SOME OWNER BASED SCHEMES WITH DYNAMIC GROUPS IN THE SCHEMATIC PROTECTION MODEL

Ravinderpal S. Sandhu
Michael E. Share

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

ABSTRACT

The discretionary ability implied in the notion of ownership is often provided in terms of groups rather than individuals. The group approach has its conveniences but is limiting since access decisions do not discriminate among members of a group. The ability for users to define group membership dynamically offers flexibility and convenience. In this paper we investigate a variety of polices for doing so in the context of a simplified file system. We specify the polices using the Schematic Protection Model (SPM). Our investigation reveals that there are many policy options and demonstrates how SPM is used to precisely specify these options.

1. INTRODUCTION

The notion of ownership is a fundamental aspect of discretionary access control policies. For instance, the owner of an entity such as a text file can determine who may or may not access the file. Typically, only the owner has the power to make these decisions and ownership is closely tied with creation, *e.g.*, the creator of a file becomes its owner by virtue of creating it.

Ownership carries with it the connotation of complete discretion regarding access by other parties. A mechanism which allows discretion at the level of individual users offers great flexibility. We often find in practise that the unit for access is a group of users. That is, the owner of a file can either provide or deny access to an entire group of users. Since access is not selective within a group the owner's discretion is limited by the group structure. At the same time, the group approach is convenient since it eliminates the need to make many individual decisions. A mechanism which permits groups to be dynamically defined by users offers flexibility and convenience.

In this paper we present several owner based policies where access decisions are made on a group basis. These policies are specified using the Schematic Protection Model or SPM [10, 11]. SPM offers a intuitive yet precise notation for defining and analyzing different policies. Section 2 presents a brief but complete description of SPM. In section 3 we develop a variety of policies based on the notion of owners and groups. The policies differ mainly in the dynamics of group membership. For simplicity we consider groups to be independent entities without any structure, such as a hierarchical organization. Our investigation reveals that even in this simple context there are genuine policy options and demonstrates the use of SPM in specifying and analyzing these options. Section 4 concludes the paper.

2. THE SCHEMATIC PROTECTION MODEL

SPM regards a system as consisting of *subjects* and *objects*. Each subject possesses a set of *privileges* called its *domain*. Objects do not possess privileges and are merely containers of information. For example, users are subjects while text files are objects. Subjects and objects are collectively called *entities*.

The distribution of privileges within a system defines the *protection state* of the system. Only those operations authorized by the current protection state can be executed. *Control operations* modify the protection state, e.g., user X gives user Y permission to read file Z, and are authorized by *control privileges*. *Inert operations* on the other hand preserve the protection state, e.g., user X reads or writes file Y. Inert operations are authorized by *inert privileges*. The initial protection state is established by the *security administrator*. The protection state subsequently evolves due to the autonomous actions of subjects as constrained by control privileges. Hereafter, we understand state to mean protection state.

The concept of *protection type* is fundamental to SPM. The idea is that entities of the same protection type are treated uniformly by control privileges. SPM requires strong typing in that every entity is created to be of a specific protection type which cannot change. In SPM the protection state of a system consists of a static component called the *scheme*

and a dynamic component comprised of *tickets*. The scheme is defined by the security administrator in terms of protection types and cannot change. The distribution of tickets may change as a result of control operations. Hereafter, type is understood to mean protection type.

Tickets are privileges of the form Y/x, where Y identifies a unique entity and the *right symbol* x authorizes the possessor of this ticket to perform some operation(s) on Y. Tickets are considered unforgeable and can be acquired only in accordance with specific rules which comprise the scheme. Sets of tickets for the same entity are abbreviated by allowing Y/uvw to denote the tickets Y/u, Y/v, and Y/w.

Tickets are similar to the well-known concept of capabilities. We use the more neutral term to avoid implying that the only representation for tickets is by means of capabilities built into the addressing mechanism of a computer [6]. The run-time representation of tickets is an implementation issue, properly left open by the model.

2.1. Types and Right Symbols

In specifying a scheme it is first necessary to specify finite sets of *object types* TO and *subject types* TS. These sets must be disjoint and their union T is the set of *entity types*. By convention types are named in lower case italics and entities in upper case normal script. For the schemes discussed in this paper, TO consists of a single type *fil* corresponding to files, while TS consists of several types such as *usr*, *dir* and *grp* corresponding to users, directories and groups respectively.

The next step is to define a finite set of right symbols R partitioned into two disjoint subsets: the inert rights RI and the control rights RC. For the schemes of section 3, RI consists of r and w respectively authorizing the read and write operations on a file. Control rights will be discussed shortly.

SPM assumes that each right symbol x can come with the *copy flag*, denoted xc, or without, denoted x. The difference between Y/x and Y/xc is that only the latter allows Y/x or Y/xc to be copied from one domain to another. The ability to copy a ticket is also dependent on other restrictions to be explained below. Moreover, this is the only difference between Y/x and Y/xc, so that the presence of Y/xc in a domain implies the presence of Y/x but not vice versa. We use x:c to denote either x or xc, with the understanding that multiple occurrences of x:c within the same context are either all x or all xc. When used with multiple right symbols the copy flag applies to each symbol, that is Y/uvc denotes Y/uc and Y/vc.

The type of a ticket is determined by the type of the entity to which it refers and the specific right symbol, *e.g.*, if F is a file then F/r is a ticket of type *fil*/r. The set of *ticket types* is thereby $T \times R$. Conventions for representing tickets, especially regarding the copy flag, extend in an obvious way to ticket types.

The remaining components of a scheme are defined in terms of T, R, $T \times R$ and their subsets. SPM recognizes three operations which change the protection state: *copy*, *demand* and *create*.

2.2. The Copy Operation

The copy operation moves a copy of a ticket from the domain of one subject to the domain of another subject. The original ticket is unchanged. The copy operation is authorized by the copy flag, a *link predicate* $link_i$, and the associated *filter function* f_i. We emphasize there is a different filter function for each link predicate.

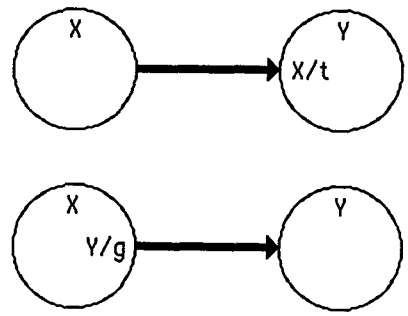

Figure 1 The Take-Grant Link

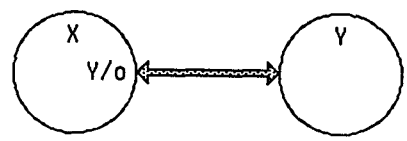

Figure 2 The Owner Link

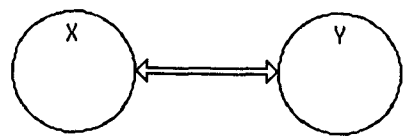

Figure 3 The Universal Link

Link Predicates

A link is a directed connection from one subject to another which facilitates copying tickets from the former to the latter. Links are brought into existence by control privileges. A link predicate is simply the definition of a link in terms of the control privileges required.

For example the *take-grant link*, adapted from the take-grant model [4, 7, 12], is defined as follows where dom denotes the domain of a subject.

$$\text{link}_{tg}(X,Y) \Leftrightarrow Y/g \in \text{dom}(X) \vee X/t \in \text{dom}(Y)$$

Here, t and g are control rights respectively called take and grant. We visualize this link as shown in figure 1, by depicting a subject as a circle with the interior of the circle being the subject's domain and by depicting the link as a directed edge. The take-grant link is used in the schemes of section 3.

Another link predicate we use in this paper is the *owner link* defined below in terms of the control right o for ownership.

$$\text{link}_{o}(X,Y) \Leftrightarrow Y/o \in \text{dom}(X) \vee X/o \in \text{dom}(Y)$$

In words, there is a owner link from X to Y if either X is the owner of Y or Y is the owner of X. We represent this pictorially in figure 2. In our diagrams we use different color edges to distinguish the owner link from the take-grant link. Note, the take-grant link is uni-directional whereas the owner link is bi-directional.

Finally, we will have occasion to use the *universal link* link_u which does not require any control rights. The universal link always exists and is represented pictorially in figure 3.

In SPM we can define any number of link predicates. The only restriction is that each predicate be *local* in that link(X,Y) requires the presence of some combination of control tickets for X and Y in the domains of X and Y. The motivation is that a local predicate can be evaluated by looking for control tickets for the two subjects of concern in the domains of these two subjects. That the authorization should be in terms of the presence and not the absence of tickets is a well-known principle for protection [1, 2, 8].

Filter Functions

The filter function determines which ticket types can be copied across a given link. The filter function is defined for each link predicate and for each pair of subject types. It specifies the types of tickets which can be copied across a link_i between subjects of given types. Therefore, f_i is a function f_i: $TS \times TS \rightarrow 2^{T \times R}$. Filter functions impose mandatory controls which are inviolable and confine the discretionary behavior of individual subjects.

To summarize, the ticket Y/x:c can be copied from dom(A) to dom(B) if and only if all of the following are true for some i, where the types of A, B and Y are a, b and y respectively.

1. $Y/xc \in \text{dom}(A)$
2. $\text{link}_i(A,B)$
3. $y/x{:}c \in f_i(a,b)$

In this manner the copy flag, a link predicate and corresponding filter function together authorize a copy operation. The first two conditions depend on the distribution of tickets whereas the third condition depends on the scheme. Selectivity in the copy operation is controlled by the filter function and specified entirely in terms of types.

2.3. The Demand Operation

The demand operation allows subjects to obtain tickets by demanding them. SPM authorizes the demand operation through specification of a demand function d: $TS \rightarrow 2^{T \times R}$. If $a/x{:}c \in d(b)$, then each subject B of type b can demand the ticket A/x:c for every entity A of type a. In particular, control tickets can be demanded. The demand operation is used in some of the schemes of section 3 to facilitate dynamic grouping.

The demand function is a powerful method for specifying availability of tickets on the basis of protection types. At the same time the policy embodied in the demand function is static because of the static definition of this function and the strong typing in SPM.

2.4. The Create Operation

The create operation is the means by which new subjects and objects are introduced. The create operation has two aspects, authorization and which tickets are introduced as a result of the operation. Authorization is specified by the *can-create relation* $cc \subseteq TS \times T$. Subjects of type a are authorized to create entities of type b if and only if $cc(a,b)$. The tickets introduced by a create operation are determined by the *create-rule* specified for each pair in cc. All create-rules are *local*, *i.e.*, tickets can only be introduced for the creating subject and created entity. If subject A of type a creates entity B of type b, the create-rule $<a,b>$ determines which tickets for A and B are placed in the domains of A and B. Note that the created entity B can obtain tickets only if B is a subject.

2.5. Summary

To summarize, a *scheme* in SPM is defined by specifying the following components.

1. A finite set of entity types T partitioned into subject types TS and object types TO.
2. A finite set of right symbols R partitioned into inert rights RI and control rights RC.
3. A finite collection of local link predicates.
4. A filter function f_i: $TS \times TS \rightarrow 2^{T \times R}$ for each link predicate $link_i$.
5. A demand function d: $TS \rightarrow 2^{T \times R}$.
6. A can-create relation $cc \subseteq TS \times T$.
7. A local create-rule for each pair in cc.

A *system* is specified by defining a scheme, the initial set of entities and the initial distribution of tickets. Thereafter the system state evolves by copy, demand and create operations.

Since the role of the security administrator is to specify the initial state, we consider that the initial state defines the *authorization policy*. The key idea in SPM is that major policy decisions are made in terms of types and embodied in the scheme, whereas more detailed considerations are reflected in the initial distribution of tickets.

3. OWNER BASED SCHEMES WITH DYNAMIC GROUPS

In this section we present a variety of owner based schemes in SPM. The particular policy decisions we make in this paper are not so important as the demonstration that there are a variety of possibilities and that SPM can accommodate this variety. We begin by discussing the assumptions and definition common to all schemes considered here.

3.1. The Basic Framework

The basic framework is a file system abstracted from the facilities typically found in current multi-user Operating Systems. In this paper we regard files as pure information containers without any embedded privileges. Thereby files are SPM objects. We define a single object type *fil* for this purpose. We recognize the read and write operations on files respectively authorized by the r and w rights. So the inert rights in our schemes are {r:c,w:c}. Inclusion of append, execute and delete operations will not substantially change our schemes and, in the interest of brevity, these operations are omitted.

The active entities in the system are users. Users can create files and at their discretion share them with other users. The system provides directories for organizing a user's files. A file is associated with a directory by placing tickets for the file in the directory's domain. Since directories contain privileges for files they are considered to be subjects in SPM. Directories are passive, however, and can not initiate any actions. Groups are included in the system to facilitate sharing. Members of a group are able to access directories in the group, and through the directories, files. Groups contain privileges for both users and directories, and thereby are subjects in SPM. Like directories, groups are passive subjects and cannot initiate any actions even though they contain privileges. We define three subject types *usr*, *dir* and *grp* corresponding to users, directories and groups respectively.

A user can create files, directories and groups. On creating a file F its creator gets the F/rwc tickets. We set up our schemes so that the creator is the only user to ever get tickets with the copy flag for a file. Other users may be able to access the file but only via non-copiable tickets.

The control rights we use in our schemes are take, grant and ownership with the link predicates $link_{tg}$ and $link_o$ as defined in section 2.2. We define our schemes so that the owner right cannot be copied or demanded. Thus the owner link is a static link which is established either in the initial state or by creation. The take-grant link on the other hand is dynamic.

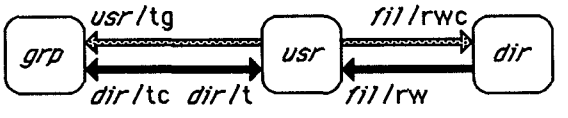

Figure 4 The Basic Filter Functions

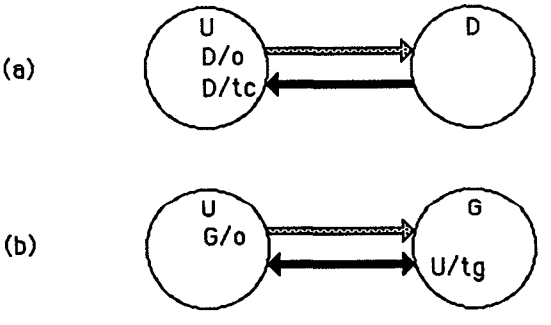

Figure 5 Create-Rules for Directories and Groups

The basic interaction between users, groups and directories is shown in figure 4. Each subject type is represented as a rounded rectangle with the labels on the colored edges denoting values of f_o and f_{tg}. Only those edges with non-empty labels are shown. Figure 4 is interpreted as follows.

$f_o(usr,grp) = \{usr/t,\ usr/g\}$
$f_o(usr,dir) = \{fil/rc,\ fil/wc\}$
All other values of f_o are empty.

$f_{tg}(usr,grp) = \{dir/tc\}$
$f_{tg}(grp,usr) = \{dir/t\}$
$f_{tg}(dir,usr) = \{fil/r,\ fil/w\}$
All other values of f_{tg} are empty.

There is no *usr* to *usr* edge in figure 4 so tickets cannot be directly copied from one user to another. Instead the interaction between users is via groups and directories. All non-empty edges involve the *usr* type at one end. It is understood that the corresponding links will be exercised by the user. That is, irrespective of where the authorization for a link lies it is the user who will initiate a copy operation across the link. In general $f_t(a,b)$ should be non-empty only if at least one of a or b is an active subject type. Figure 4 is clearly consistent with this requirement.

The owner link is the only way to place copiable tickets for files in a directory's domain. The take-grant link allows users to obtain file tickets from a directory. Tickets acquired from a directory in this manner are without the copy flag. It follows that a user has copiable tickets only for those files that he creates. The net effect is that a directory contains tickets for files created by the directory's owner. For a particular file F it is up to the creator to place F/rc or F/wc or both in any directory that he owns.

Membership in a group is effected by placing take and grant tickets for a user in the group's domain. Since these tickets are distinct there are actually two kinds of group membership. If U/g is in group G's domain then there is a $link_{tg}$(G,U) and U can access directories in the group. Conversely, if U/t is in G's domain then there is a $link_{tg}$(U,G) and U can contribute directories to the group.

User U can be made a member of a group only by one of the group's owners. In order to do so, the group's owner must first obtain the tickets U/tgc and then copy U/t or U/g or both to the group's domain. The schemes presented in this paper differ primarily in the method by which a user obtains tickets of type *usr*/tgc. For now we ignore this issue.

When a directory D is created the tickets D/o and D/tc are placed in the creator's domain by the <*usr,dir*> create-rule as shown in figure 5(a). D is made accessible to members of a group by placing D/tc in the group's domain. Members of the group can then copy the ticket D/t thereby obtaining access to files in D. The create operation is the only way for a user to obtain tickets of type *dir*/tc, so only the owner of a directory can place it in a group.

When a group G is created by user U the ticket G/o is placed in U's domain so the creator becomes the owner of the group. At the same time, the tickets U/tg are placed in G's domain so that U is automatically made a member of G. This <*usr,grp*> create-rule is shown in figure 5(b).

We use the demand function very sparingly in our schemes. For now we set it to be empty. All this results in the following scheme.

1. $TO = \{fil\}$
   $TS = \{usr,\ dir,\ grp\}$
2. $RI = \{r{:}c,\ w{:}c\}$
   $RC = \{t{:}c,\ g{:}c,\ o{:}c\}$
3. Two link predicates $link_{tg}$ and $link_o$ as in figures 1 and 2.
4. The filter functions f_{tg} and f_o as shown in figure 4.
5. $d(usr) = d(dir) = d(grp) = \phi$
6. $cc = \{<usr,fil>,\ <usr,dir>,\ <usr,grp>\}$
7. The <*usr,fil*> create-rule is that the creator of file F gets the F/rwc tickets. The <*usr,dir*> and <*usr,grp*> create-rules are as shown in figure 5.

An example of a protection state for this scheme is shown in figure 6. Users U_1 and U_2 are both members of group G. U_1 has created files F_1, F_2, F_3 and directories D_1, D_2. U_2 has created files F_4, F_5 and directories D_3, D_4. Both users have placed some of the tickets for their files in their directories. The state of figure 7 can now be derived in the following manner. U_2 copies D_3/tc to G's domain from where U_1 obtains D_3/t thus establishing $link_{tg}(D_3,U_1)$. This enables U_1 to copy F_4/r and F_5/rw from D_3's domain. Unless U_2 takes some further action U_1 cannot obtain the ticket F_4/w. Similarly, unless U_1 places one of his directories in G's domain, U_2 cannot obtain any tickets for U_1's files.

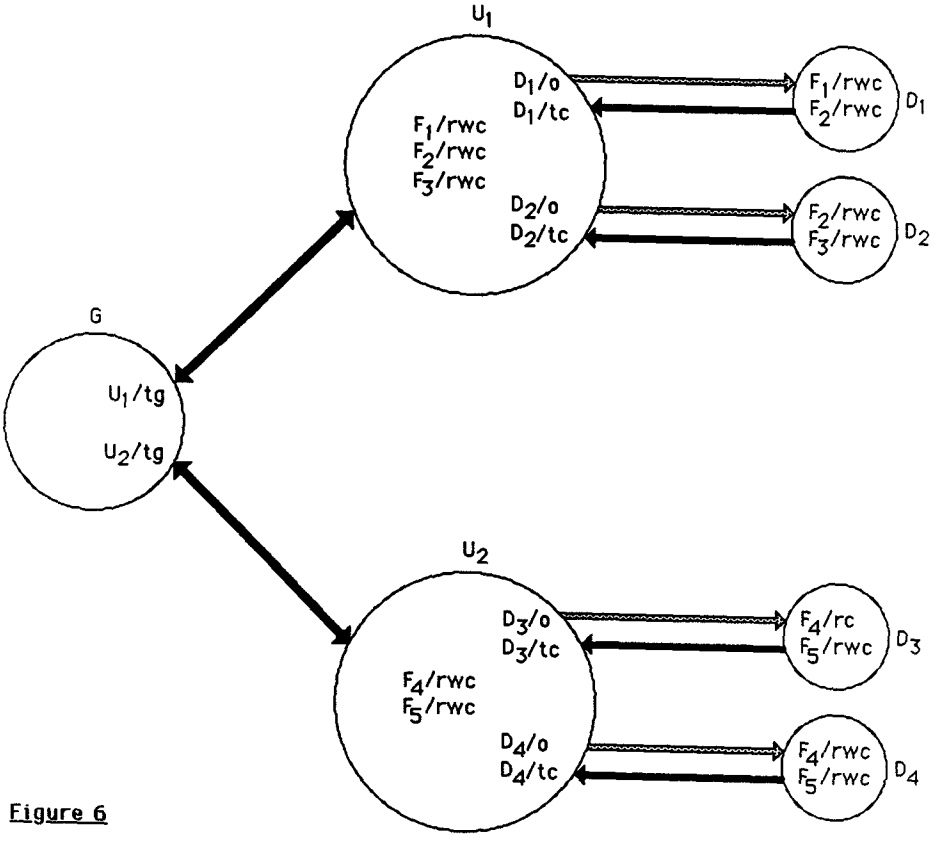

Figure 6

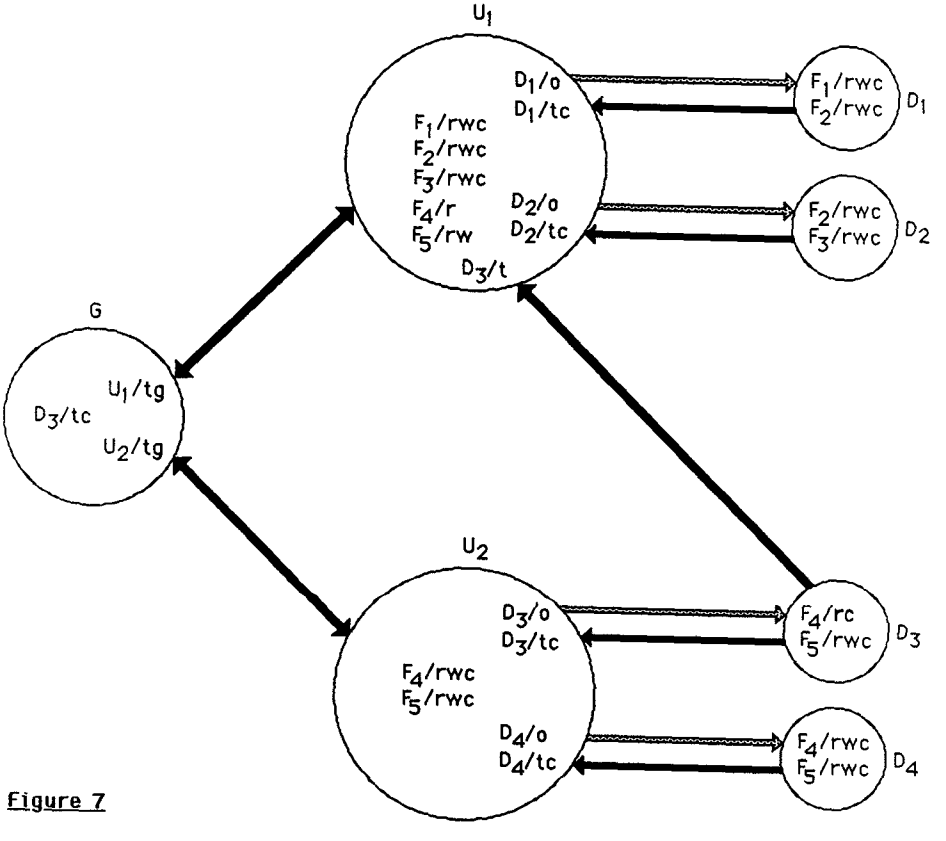

Figure 7

As mentioned earlier the above scheme is incomplete in that there is no method for a user to obtain tickets of type *usr*/tgc. Thus the dynamics of group membership is severely limited by the initial distribution of *usr*/tgc tickets in users' domains. We now present a variety of schemes which allow users to dynamically obtain *usr*/tgc tickets in a number of different ways. For simplicity we will not distinguish methods for obtaining *usr*/tc tickets from those for obtaining *usr*/gc tickets.

We assume throughout that a directory does not contain privileges for other directories, and a group does not contain privileges for other groups. This framework cannot accommodate hierarchies of directories and hierarchies of groups. Specification of such hierarchies in SPM is fairly straightforward, but is beyond the scope of this paper.

3.2. Dynamic Group Membership Based on the Copy Operation

One approach is to allow users to obtain *usr*/tgc tickets from the existing groups. The motivation is that a user can establish groups whose members are derived from groups he currently has access to. Thus there is a degree of mandatory control on the discretionary ability of group owners to introduce new members. There are a variety of ways this can be achieved in SPM resulting in different policies. Some of these are shown in figure 8.

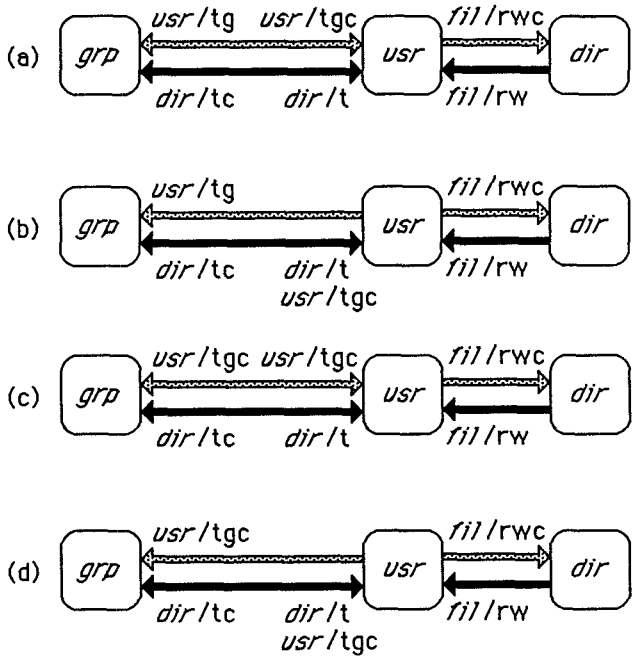

Figure 8 Variations Based on the Copy Operation

In figure 8(a) a user can obtain *usr*/tgc tickets from groups that he owns. Thus a user can propagate membership across only his owned groups. When the system is initialized, copiable control rights for certain users are placed into certain groups at the security administrator's discretion. The owner of a group containing such rights can copy them from that group to any other group he owns. This scheme allows created groups to be used for sharing, however the initial distribution of copiable user control rights strictly limits such sharing.

The policy of figure 8(b) is more liberal in that any group member, not just the group owner, can obtain *usr*/tgc tickets from a group. Again, these tickets are distributed when the system is initialized and this limits the dynamics of group membership.

Figure 8(c) is a modification of figure 8(a) which allows a group owner to place *usr*/tgc tickets in the group's domain. Now the *usr*/tgc tickets in a group's domain are no longer limited to those present in the initial state, so the modified policy is more liberal. This change has an impact only if there are some groups in the initial state with multiple owners. Figure 8(d) is a similar modification of figure 8(b). In this case the resulting policy is extremely liberal and there are essentially no mandatory controls on group membership, if the users cooperate.

It is apparent that 8(a) is the least liberal policy and 8(d) the most liberal. The other two policies are somewhere in between. Our purpose here is not so much to analyze these options in detail. Rather we wish to point out that many options exist with subtle differences in the resulting policy and that SPM makes these distinctions precise and analyzable. From a practical viewpoint the policies of figure 8 are surprisingly subtle and their exact implications somewhat difficult to understand. It is our intent in SPM to avoid such subtleties. Nevertheless it is a great asset of SPM that such policies can be formally analyzed [11].

3.3. Dynamic Group Membership Based on the Demand Operation

A much cleaner and straightforward approach is to allow users to obtain *usr*/tgc tickets by means of the demand operation. We specify this in SPM by setting $d(usr)$ to $\{usr/tgc\}$. This is useful if we wish to allow users complete discretion regarding membership in the groups they own. We show this in figure 9 where the funnel attached to the *usr* type represents the demand function. So long as mandatory controls are not required this policy is highly desirable. It has the appealing property that a user can establish a group consisting of an arbitrary subset of users.

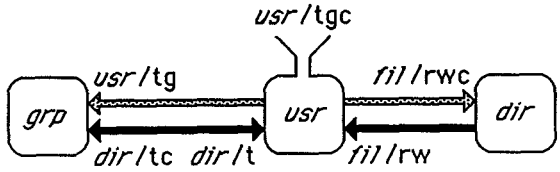

Figure 9 Dynamic Group Membership Based on Demand

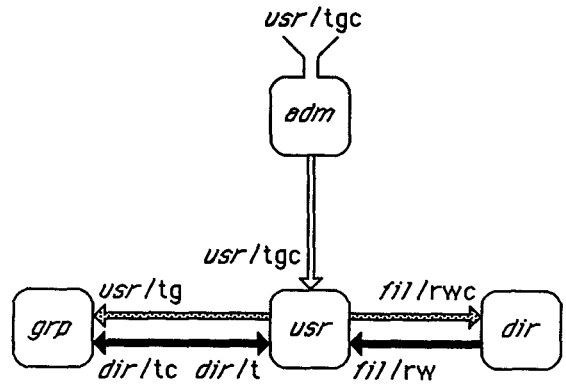

Figure 10 Dynamic Group Membership Based on the Administrator's Discretion

It is possible to specify mandatory controls in the demand operation. Consider a requirement that a user should only be able to form groups consisting of users in the same department as himself. Instead of a single subject type *usr*, we can define a subject type usr_i for each department i and set $d(usr_i)$ to be $\{usr_i/\text{tgc}\}$. This is an attractive and simple method for specifying the stated constraint.

Using the demand operation in this way has its limitations because of the strong typing of SPM entities and the static definition of the demand function. So long as the policy decisions can be cast in concrete and are not likely to be frequently changed we can build policy into the demand operation. We now demonstrate that SPM is also capable of specifying such mandatory controls dynamically.

3.4. Dynamic Group Membership with Administrator's Discretion

In the policy of figure 9 the user has complete discretion regarding membership in his own groups. We will show how this discretion can be mediated by system administrators. For this purpose we introduce a new subject type *adm* and make use of the universal link of figure 3 which always exists. We authorize administrators to demand copiable take-grant tickets for users by defining $d(adm)$ to be $\{usr/\text{tgc}\}$. Users can obtain these tickets from administrators across the universal link. This results in figure 10. In effect the discretionary ability of users to demand *usr*/tgc tickets has been replaced by a copy operation from administrators. This ensures that only those *usr*/tgc tickets which a administrator considers appropriate will be available to a particular user.

The obvious problem with this policy is that a administrator has to intervene whenever a user wants to make another user a member of one of the groups that he owns. Moreover the administrator exercises his discretionary power at the very low level of individual tickets. This places a considerable burden on administrators. We solve this problem by introducing a new subject type *sgrp* for system groups. System groups are used to supply users with *usr*/tgc tickets. The administrator "stocks" system groups with these tickets and connects them to users using take-grant links. This can be done dynamically by the administrator. All this results in figure 11. Now administrators need not be involved in every user request for *usr*/tgc tickets. To complete the scheme we allow administrators to create system groups as well as users. In both cases the create-rule is empty since no tickets need to be generated.

As a enhancement we may as well allow system groups to be used for sharing directories in addition to being a source for *usr*/tgc tickets. This is shown in figure 12. The system groups now provide added convenience for users who can use them for sharing so long as the policy built therein, by the security administrator, suffices for their needs. At the same time users can create their own groups, to provide more restricted sharing or to combine users from different system groups in to a single group.

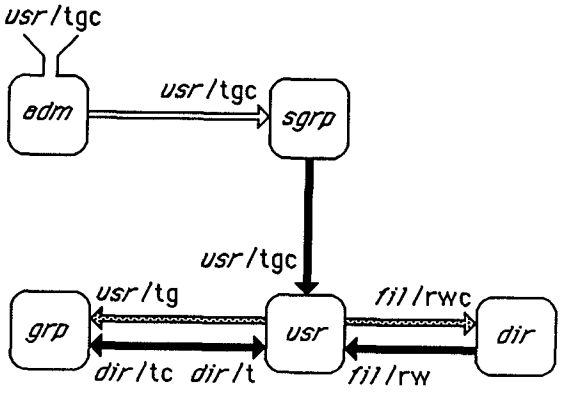

Figure 11 Dynamic Group Membership Based on System Groups

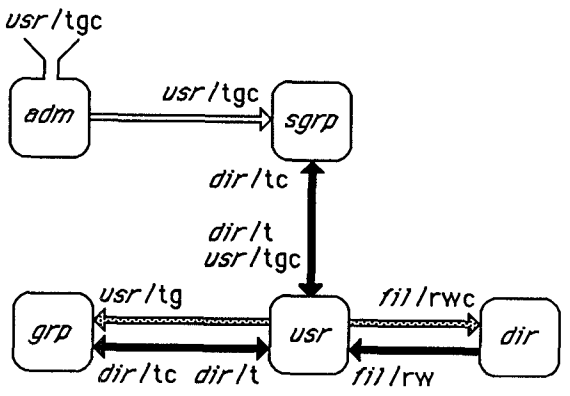

Figure 12 Enhancement of System Groups for Sharing Directories

4. CONCLUSION

It has been shown that SPM can conveniently model owner based discretionary policies. The specifications are intuitive. Simple policies result in simple specifications and small changes to the policies result in small changes to the specifications. Although SPM has strong typing, the examples have shown that it is possible to model dynamic grouping with SPM in a straightforward way. We emphasize that the notion of ownership is not an inevitable component of every SPM scheme. Indeed there are many situations where ownership is inappropriate and SPM can model such policies [9].

Inspite of the simplicity of the policies specified here, few Operating Systems can support all of them. We believe it is important for system designers to provide facilities to conveniently support the variety of policies which arise even in simple contexts. At the same time the policy specifications must be analyzable so we may verify that the formalism captures our requirements.

SPM has been carefully developed to balance these conflicting goals of convenient generality and tractable analysis. SPM provides a much richer structure than the traditional access-matrix model, first proposed by Lampson [5]. Moreover analysis of the access-matrix is undecidable in general and there do not appear to be meaningful constraints leading to tractable analysis while retaining generality [3]. In contrast analysis of SPM is tractable given that the only cycles in the can-create relation are of length one [11] (which allow a subject to create subjects of its own type). The owner-based schemes discussed here trivially satisfy this constraint.

5. REFERENCES

[1] Denning, D. E.
Cryptography and Data Security.
Addison-Wesley, 1982.

[2] Graham, G. S. and Denning, P. J.
Protection - Principles and Practice.
In *Proc. SJCC*, pages 417-429. AFIPS, 1972.

[3] Harrison, M. H., Russo, W. L. and Ullman, J. D.
Protection in Operating Systems.
CACM 19(8):461-471, Aug., 1976.

[4] Jones, A. K., Lipton, R. J. and Snyder, L.
A Linear Time Algorithm for Deciding Security.
In *Proc. 17th Symp. on the Foundations of Comp. Sci.* IEEE, 1976.

[5] Lampson, B. W.
Protection.
Proc. 5th Princeton Symp. of Info. Sci. and Syst. :437-443, March, 1971.

[6] Levy, H. M.
Capibility-Based Computer Systems.
Digital Press, 1984.

[7] Lipton, R. J. and Snyder, L.
A Linear Time Algorithm for Deciding Subject Security.
JACM 24(3):455-464, Dec., 1977.

[8] Saltzer, J. H. and Schroeder, M. D.
The Protection of Information in Computer Systems.
Proc. of IEEE 63(9):1278-1308, Sept., 1975.

[9] Sandhu, R. S.
The SSR Model for Specification of Authorization Policies: A Case Study in Project Control.
In *Proc. 8th Int. Comp. Softw. and Appl. Conf*, pages 482-491. IEEE, November, 1984.

[10] Sandhu, R. S.
Analysis of Acyclic Attenuating Systems for the SSR Protection Model.
In *Proc. 1985 Symp. on Security and Privacy*, pages 197-206. IEEE, April, 1985.

[11] Sandhu, R. S.
The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes.
Submitted for publication.

[12] Snyder, L.
Formal Models of Capability-Based Protection Systems.
IEEE Trans. Comp. C-30(3):172-181, March, 1981.