Computers & Security, 7 (1988) 599–605

Refereed Article

Nested Categories for Access Control

Ravinderpal Singh Sandhu

Department of Computer and Information Science, The Ohio State University, Colombus, OH U.S.A.

We consider an access control hierarchy based on nested categories. Nested categories are sets of compartments with the property that any two categories are either disjoint or one wholly contains the other. We present a method for representing nested categories by assigning a pair of integers called *lr*-values to each category. Authorization of a user to access an object is determined by comparing *lr*-values of the user and object categories. Our method has the notable property that categories can be reorganized by computing new *lr*-values for categories which are changed without affecting *lr*-values of unchanged categories.

Keywords: Access control, Protection, Hierarchy.

1. Introduction

It is often desirable to classify the users and objects in a computer system by means of protection attributes. For instance the objects in a project database may be classified by the attributes t_1, t_2 and t_3 corresponding to three independent tasks of the project. Members of each task team have the corresponding task as their protection attribute and can thereby access information in the project database regarding that task. Protection attributes may be combined conjunctively, for example $t_1 \wedge t_2 \wedge t_3$. Objects tagged with this composite attribute should be accessible only by users who are cleared for all three tasks, such as the project supervisors. Similarly, the disjunctive composite attribute $t_1 \vee t_2 \vee t_3$ identifies objects which can be accessed by users cleared for any of the tasks. That is information in this category is available to all members of the project regardless of the task they are assigned. Military security policies are the best known example of conjunctive protection attributes (for example, see refs. [1], [2] and [4]).

A set of *compartments* is defined. The *categories* are non-empty subsets of the set of compartments. The category $\{a, b, c\}$ is interpreted *conjunctively* as $a \wedge b \wedge c$, so information in this category is available to users who are members of all three compartments a, b, c. It is often useful to adopt the dual interpretation, so category $\{a, b, c\}$ is interpreted *disjunctively* as $a \vee b \vee c$ with access available to users who are members of any of the a, b, c, compartments (for example, see ref. [3]). We use the *notation* $\bigwedge p$ and $\bigvee p$ to denote the conjunctive and disjunctive interpretations of category p respectively. If p is a *singleton category* such as $\{a\}$ or $\{b\}$ both interpretations are equivalent and we usually omit specifying which one is intended.

Our project example above shows it is useful to have both conjunctive and disjunctive interpretations of categories in a system. Let compartments t_1, t_2 and t_3 be defined for three independent tasks. Members of these task teams are tagged by the corresponding singleton category $\{t_1\}$, $\{t_2\}$ or $\{t_3\}$. Information and resources exclusive to each task team are similarly tagged with the appropriate one of these singleton categories. In general, disjoint categories support *separation* of access. Now consider the category $p = \{t_1, t_2, t_3\}$. Information and resources tagged by $\bigvee p$ are to be accessible by users in any of the task teams. This allows the individual task teams to share information and resources which are common to the entire project. In this manner, disjunctive categories support *sharing*. In contrast, information and resources tagged conjunctively $\bigwedge p$ are to be accessible by users who happen to be members of all the task teams, say project supervisors. Supervisors are then tagged by $\bigwedge p$. Information tagged by $\bigwedge p$ in the project database is private to the supervisors. Moreover, $\bigwedge p$ subsumes the tags for the individual task categories so that in effect the supervisors are also automatically members of each task team. We say that conjunctive categories with this interpretation support *oversight*.

We require that each user in the system and each information item or resource is (securely) tagged by a disjunctive or conjunctive category. For the information and resources in the system we actually need a separate tag for each access

0167-4048/88/$3.50

mode. Thus a file may have its write mode tagged by $\{t_1\}$ and its read mode by $\bigvee \{t_1, t_2, t_3\}$. That is the file can be read by members of any task team but can only be written to by members of t_1. Having made this point, for simplicity in our discussion, we assume there is a single access mode for each information item and resource so there is a single tag on it. Our access rules are formally stated in Table 1. The access rule in the table states the condition under which access by the user to the information item is permitted.

Separation, sharing and oversight are fundamental access control issues which arise in practically every system. In this paper we propose a simple method for constructing categories in a natural way to support these basic issues. In a large organization with lots of diverse activities the number of compartments is likely to be very large, say in the hundreds. This is especially so if compartments are defined at a rather detailed level such as tasks within a project. With hundreds of compartments the possible set of categories becomes truly astronomical ($\approx 2^{100}$), so it is inevitable that only a very small fraction of these will ever be used. It is also a fact that in such a situation the compartments and categories are likely to change frequently over the course of time. A useful representation for categories in this context must take these issues into account. We consider that with a large number of changing compartments it is not necessary to have the complete flexibility to define any subset of compartments as a category. Without some discipline in constructing categories there is a likelihood of mistakes in defining these. Moreover, users may be hard pressed to know which categories do assign to their data objects. We consider it imperative that users see some simple structure of categories to which they can easily relate and which meets their most typical needs. We should single out the categories which are most frequently used and make this structure visible to the users and administrators. A complex structure with lots of rarely used categories is, in all probability, going to be confusing to the users and system administrators.

Table 1
Access rules for conjunctive and disjunctive categories

User category	*Information category*	*Access rule*
$\bigwedge u$	$\bigwedge i$	$u \supseteq i$
$\bigwedge u$	$\bigvee i$	$u \cap i \neq \phi$
$\bigvee u$	$\bigwedge i$	no access[a]
$\bigvee u$	$\bigvee i$	$u \subseteq i$

[a] Strictly speaking access should be allowed here if and only if u and i are the same singleton category, *i.e.* $u = i = \{a\}$. We can adopt this extension to the rules or require that singleton categories as user tags be interpreted conjunctively.

For instance, consider two projects p_1 and p_2 with tasks t_{11}, t_{12}, t_{13} and t_{21}, t_{22} respectively. The categories $\{t_{11}, t_{12}, t_{13}\}$ and $\{t_{21}, t_{22}\}$ are immediately suggested by this statement. If the two projects are related we might have the combined category $\{t_{11}, t_{12}, t_{13}, t_{21}, t_{22}\}$ also. However, the situation where we need a category such as $\{t_{13}, t_{21}\}$ is in all likelihood unusual. The norm would be that the protection requirements are met by the categories listed earlier which either include or exclude all task teams of a project. Our viewpoint is that while it is important to allow for unusual situations, it is far more important to handle the typical situation in a clean and efficient manner. We propose the following restriction on the definition of categories.

Definition 1

A set of categories is *nested* if any two distinct categories are either disjoint or one is properly contained in the other.

The categories $\{t_{11}\}$, $\{t_{12}\}$, $\{t_{13}\}$, $\{t_{21}\}$, $\{t_{22}\}$, $\{t_{11}, t_{12}, t_{13}\}$, $\{t_{21}, t_{22}\}$ and $\{t_{11}, t_{12}, t_{13}, t_{21}, t_{22}\}$ are nested. If we include $\{t_{13}, t_{21}\}$ the categories are no longer nested.

There are obviously some limitations in making this restriction. With the nested categories enumerated above, tasks t_{13} and t_{21} can share information only by using the category $\{t_{11}, t_{12}, t_{13}, t_{21}, t_{22}\}$. So t_{13} and t_{21} are unable to share information without making it accessible by every task team in both projects. In most applications this will not be a major restriction and is certainly consistent with the view that tasks within a project are more "tightly coupled" than tasks across different projects. At the same time we consider it very desirable that a mechanism allow some "escape" to accommodate deviations from a rigid nested structure. It is appropriate to require some additional effort for this purpose.

A simple but surprisingly powerful escape mechanism for this purpose is to allow multiple tags on users and information items. Then a file tagged by $\{t_{13}\}$ and $\{t_{21}\}$ is accessible by members of both task teams to the exclusion of all other task teams. Access is allowed if the user's tag matches any one of the file's tags according

to the rules stated in Table 1. This is an additional complication for the users who now have to assign more than one tag to some files when the access does not fall within the typical pattern. But this is acceptable if the typical usage only requires a single tag. At the very least, the need to assign more than one tag to a file alerts the user that this is a non-standard access decision on his part. Similarly, a user can be tagged with more than one category. For instance someone assigned to tasks t_{13} and t_{21} should have two tags corresponding to his different roles as a member of the two different task teams. The access rules are easily extended to allow access if any one of the user's tags matches any one of a file's tags as per Table 1. Multiple tags can accommodate a large class of deviations from a strict nested structure. However, there still are cases which cannot be accommodated. In this example a file cannot be tagged with $\bigwedge \{t_{13}, t_{21}\}$ but can be effectively tagged by $\bigvee \{t_{13}, t_{21}\}$ by assigning it multiple tags $\{t_{13}\}$ and $\{t_{21}\}$. Alternatively, a user cannot be tagged with $\bigvee \{t_{13}, t_{21}\}$ but can be effectively tagged by $\bigwedge \{t_{13}, t_{21}\}$ by assigning him multiple tags $\{t_{13}\}$ and $\{t_{21}\}$. Fortunately, the deviations which are accommodated are the more likely ones to occur.

One of the appealing properties of nested categories is that if we drop some categories those remaining continue to be nested. This is significant because in practice a user will be interested only in a small fraction of the categories that have been defined. For instance a member of task team t_{11} may not care to know about categories which do not involve tasks from his project. So his interest is only in the categories $\{t_{11}\}$, $\{t_{12}\}$, $\{t_{13}\}$, $\{t_{11}, t_{12}, t_{13}\}$ and $\{t_{11}, t_{12}, t_{13}, t_{21}, t_{22}\}$. For nested categories the user is guaranteed to always see a nested set, whatever categories he may choose to focus on.

The superset relation on nested categories is a forest of trees (more precisely the Hasse diagram of this relation has this shape). For instance the project categories enumerated earlier have the tree structure shown in Fig. 1. Categories consisting of single compartments are the leaves of this tree. If the topmost category is removed we have a forest. Formally we have the following lemma.

Lemma 1. The superset relation on nested categories is a forest of trees.

Proof. By definition of nested categories for distinct B, C either $B \subset C$ or $C \subset B$ or $B \cap C = \phi$. Now if A is a common subset of B and C, the last option is ruled out. That is, $B \supset A$ and $C \supset A$ implies $B \subset C$ or $C \subset B$. So the supersets of any category are linearly ordered. That is, in the Hasse diagram there is at most one path leading upward from every category. But this describes a forest of trees.

This is a crucial property of nested categories. Henceforth we assume the nested categories are given to us as a forest of trees.

In Section 2 we present a simple technique for representing a nested set of categories so it is easily determined when one category is a subset of another. Our technique assigns a pair of integers called *lr*-values to each category so that $A \supset B$ if and only if $l[A] < l[B]$ and $r[A] < r[B]$. In Section 3 we extend the method so that when a subtree of categories is reorganized new *lr*-values need only be computed for categories which replace this subtree without affecting *lr*-values of unchanged categories. Section 4 concludes the paper with a discussion.

2. *Lr*-numbering

Our technique for assigning *lr*-values to nested categories is based on the familiar concept of preorder traversal of a forest defined by the following recursive procedure for each tree in the forest.

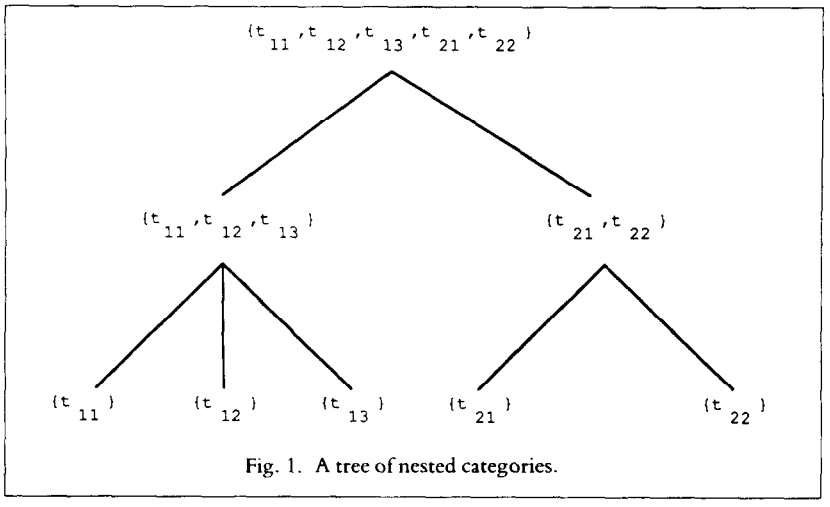


Fig. 1. A tree of nested categories.

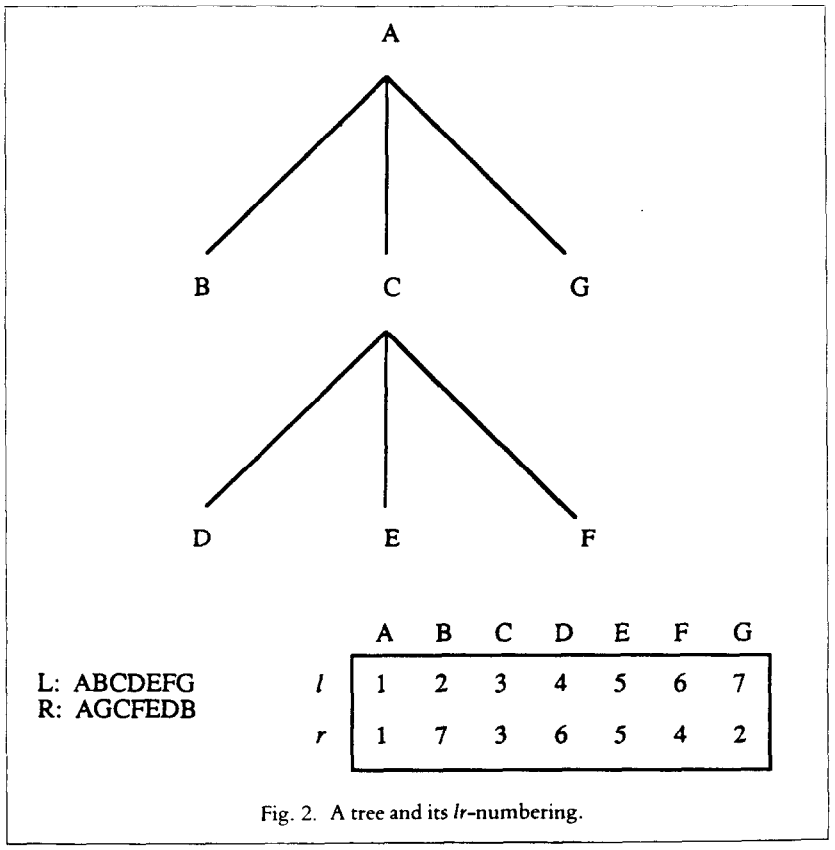
(1) Visit the root.
(2) Traverse the subtrees, if any, of the tree in pre-order.

We perform two pre-order traversals of the forest. In the left pre-order traversal the individual trees in the forest and the subtrees of each tree are visited in left to right order; in the right pre-order traversal they are visited in right to left order. These traversals are respectively identified as L and R. For the tree of Fig. 2, $L = ABCDEFG$ and $R = AGCFEDB$. The position of a category in the L and R linear orderings respectively defines the *lr*-values assigned to the category, as indicated in the figure. From these values it is easily determined for instance that $A \supset C \supset D$ while B and C are incomparable. The assignment of *lr*-values is called an *lr-numbering* of the forest.

The following theorem is easily proved.

Theorem 1. $A \supset B$ is and only if $l[A] < l[B]$ and $r[A] < r[B]$.

Proof. By lemma 1, $A \supset B$ implies A is a predecessor of B in the forest. Then A precedes B in L and R, so $l[A] < l[B]$ and $r[A] < r[B]$. Conversely, if $l[A] < l[B]$ and $r[A] < r[B]$ then A precedes B in both L and R so $A \supset B$. If A and B are incomparable the path from A to the root of its tree is either to the left or right of the path from B to the root of its tree. In the former case, A precedes B in L and follows B in R, while in the latter case A follows B in L and precedes B and R. So if A and B are incomparable we cannot have $l[A] < l[B]$ and $r[A] < r[B]$. Thus $l[A] < l[B]$ and $r[A] < r[B]$ implies $A \supset B$.



	A	B	C	D	E	F	G
l	1	2	3	4	5	6	7
r	1	7	3	6	5	4	2

Fig. 2. A tree and its *lr*-numbering.

The *lr*-numbering is a convenient and efficient representation of nested categories. The access decisions of Table 1 can be easily determined on the basis of *lr*-values of the categories. Table 2 restates the rules of Table 1 in terms of *lr*-values for nested categories. The third row is unchanged while the first and fourth rows follow trivially from theorem 2. The second row follows from theorem 2 and the observation that for nested categories if $u \cap i \neq \phi$ either $u \supseteq i$ or $i \supseteq u$ (or both). In the context of Fig. 2 a user with tag $\bigwedge C$ can access information with any of the tags $\bigwedge C, D, E, F, \bigvee C$ or $\bigvee A$. However, a user with tag C can access only those information items tagged by $\bigvee C$ or $\bigvee A$. We can visualize these rules as stating that a conjunctive tag allows access to conjunctive and disjunctive categories below that tag and also the disjunctive categories above that tag in the path to the root of the tree. A disjunctive tag allows access only to disjunctive categories above that tag in the path to the root of the tree.

3. *Lr*-numbering with Quotas

Now consider what happens if the categories are reorganized. By *reorganization* we mean that a subtree is replaced by a forest of subtrees. For instance, the subtree un-

Table 2
Access rules for nested categories in terms of *lr*-values

User category	*Information category*	*Access rule*
$\bigwedge u$	$\bigwedge i$	$l(u) \leq l(i) \bigwedge r(u) \leq r(i)$
$\bigwedge u$	$\bigvee i$	$l(u) \leq l(i) \bigwedge r(u) \leq r(i) \bigvee$ $l(i) \leq l(u) \bigwedge r(i) \leq r(u)$
$\bigvee u$	$\bigwedge i$	No access[a]
$\bigvee u$	$\bigvee i$	$l(i) \leq l(u) \bigwedge r(i) \leq r(u)$

der C in Fig. 2 may be replaced by the subtree shown in Fig. 3. In the resulting *lr*-numbering the *lr*-values of all categories other than A and C get changed. This presents a problem in applying our technique in practice, since reorganization of categories in this manner is likely to be a frequent occurrence. It is particularly awkward that *lr*-values of categories B and G which are unrelated to the reorganized subtree are affected. It should be noted that reorganization may involve a sweeping change of a tree or be confined to a very small part of it.

In the remainder of this paper we present a technique for assigning *lr*-values in such a way that categories can be reorganized by computing new *lr*-values for categories which are changed without affecting the *lr*-values of unchanged categories. The key idea is to assign a *quota* $q(A)$ to each category A specifying the maximum number of new categories that can be introduced in place of A. More precisely, when a subtree of categories is replaced by a forest of categories the maximum number of categories in the new forest cannot be greater than the sum of the quotas of the categories in the original subtree. The minimum value of the quota for any category is 1. The method works for any initial assignment of quotas. The maximum number of categories that can ever exist is the sum of the quotas of all existing categories. When a subtree is reorganized, the total quota of the subtree is arbitrarily partitioned among the categories in the new forest.

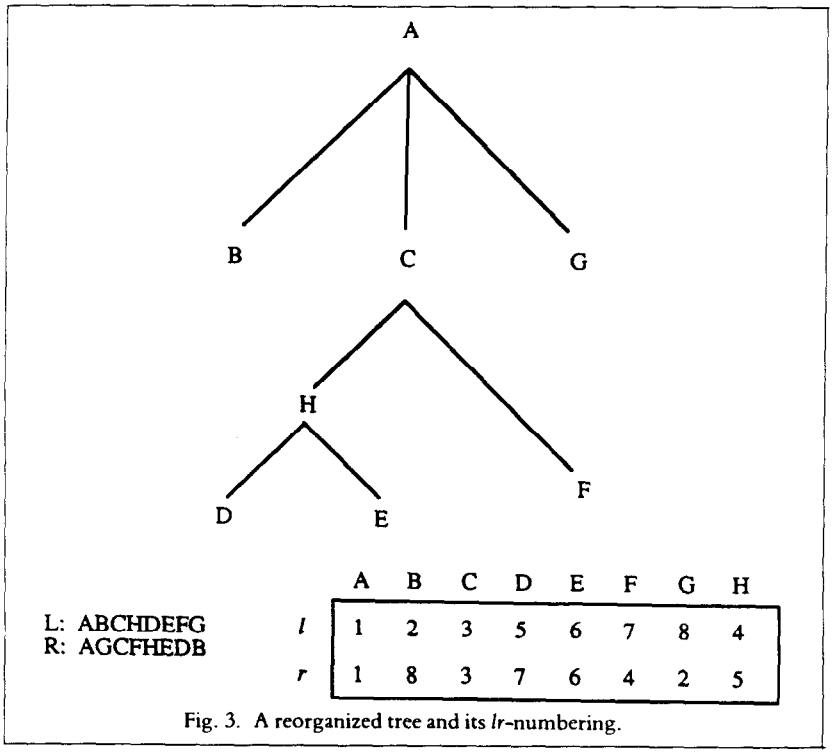


Fig. 3. A reorganized tree and its *lr*-numbering.

We first describe our technique informally by example and then prove its correctness. Assume we are given left and right pre-order traversals L and R respectively. The *lr*-values are assigned as follows.

$$l(A) = 1 + \sum q(X)$$

for all X preceding A in L

$$r(A) = 1 + \sum q(X)$$

for all X preceding A in R

We call this the *quota-based lr-numbering*. If the categories in Fig. 2 have a quota of 5 each, the *lr*-numbering obtained by this method is as shown in Fig. 4(a). Now let the subtree under C in Fig. 2 be reorganized to obtain the tree shown in Fig. 3. In the former case the subtree under C has a total quota of 20. Our restriction is that the new subtree under C should also have a total quota of 20. That is, the total quota is conserved under reorganization. Let us say the

categories in the subtree under C in Fig. 3 have a quota of 4 each. The lr-numbering obtained by the quota-based method is shown in Fig. 4(b). It should be noted that the lr-values of categories A, B, C and G are unchanged.

In general, with quota-based lr-numbering, new lr-values need only be assigned to categories in the forest which replaces the reorganized subtree. The reason for this is that in the pre-order traversal, categories in a subtree all occur consecutively. But then, so long as the sum of quotas is conserved, categories outside the reorganized subtree will be assigned the same lr-values. We now prove this property.

Theorem 2. With quota-based lr-numbering when a subtree is reorganized the lr-values of categories outside this subtree are unchanged, provided reorganization conserves the total quota of the subtree.

Proof. Let L and R respectively be the left and right pre-order traversals before reorganization. Let L' and R' respectively be left and right pre-order traversals after reorganization. The categories in the reorganized subtree must occur consecutively in L and in R. A category that precedes the reorganized subtree in L has the same prefix of categories in L and L'. So its l-value is unaffected by reorganization. A category that follows the reorganized subtree in L has a different prefix in L and L'. However, the sum of the quotas of categories in its prefix is unchanged. Therefore, its l-value is unchanged. Similar arguments are easily made for the r-values.

It follows from the theorem that lr-values for the reorganized categories can be computed from the left and right pre-order traversals of the forest which replaces the reorganized subtree rather than requiring these traversals for the entire forest. The computation begins with the lr-values of the root of the reorganized tree which tells us the sum of the quotas of all preceding categories up to this point and then proceeds by accumulating the sum of the quotas. As a particular case, if the reorganized subtree is replaced by another subtree (rather than a forest of subtrees) the lr-values of the category at the root of the subtree are also unchanged. Another important special case is that a leaf is replaced by a forest. That is, a compartment is broken up into finer compartments. In this case the lr-values of all categories other than this leaf are unchanged.

L: ABCDEFG
R: AGCFEDB

	A	B	C	D	E	F	G
q	5	5	5	5	5	5	5
l	1	6	11	16	21	26	31
r	1	31	11	26	21	16	6

(a)

L: ABCHDEFG
R: AGCFHEDB

	A	B	C	D	E	F	G	H
q	5	5	4	4	4	4	5	4
l	1	6	11	19	23	27	31	15
r	1	31	11	27	23	15	6	19

(b)

Fig. 4. Quota-based lr-numbering: (a) for Fig. 2; (b) for Fig. 3.

4. Discussion

We have proposed an access control hierarchy based on nested categories as suitable for an environment with a large number of changing categories and compartments. We have shown that the superset relation on nested categories is a forest of trees. We have devised a representation for nested categories by assigning a pair of integers called lr-values to each category so that $A \supset B$ if and only if $l[A] < l[B]$ and $r[A] < r[B]$. We have shown that by assigning quotas we can reorganize a subtree of categories without affecting the lr-values of categories outside this subtree. This notable property

makes our representation easy to maintain in the face of inevitable changes to categories and compartments. Our techniques offer a simple and efficient mechanism which directly solves the commonly occurring case of nested categories. Commonly occurring deviations from the nested structure can be accommodated by allowing multiple tags on users and information items. We consider the additional effort required to specify multiple tags quite acceptable.

The quota-based method allows us to reorganize a subtree without affecting the *lr*-values of categories outside the subtree. The method is feasible only if the total quota of the old subtree and the new subforest are the same. Consider the reorganization of the old subtree under C in Fig. 2 to the new subtree in Fig. 3. Suppose the quotas of C, D, E, and F in Fig. 2 happen to be at the minimum value of 1. In that case this reorganization is not feasible since the new subtree requires a minimum value quota of 5. However, if there is sufficient quota available in the subtree under A we can treat this as a reorganization of the subtree under A. The point is that if the quota is insufficient we can back up one level and attempt to reorganize there. Of course the cost is that the *lr*-values of all categories may now be changed. Nevertheless, it is an important property of the method that limitations on reorganization due to quota can be overcome by backing up far enough so sufficient quota is available. With proper planning and generous allocation of quotas these occurrences should be rare. It should be noted that we can afford to be generous in allocating quotas since the *lr*-values require only $\log_2(M)$ bits each, where M is the total quota of the tree.

The access control hierarchy discussed in this paper is a special case of the ntree hierarchy recently defined by this author [5]. As such, the *lr*-numbering of nested categories can be derived from the similar *lr*-numbering of ntrees. Our method for reorganization of a subtree is, however, specific to nested categories. With nested categories the subtree is a natural unit for reorganization. In the case of ntrees there is no such immediately apparent natural unit for reorganization. It would be interesting and useful to develop a similar notion of reorganization for ntrees.

References

[1] D.E. Denning, lattice model of secure information flow, *Commun. ACM*, 19(5) (May 1976) 236–243.

[2] D.E. Denning and P.J. Denning, Data security, *ACM Comput. Surv.*, 11(3) (September 1979) 227–249.

[3] D. Estrin, Controls for inter-organization networks, *IEEE Trans. Software Eng.*, SE-13(2) (February 1987) 249–261.

[4] C.E. Landwehr, Formal models for computer security, *ACM Comput. Surv.*, 13(3) (September 1981) 247–278.

[5] R.S. Sandhu, The ntree: a two dimension partial order for protection groups, *ACM Trans. Comput. Syst.*, 6(2) (May 1988) 197–222.

Dr. Ravi Sandhu is currently an assistant professor of computer and information science at the Ohio State University in Columbus, OH 43210, where he has been since 1982. He earned his Ph.D. degree in computer science from Rutgers University in 1983, from where he had earlier obtained an M.S. degree. He also holds the M.Tech. and B.Tech. degrees in electrical engineering from the Indian Institutes of Technology at Bombay and Delhi respectively. He has worked for Hindustan Computers Limited, Jawaharlal Nehru University and the Indian Institute of Technology all in New Delhi, India. His principal research activity has been in the area of formal models for specification, analysis and implementation of computer security policies. He has developed the Schematic Protection Model and the NTree Hierarchy, articles on which have recently appeared in *Journal of the ACM* and *ACM Transactions on Computer Systems* respectively. He has also published on cryptographic techniques for access control. He has presented his work at several conferences including the *IEEE Symposium on Security and Privacy*. His larger research interests are centered on the principles and foundations of computer systems design, particularly operating systems, parallel and distributed computing and information systems.