarchical relations are

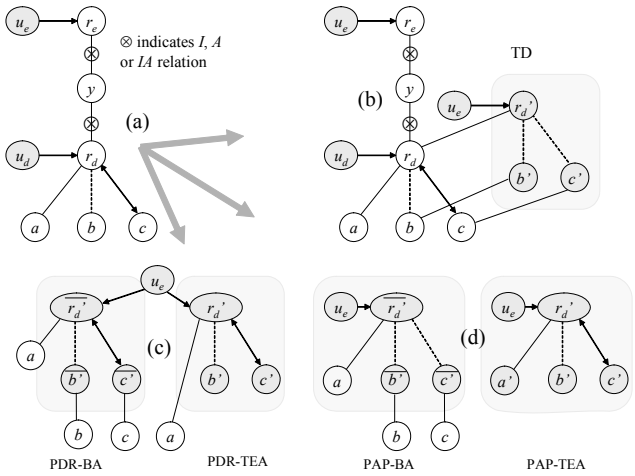

Figure 4.8 Upward delegation in a hierarchy

introduced between corresponding *delegation roles* and *sub-roles*, as depicted in Figure 4.8.

5. CONCLUSION AND FUTURE WORK

We have addressed the issues of delegation within the context of different role hierarchy types and hybrid hierarchies. We have presented several fine-grained downward and upward delegation schemes. We have shown that upward delegation plays an important role within the RBAC models. Furthermore, we argued that accountability consideration is crucial when delegation policies are considered. This affects the use of hierarchical relations when, during the delegation process, new roles are created. The delegations schemes subsume existing role-based delegation schemes. Due to space limitation, we focused on user-to-user delegation which is more fine-grained than role-to-role delegation and hence R2R delegation schemes can be easily derived from the U2U schemes presented here. Furthermore, cross-sectional delegation was not addressed due to space limitation; but it is simpler than delegation in the presence of hierarchies and can be easily derived from downward and upward schemes presented in detail here. Furthermore, we have not dealt with constraints on delegation such as specifying intervals or durations in which the delegation is to be valid, as well as the multi-step delegation and revocation schemes. We plan to pursue these as future work. Another future work is to develop a generic analysis framework for verifying correctness of policies when hierarchical, SoD, and delegation policies co-exists.

Acknowledgement: This research has been supported by the US National Science Foundation award IIS-0545912. We thank the anonymous reviewers for their helpful comments.

6. REFERENCES

[Atl05] V. Atluri, J. Warner, Supporting Conditional Delegation in Secure Workflow Management Systems, *ACM Symposium on Access Control Models and Technologies*, Sweden, Jun 1-3, 2005.

[Bar00] E. Barka and R. Sandhu, A Role-Based Delegation Model and Some Extensions, Pr*oc. of 23rd National Information Systems Security Conference,* Dec, 2000.

[Bar05] E. Barka and R. Sandhu, Role-Based Delegation Models/Hierarchical Roles, *Proc Annual Computer Security Application Conference*. 2004.

[Fer93] D. F. Ferraiolo, D. M. Gilbert, and N Lynch. An Examination of Federal and Commercial Access Control Policy Needs. In *Proceedings of NISTNCSC National Computer Security Conference*, pages 107-116, Baltimore, MD, September 20-23 1993.

[Fer01] D. F. Ferraiolo, R. Sandhu , S. Gavrila , D. Richard Kuhn , R. Chandramouli. Proposed NIST Standard for Role-based Access Control. *ACM Transactions on Information and System Security* (TISSEC) Volume 4, Issue 3, August 2001.

[Gas90] M. Gasser, E. McDermott, An Architecture for practical Delegation in a Distributed System, 1990 *IEEE Computer Society Symposium on Research in Security and Privacy*. May, 1990.

[Giu97] L. Giuri. Role-based access control: A natural approach. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[Goh98] C. Goh and A. Baldwin, Towards a more Complete Model of Role, Proc. of 3rd ACM Workshop on Role-Based Access Control. October, 1998.

[Jos01a] J. B. D. Joshi, A. Ghafoor, W. Aref, E. H. Spafford. Digital Government Security Infrastructure Design Challenges. *IEEE Computer*, Vol. 34, No. 2, February 2001, pages 66-72.

[Jos01b] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford. Security models for web-based applications. *Communications of the ACM*, 44, 2 (Feb. 2001), pages 38-72.

[Jos02a] J. B. D. Joshi, E. Bertino, A. Ghafoor. Temporal hierarchy and Inheritance Semantics for GTRBAC. 7th *ACM Symposium on Access Control Models and Technologies*. Monterey, CA, June 3-4, 2002.

[Jos03] J. B. D. Joshi, E. Bertino, A. Ghafoor. Hybrid Temporal Role Hierarchies in GTRBAC. *Submitted to ACM Transactions on Information and System Security*.

[Jos05] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor. Generalized Temporal Role Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering*, Vol 17, No. 1 pages 4-23, Jan, 2005.

[Liu04] R.W.C. Lui and L.C.K. Hui, A Model for Delegation of Accountability, *IASTED International Conference on Software Engineering*, SE 2004.

[Mof90] J. D. Moffett, Delegation of Authority Using Domain Based Access Rules, PhD Thesis. *Dept of Computing, Imperial College*, University of London. 1990.

[Nag98] N. Nagaratnam, D. Lea, Secure Delegation for Distributed Object Environments, *USENIX Conference on Object Oriented Technologies and Systems*. April, 1998.

[Osb00] S. Osborn, R. Sandhu, Q. Munawer. Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and System Security*, 3(2):85-106, May 2000.

[San96a] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman. Role-Based Access Control Models. *IEEE Computer* 29(2): 38-47, IEEE Press, 1996

[San96b] R. Sandhu. Role Hierarchies and Constraints for Lattice-based Access Controls. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo Eds., *Computer Security - Esorics'96*, LNCS N. 1146, Rome, Italy, 1996, pages 65-79.

[San98] R. Sandhu, Role Activation Hierarchies, 3rd *ACM Workshop on Role-Bused Access Fairfax* VA, 1998.

[San99] R. Sandhu, V. Bhamidipati and Q. Munawer, The ARBAC97 Model for Role-Based Administration of Roles, *ACM Transactions on Information and System Security*, Volume 2, Number 1, February, 1999.

[Sha04] B. Shafiq, J. B. D. Joshi, E. Bertino, A. Ghafoor, Secure Interoperation in a Multi-Domain Environment Employing RBAC Policies, Submitted to *IEEE Transactions on Knowledge and Data Engineering* (2004).

[Ste87] L. A. Stein, Delegation Is Inheritance, P*roc. Of Object-Oriented Programming Systems, Languages*, and Applications (OOPSLA '87). October, 1987.

[Tho97] R. K. Thomas. Team Based Access Control (TBAC): A Primitive for Applying Role-based Access Controls in Collaborative Environments. ACM Proceedings of the second ACM workshop on Role-based access control Fairfax., Nov, 1997.

[Wai05] J. Wainer, A. Kumar, A Fine-grained, Controllable, User-to-user Delegation Method in RBAC, *ACM Symposium on Access Control Models and Technologies*, Sweden, Jun 1-3, 2005.

[Zha03a] X. Zhang, S. Oh and R. Sandhu, PBDM: A Flexible Delegation Model in RBAC, *SACMAT* 2003.

[Zha03b] L. Zhang, G. Ahn, and B. Chu, A rule-based Framework for Role-Based Delegation, *ACM Transactions on Information and Systems Security*, Vol 6, No. 3, August 2003, Pages 404-4.