THE SSR MODEL FOR SPECIFICATION OF AUTHORIZATION POLICIES: A CASE STUDY IN PROJECT CONTROL

Ravinderpal S. Sandhu

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

ABSTRACT

The distribution of privileges in the domains of subjects (e.g. users, processes) defines the **protection state** of a system. Subjects can change this state as authorized by the current state. An **authorization policy** specifies which states are safe and also how they should be derived. The **SSR model** (Schematic Send-Receive model) is motivated by two conflicting goals: generality in specifying practical policies, and analyzability in characterizing derivable states. The key notion is **protection types.** SSR regards the domain of each subject as consisting of a static part determined by the subject's type and specified by an **authorization scheme**, and a dynamic part consisting of **tickets** (capabilities). The authorization scheme embodies major policy decisions while details are reflected in the initial distribution of tickets. We demonstrate the expressive power of this model, and the role of authorization policies, in the context of a project documentation system.

1. INTRODUCTION

The **authorization** or **protection** problem arises in any computer system which permits sharing of data objects and other resources. Such systems are viewed as consisting of **subjects** and **objects.** Subjects model active entities such as users and processes. Objects model passive entities such as text files. Protection is enforced by ensuring that only those operations for which the invoking subject possesses **privileges** in its **domain** actually get executed. Operations may be performed on objects, e.g., reading a text file, and on subjects, e.g., blocking a process.

We regard subjects and objects as disjoint sets and use the term **entity** to denote either a subject or object. We assume that objects do not possess privileges. Passive entities which possess privileges are regarded as subjects. Passive subjects cannot initiate operations but serve an important function as repositories of privileges which can be obtained and used by active subjects, e.g., a directory which contains privileges for accessing files but itself cannot initiate the use of these privileges. In this respect, our viewpoint differs from that prevailing in the literature where subjects are regarded as a subset of objects.[1, 2, 3] The rationale for the prevalent viewpoint is that any entity on which operations can be performed is an object, and since operations can be performed on subjects they too are objects. We prefer to take it for granted that operations may be performed on both subjects and objects, while making explicit the distinction between entities that do and do not possess privileges.

The distribution of privileges in the domains of subjects defines the **protection state** of a system. **Inert privileges** authorize operations which do not modify the protection state, e.g., reading a file. **Control privileges** authorize operations which modify the protection state, e.g., user X authorizes user Y to read file Z. The paradigm is that an initial protection state is established and thereafter the state evolves as constrained by control privileges. The challenge is to construct an initial state such that all derivable states are consistent with the underlying policy.

Now what do we mean by policy in this context? At the simplest level an **authorization policy** defines a set of **safe** protection states where the distribution of privileges is consistent with the underlying objectives, e.g., the policy that states where user X cannot read file Y are safe. At all times the system must be in a safe state. Safety considerations are typically **value-based** and concerned with classes of entities rather than individuals, e.g., the policy that only users in department D can access files internal to department D. Such a policy is also said to be **selective** since users and files in different departments are treated differently.

At a more sophisticated level, an authorization policy must consider the **sequence of state transitions** which derive a safe state. It is not enough that the system be in a safe state, we must additionally ensure the system arrived at the safe state in a proper manner. For instance, the policy that users outside department D may access internal files of department D provided the chairperson of D approves. Besides being value-based and selective this policy is also **cooperative** in that the chairperson's cooperation

0730-3157/84/0000/0482$01.00

is required, and **discretionary** in that the chairperson decides which non-departmental users may access which internal files.

A **protection model** provides a framework and formalism for policy specification and must be general enough to conveniently state the kinds of concerns discussed above. But generality by itself is not enough. To understand the formal statement of a policy and to assure it captures our intent, we need analysis techniques to characterize the states that a system with a given initial protection state may arrive at. Since subjects are usually authorized to create new subjects and objects, we are confronted with unbounded systems and it is not certain that analysis can be decidable let alone tractable without sacrificing generality.

The central issue in formulating a protection model is to balance these conflicting goals of convenient generality and tractable analysis. Analysis issues were first formalized[4] in context of the well-known access-matrix model.[1, 2, 3] Not surprisingly, analysis is undecidable in this general setting. A further drawback of the access-matrix is the lack of any structure to conveniently address practical policy concerns. The take-grant model,[5, 6] and its variations,[7] while efficiently analyzable accommodate only a very specific class of simple policies. The send-receive transport model of Minsky[8] incorporated powerful facilities for specifying the kinds of policy concerns discussed above. It also introduced the notion of local analysis, i.e., analysis based on examination of a subset of a larger system and thereby independent of the rest of the system, a property crucial to modular design.

In section 2 we present the **Schematic Send-Receive** or **SSR model**. In large part, SSR is based on Minsky's send-receive transport model. SSR adopts some simplifying assumptions and has a set-theoretic formulation in contrast to Minsky's formulation which has a production-rule flavor. Though not named as such, the SSR model is developed in greater detail in Sandhu.[9] In sections 3 and 4 we illustrate the expressive power of SSR by specifying a variety of policies for a project documentation system. Our discussion is relevant to any project where the preparation and use of documentation by a team is computerized, e.g., software development projects. Section 5 concludes the paper.

2. THE SSR MODEL

The key concept in SSR is **protection types**. Intuitively, instances of the same protection type are treated uniformly by control privileges. Henceforth, we use type as synonymous with protection type. A critical assumption in SSR is **strong typing**, i.e., every entity is created to be of exactly one type which cannot change thereafter. SSR treats a subject's domain as consisting of two parts: a static part determined by the subject's type, and a dynamic part which varies with the protection state.

Dynamic privileges are represented as **tickets** (capabilities) of the form Y/x, where Y identifies some unique entity and the **right symbol** x authorizes the possessor of this ticket to perform some operation on Y. We do not intend that tickets necessarily be represented at run-time as capabilities built into the addressing mechanism of a computer.[10, 11, 12, 13] The correspondence between SSR tickets and the run-time representation of dynamic privileges in a given implementation, is a separate issue. In particular, the assumption that a ticket carries only one right symbol, and identifies exactly one entity, simplifies the formal framework. A capability with multiple right symbols is modeled as a set of tickets each with one right symbol. Indeed, we sometimes abbreviate a set of tickets in this manner so that A/xy denotes A/x and A/y.

We assume every right symbol x comes in two variations x and xc where c is the **copy flag**. A ticket Y/x cannot be copied from one domain to another, whereas Y/xc possibly may be as determined by additional conditions. In all other respects these two tickets are identical. By definition, whenever A/xc belongs to a set of tickets A/x also belongs to the set but not vice versa. We use x:c to denote either x or xc. Multiple occurrences of x:c in the same context are either all read as x or all as xc. A/xy:c is an abbreviation for A/x:c and A/y:c.

Static type-dependent privileges are determined by the **authorization scheme**. The scheme is defined when a system is first set-up and cannot thereafter be changed. The person responsible for defining the scheme is called the **security administrator**. Major policy decisions are built into the scheme, while details are reflected in the initial distribution of tickets also specified by the security administrator. We recognize the need for modifying the scheme over the life of a system. However, such modification must be carried out by the security administrator acting outside the system rather than by subjects within the system, and as such fall outside the scope of the SSR model. For the purposes of SSR any change in the scheme amounts to specification of a new system. We find it useful to view an authorization scheme as analogous to a database schema, and the distribution of tickets as analogous to an extensional database.

The first step in defining a scheme is to specify the set of **object types TO**, and the set of **subject types TS**. These sets must be disjoint. Their union **T** is the entire set of protection types. By convention, types are named in lower case boldface and entities in upper case normal script.

The next step is to define the right symbols which can be carried by tickets. The set of right symbols **R** is partitioned into two disjoint subsets: **RI** the set of inert rights and **RC** the set of control rights. **RC** is fixed to consist of the **send** and **receive** rights denoted s and r respectively, and their copiable variants, i.e.,

$$\mathbf{RC} = \{s, r, sc, rc\}$$

These control rights authorize transport of tickets via the **link relation** defined by

$$\text{link}(A,B) \text{ iff } [B/s \in \text{dom}(A) \text{ and } A/r \in \text{dom}(B)]$$

where dom denotes the set of tickets possessed by the indicated subject. The existence of link(A,B) is necessary, but not sufficient, for transport of tickets from A to B. In short **RC** is fixed and interpreted in terms of the link relation, while **RI** is specified by the security administrator and regarded by SSR as a set of uninterpreted symbols.

We define the type of a ticket Y/x:c to be **t**(Y)/x:c, where the **type function t** returns the type of its argument entity. Conventions for representing tickets, especially regarding the copy flag, extend in an obvious way to ticket types. In particular, **t**(Y)/x and **t**(Y)/xc are different ticket types. This is an important distinction because of the role of the copy flag. The entire set of **ticket types** is **TXR.**

The remaining components of an authorization scheme are defined entirely in terms of subject, object, and ticket types, i.e., in terms of the sets **TS**, **TO**, and **TXR**. SSR recognizes three operations by which a subject can obtain tickets: **transport**, **demand**, and **creation**. A subject in the initial state may be given an arbitrary set of tickets to begin with as specified by the security administrator. We now discuss in turn each of the three operations which change the protection state.

The **transport operation** moves a copy of a ticket from the domain of one subject to the domain of another. The original ticket is left intact in the former domain. The scheme constrains the transport operation by means of the **filter function f** which maps every pair of subject types to a subset of ticket types, i.e.,

$$\mathbf{f}: \mathbf{TS} \times \mathbf{TS} \rightarrow 2^{\mathbf{TXR}}$$

The interpretation is that a ticket Y/x:c can be copied from dom(A) to dom(B) if and only if

1. $Y/xc \in \text{dom}(A)$,

2. link(A,B) exists,

3. $\mathbf{t}(Y)/x{:}c \in \mathbf{f}(\mathbf{t}(A),\mathbf{t}(B))$.

The first two conditions were stated earlier as necessary but not sufficient. The filter function completes this list and defines the selectivity in the transport operation in terms of the types of source and destination subjects and type of ticket being transported. SSR makes no assumptions about the role of A and B in this operation. It is equally acceptable that transport take place at the initiative of A or B alone or require both to cooperate. We often speak of the transport operation as copying a ticket from one subject to another, although it is technically more correct to say that a ticket is copied from one subject's domain to another's domain.

The **demand operation** allows a subject to obtain tickets simply by demanding them. A scheme authorizes this operation by the **demand function d** which maps every subject type to a subset of ticket types, i.e.,

$$\mathbf{d}: \mathbf{TS} \rightarrow 2^{\mathbf{TXR}}$$

The interpretation of $\mathbf{a}/x{:}c \in \mathbf{d}(\mathbf{b})$ is that every subject of type **b** can demand the ticket A/x:c for every entity A of type **a**. In particular, control tickets can be demanded allowing us to incorporate special cases conveniently, e.g.

1. If $\mathbf{b}/s \in \mathbf{d}(\mathbf{a})$ then every subject A of type **a** can demand B/s for every subject B of type **b**. The definition of link(A,B) for such subjects effectively reduces to $A/r \in \text{dom}(B)$.

2. Correspondingly, if $\mathbf{a}/r \in \mathbf{d}(\mathbf{b})$ then for subjects A, B of types **a**, **b** respectively the definition of link(A,B) effectively reduces to $B/s \in \text{dom}(A)$.

3. If both $\mathbf{b}/s \in \mathbf{d}(\mathbf{a})$ and $\mathbf{a}/r \in \mathbf{d}(\mathbf{b})$ then for subjects A, B of types **a**, **b** respectively link(A,B) is effectively true.

The **create operation** introduces new subjects and objects in the system. There are two issues here: what types of entities can be created, and what happens after a create operation occurs. The first issue is specified in a scheme by means of the **can-create relation cc** which relates subject types to types, i.e.,

$$\mathbf{cc} \subseteq \mathbf{TS} \times \mathbf{T}$$

The interpretation is that subjects of type **a** are authorized to create entities of type **b** if and only if $\langle \mathbf{a},\mathbf{b} \rangle \in \mathbf{cc}$. The second issue is specified by a **create rule** for every pair in **cc.** Assume, subject A of type **a** creates entity B of type **b.** If B is an object, the **<a,b>** create rule tells us which tickets for B are placed in A's domain as a result of this operation. If B is a subject, the create rule must also tell us which tickets for A are placed in B's domain. SSR requires every create rule be **local** in that the only tickets introduced are for the creating and created entities in the domains of the creating and created entities. The idea is that frequently occurring incremental events such as creation of an entity should immediately have only a local incremental impact on the state. We emphasize the create rule may be different for each pair in **cc**. We omit the symbolic formalism for specifying create rules.

In summary, the SSR model requires the security administrator to specify an authorization scheme by defining the following components.

1. **TS** the set of **subject types**, and **TO** the set of **object types. T**, the set of types, is the union of these two disjoint sets.

2. **RI** the set of **inert rights. RC** = {s, r, sc, rc}, the set of **control rights** is fixed. **R**, the set of rights, is the union of these two disjoint sets.

3. **f: TS** X **TS** -> $2^{\mathbf{TXR}}$, the **filter function** to specify selectivity in the transport operation.

4. **d: TS** -> $2^{\mathbf{TXR}}$, the **demand function** to specify the types of tickets that can be obtained on demand by subjects of a given type.

5. **cc** $\subseteq$ **TS** X **T**, the **can-create relation** to specify the types of entities that can be created by subjects of a given type.

6. A **local create rule** for each pair in **cc** to specify the immediate result of a create operation authorized by this pair.

A **protected system** is specified by defining an authorization scheme, the initial set of entities, and the tickets in the initial domain of every subject.

In our discussion so far we have ignored the important aspect of **revocation** of privileges. Our cavalier treatment of revocation is justified by adopting the **restoration principle** that whatever can be revoked can be restored. What is the interpretation of this principle in the context of SSR? The assumption that the scheme cannot be changed is fundamental to SSR and implies that revocation is limited to tickets. Any policy for revocation of tickets consistent with the restoration principle is then acceptable.

Now consider the ways in which a subject obtains tickets. If a ticket obtained by a transport operation is revoked it can be restored by copying it again. Similarly, if a ticket obtained by a demand operation is revoked it can be restored by demanding it again. However, if a ticket introduced by the create rules get revoked it cannot be restored by repeating the create operation since each create operation is unique in that it introduces a unique new entity. Tickets distributed in the initial state also may not be restorable if revoked. Thus the restoration principle appears to have an impact only on tickets distributed in the initial state and tickets introduced as the result of a create operation. If we assume such tickets are irrevocable the restoration principle does not entail any loss of generality.

The restoration principle merely states that a variety of revocation policies are consistent with a SSR scheme. The problem of specifying revocation policies still remains. For now we have chosen to set aside this problem and focus on policies for distribution of tickets rather than on policies for revocation. A crucial consequence of the restoration principle is that in a worst-case scenario, where we assume all subjects will cooperate with one another, we can assume no revocation occurs since tickets which get revoked can be restored.

3. PROJECT CONTROL: SINGLE TEAM

We now develop a case study to demonstrate the expressive power of the SSR model, as well as the role of authorization policies, in the context of a project requiring preparation and use of documents by members of a team. We begin by considering a single team working in isolation. We classify team members into two types: **sup** for supervisors and **wor** for workers. We distinguish three types of documents:

1. Permanent documents, of type **pdoc**, represent products which have satisfactorily undergone quality assurance procedures. Only supervisors can create these and are responsible for the quality. Permanent documents can be consulted by all members. They should be modified infrequently if at all and we rely on the judgment of supervisors in this respect. Supervisors may modify permanent documents themselves or designate a worker to do so.

2. Working documents, of type **wdoc**, are used for communication among team members. Members with the need to work together on some aspect of the project should be able to share working documents. Here again we trust the judgment of supervisors. Workers need not, and should not, be able to share working documents at random with other workers.

3. Supervisory documents, of type **sdoc**, are used to communicate among supervisors for project control and review purposes. These documents cannot be accessed by workers.

We now formulate these requirements in the SSR framework, filling in missing details as we proceed. The protection types have already been identified as follows.

TS = {**sup, wor**}
TO = {**pdoc, wdoc, sdoc**}

The policy concerning **pdoc**'s is selective regarding the ability to modify such documents. This suggests we need to distinguish two kinds of right symbols: v-rights authorizing operations that return some value without modifying the document, and o-rights authorizing operations that modify the document. Since the policy does not distinguish among v-rights, or among o-rights, we need introduce only these two symbols and their copiable variants, i.e.,

RI = {v, o, vc, oc}

Turning to the demand function, we authorize supervisors and workers to demand tickets of type **pdoc**/v so they may consult every permanent document. Further since supervisors may modify **pdoc**'s and also designate workers to do so, we authorize supervisors to demand tickets of type **pdoc**/oc. For inert rights the demand function is then specified as follows.

1. **pdoc**/v, **pdoc**/oc ∈ **d**(**sup**)
2. **pdoc**/v ∈ **d**(**wor**)

In regard to control rights the policy makes no explicit statement. The security administrator would need to clarify the exact intent. One possibility is that supervisor-supervisor, supervisor-worker, worker-supervisor links be established on demand whereas worker-worker links cannot exist. This is achieved as follows, assuming no worker-worker links exist in the initial state.

1. **sup**/sr, **wor**/sr ∈ **d**(**sup**)
2. **sup**/sr ∈ **d**(**wor**)

Next consider the filter function. Since control tickets are obtained on demand we need only consider inert tickets. Supervisors may share all types of documents. Access of **pdoc**'s by supervisors has already been specified by means of the demand function. To facilitate sharing of **wdoc**'s and **sdoc**'s among supervisors as needed, we define

1. **f**(**sup**,**sup**) = {**wdoc**/voc, **sdoc**/voc}

That sharing of **wdoc**'s between workers and supervisors is unrestricted but requires approval of a supervisor between workers and workers, and that the right to modify a **pdoc** may be passed from supervisors to workers is specified as follows.

2. **f**(**sup**,**wor**) = {**wdoc**/voc, **pdoc**/o}
3. **f**(**wor**,**sup**) = {**wdoc**/voc}
4. **f**(**wor**,**wor**) = ∅

A ticket of type **wdoc**/vc or **wdoc**/oc can then be copied from a worker to a supervisor, along with the copy flag. Such a ticket can be further copied from the supervisor to another worker. This enables the sharing of **wdoc**'s among workers at the discretion of supervisors.

Regarding creation of objects we interpret the policy to be that supervisors can create **wdoc**'s, **pdoc**'s and **sdoc**'s, whereas workers may only create **wdoc**'s. This is consistent with the role of these document types. Regarding creation of subjects, workers should not create supervisors otherwise the control imposed by supervisors is subverted. Also, it is clearly useful for supervisors to create new workers. Not so evident is the utility of allowing supervisors to create new supervisors. Firstly, this provides a means of introducing new supervisors in the project. More significantly it is an useful organizational device, e.g., a supervisor responsible for several activities may want to create a separate supervisory subject for each activity. For the latter organizational reason it is also useful to allow workers to create workers. This is reminiscent of TOPS-20, UNIX, and other operating systems where users can create sub-directories.

It remains to define the create rules. Both **sdoc**'s and **wdoc**'s are shared by copying tickets from one domain to another. When one of these documents, say X, is created the creator gets the X/oc and X/vc tickets so these tickets can be copied. For uniformity we apply the same rule to creation of **pdoc**'s although the demand function already provides much the same effect. For creation of subjects the create rule need not introduce any tickets, since links between subjects are established on demand.

To summarize we have defined an authorization scheme for project documentation control as follows.

1. **TS** = {**sup**, **wor**}
   TO = {**wdoc**, **pdoc**, **sdoc**}
2. **RI** = {v, o, vc, oc}
3. **d**(**sup**) = {**sup**/sr, **wor**/sr, **pdoc**/v, **pdoc**/oc}
   d(**wor**) = {**sup**/sr, **pdoc**/v}
4. **f**(**sup**,**sup**) = {**wdoc**/voc, **sdoc**/voc}
   f(**sup**,**wor**) = {**wdoc**/voc, **pdoc**/o}
   f(**wor**,**sup**) = {**wdoc**/voc}
   f(**wor**,**wor**) = ∅
5. **cc** = {<**sup**,**wdoc**>, <**sup**,**pdoc**>, <**sup**,**sdoc**>, <**sup**,**sup**>, <**sup**,**wor**>, <**wor**,**wdoc**>, <**wor**,**wor**>}
6. The create rule for creation of objects is that when an object X is created the creator gets X/vc and X/oc. The create rule for creation of subjects does not introduce any tickets.

This is a simple and practical authorization policy for project documentation control. It is sobering to realize that most operating systems would be hard put to support even this simple policy.

So far we have not considered relationships among documents. Specifically that a document may reference other documents. Consider the policy that access to a document implies access to referenced documents, e.g., access to a software module implies access to referenced modules. To an extent our scheme provides this facility for **pdoc**'s referenced within other **pdoc**'s, since both supervisors and workers are authorized to demand all tickets of type **pdoc**/v. But even with respect to **pdoc**'s this facility does not go far enough, e.g., **pdoc** P references **wdoc** W with the understanding that W will eventually be replaced by a **pdoc**. For a subject S with access to P to automatically obtain access to W in this situation, a ticket for W must be embedded in P and S must have some means of obtaining this ticket from P. SSR is capable of expressing such policies. However, this facility is not feasible if documents are modeled as objects. So for this paper, tickets to access documents referenced within a document must be independently obtained.

VARIATIONS

It is possible to modify our scheme in several ways without altering the policy. For instance the filter function does not allow copying of any tickets from one worker to another. Thereby it makes no difference if we authorize worker-worker links to be established on demand. Moreover the filter function does not allow any copying of control tickets, so the copy flag on control tickets has no significance in this scheme. But then, the following modifications to our scheme leave the policy unchanged.

1. **sup**/src, **wor**/src $\leftarrow$ **d**(**sup**)
2. **sup**/src, **wor**/src $\leftarrow$ **d**(**wor**)

This modification makes explicit the fact that control tickets essentially play no role. Recognition of this fact can have a significant impact on the implementation of the policy.

A second variation arises by recognizing that if worker-worker links cannot exist we may define **f**(**wor**,**wor**) to whatever value we find convenient. In particular if we set **f**(**wor**,**wor**) to {**wdoc**/voc}, the modified scheme has the property that **wdoc**/voc is present in all values of **f**. Then there is no need to perform any checking with respect to the filter function when copying tickets of type **wdoc**/voc from one subject to another.

As a third and final variation, observe that since tickets of type **pdoc**/v can be demanded by all subjects there is no need to copy a ticket of type **pdoc**/v from one subject to another. But then, it does not matter if allow supervisors to demand tickets of type **pdoc**/vc rather than **pdoc**/v. Combining this with our earlier observation that the copy flag on control tickets has no significance we can modify **d**(**sup**) to be {**sup**/src,**wor**/src,**pdoc**/voc}. Now all ticket types not in **f**(**sup**,**sup**) can actually be demanded by supervisors. Hence it does not matter if we allow tickets of these types to be copied from one supervisor to another. But then we can define **f**(**sup**,**sup**) to be **TXR**, so there is no need to do any checking with respect to the filter function when tickets are copied from one supervisor to another.

The trade-offs between these equivalent variations will depend on the run-time mechanism used for implementing the policy. We emphasize, this ability to specify the same policy in alternate ways in SSR is a significant asset of the model and allows for investigation of implementation trade-offs.

Next consider some variations which change the policy in a significant way. The requirement that workers share **wdoc**'s only as approved by supervisors, is enforced in our scheme by intervention of a supervisor on every occasion a ticket is copied from one worker to another. A less restrictive policy is to allow worker-worker links to be established by supervisors. A supervisor still intervenes to establish the link but need not intervene thereafter. We achieve this by allowing supervisors to demand copiable control tickets for workers and transfer these to the domains of workers. The following additions to our scheme account for this change.

1. **wor**/src $\leftarrow$ **d**(**sup**)
2. **wor**/sr $\leftarrow$ **f**(**sup**,**wor**)
3. **f**(**wor**,**wor**) = {**wdoc**/vo}

Let S be a supervisor and X, Y be workers. S can demand the tickets X/sc, X/rc and Y/sc. X, Y can demand the ticket S/r. This results in link(S,X) and link(S,Y). Now, Y/s can be copied from S to X and X/r from S to Y thereby establishing link(X,Y). Tickets of types **wdoc**/vo can then be copied from X to Y. However, Y cannot obtain copiable tickets from X. The net effect is that, without further supervisor intervention, only those workers with direct links between them can share **wdoc**'s. A more liberal policy is obtained by setting **f**(**wor**,**wor**) to {**wdoc**/voc}, so that tickets for **wdoc**'s can be copied along a sequence of links between workers. We might also modify the <**wor**,**wor**> create rule so that when a worker W creates a new worker W', link(W,W') and link(W',W) are immediately established.

The former set of variations, which left the earlier policy unchanged, may or may not preserve these modified policies. Indeed the first variation does not preserve the modified policies since now worker-worker links have some significance. The second variation is subsumed by the more liberal modification where **f(wor,wor)** is {**wdoc**/voc} but is inconsistent with the less liberal modification where **f(wor,wor)** is {**wdoc**/vo}. Finally, the third variation does preserve the modified policies.

Having seen how such variations, both policy-preserving and policy-modifying, are readily accommodated we will continue with the scheme defined and summarized earlier.

<u>REVOCATION</u>

Consider the consequences of the restoration principle on revocation polices in the context of our scheme. Since tickets of type **pdoc**/v can be demanded by all subjects, any policy regarding revocation of such tickets is acceptable. Even the absurd policy that every subject can revoke tickets of type **pdoc**/v from every domain. The situation with regard to tickets of type **pdoc**/o:c is similar. Again it is acceptable, albeit absurd, that a worker may revoke such a ticket in another worker's domain. Realistically, only a supervisor should be allowed to revoke a ticket of type **pdoc**/o in a worker's domain. Indeed perhaps only the supervisor who gave the ticket to the worker should be allowed to revoke it.

Next consider tickets of type **wdoc**/vo:c. The creator X of a **wdoc** W gets the W/voc tickets immediately on creation. Tickets W/vo:c in the domain of any subject Y other than the creator X are obtained by copying. Any policy for revocation of the tickets W/vo:c in the domain of $Y \neq X$ is then acceptable. Again this allows for an absurd policy that any subject can revoke a ticket for a **wdoc** from the domain of any subject other than the creator of the **wdoc**. Realistically, perhaps only the creator of **wdoc** W should be allowed to revoke tickets for W. Or perhaps, only the subject from whose domain the ticket was copied should be allowed to revoke the copy. At the same time, the restoration principle rules out any policy which allows revocation of the original ticket in the creator's domain since there is no means for restoring this ticket once destroyed.

To summarize, the only significant constraint imposed by the restoration principle on revocation policies for the single project scheme is that tickets obtained as a result of creating **wdoc**'s and **sdoc**'s are irrevocable. Any policy regarding the revocation of all other tickets is acceptable.

4. PROJECT CONTROL: MULTIPLE TEAMS

Next we generalize the context to several projects, say N projects identified as 1 through N. If there is no sharing of documents across different projects we can simply extend the single project case by defining different supervisor, worker, and document types for each project, and follow our earlier policy within each project. This is expressed by the scheme below, which amounts to a collection of N independent schemes corresponding to N isolated project teams.

1. **TS** = {**sup.i**, **wor.i**| i=1..N}
   TO = {**wdoc.i**, **pdoc.i**, **sdoc.i**| i=1..N}

2. **RI** = {v, o, vc, oc}

3. For i=1..N,
   d(sup.i) = {**sup.i**/sr, **wor.i**/sr, **pdoc.i**/v, **pdoc.i**/oc}
   d(wor.i) = {**sup.i**/sr, **pdoc.i**/v}

4. For i=1..N,
   f(sup.i,sup.i) = {**wdoc.i**/voc, **sdoc.i**/voc}
   f(sup.i,wor.i) = {**wdoc.i**/voc, **pdoc.i**/o}
   f(wor.i,sup.i) = {**wdoc.i**/voc}
   f(wor.i,wor.i) = $\emptyset$

   For i,j=1..N, $i \neq j$
   f(sup.i,sup.j) = $\emptyset$
   f(sup.i,wor.j) = $\emptyset$
   f(wor.i,sup.j) = $\emptyset$
   f(wor.i,wor.j) = $\emptyset$

5. **cc** = {<**sup.i**,**wdoc.i**>, <**sup.i**,**pdoc.i**>, <**sup.i**,**sdoc.i**>, <**sup.i**,**sup.i**>, <**sup.i**,**wor.i**>, <**wor.i**,**wdoc.i**>, <**wor.i**,**wor.i**>| i=1..N}

6. The create rule for creation of objects is that when an object X is created the creator gets the X/vc and X/oc tickets. The create rule for creation of subjects does not introduce any tickets.

Now consider interaction between project teams by sharing of documents across projects. Specifically the policy that only permanent documents of a project may be shared with other projects. In its most liberal interpretation, this policy is specified by authorizing all types of subjects to demand tickets for all types of permanent documents, as follows.

1. **pdoc.i**/v $\leftarrow$ **d(sup.j)**, i,j=1..N

2. **pdoc.i**/v $\leftarrow$ **d(wor.j)**, i,j=1..N

The policy regarding the creation and modification of **pdoc.i**'s remains unchanged and under control of corresponding **sup.i**'s.

But what if the sharing of permanent documents across projects is to be selective? For complete generality, the supervisors of project i should be able to select any arbitrary subset of the permanent documents of project i as being accessible by any arbitrary subset of project teams. We consider two distinct approaches for specifying this policy in SSR.

DEMAND BASED SOLUTION

Our first solution is based on the demand function. For each i, we refine **pdoc.i** into 2^{N-1} types designated **pdoc.i.M** where M is a subset of $\{k|k=1..N,k\neq i\}$. The idea is that documents of type **pdoc.i.M** are accessible by members of project $j\neq i$ if and only if $j\in M$. Within project i there is no distinction between the types **pdoc.i.M** for different M's. The policy in the latter regard is easily restated as follows: for i=1..N replace every occurrence of

1. **pdoc.i** by **pdoc.i.M**, $\forall M$
2. **pdoc.i**/x:c by **pdoc.i.M**/x:c, $\forall M$
3. <**sup.i,pdoc.i**> by <**sup.i,pdoc.i.M**>, $\forall M$

where the quantifier $\forall M$ signifies all subsets of $\{k|k=1..N,k\neq i\}$. The policy regarding access across projects is then stated by further enhancing the demand function as follows, for i,j=1..N.

1. **pdoc.i.M**/v $\in$ **d(sup.j)** iff $j \in M$
2. **pdoc.i.M**/v $\in$ **d(wor.j)** iff $j \in M$

For each project this solution introduces $2^{N-1}-1$ new object types, thereby increasing the total number of object types by $N*(2^{N-1}-1)$.

LIBRARY BASED SOLUTION

The second solution is based on the notion of a library which is a repository of privileges for sharing documents across projects. We introduce 2^N-1 new subject types designated **lib.M** where M is a non-empty subset of $\{k|k=1..N\}$. In this context we will understand the quantifier $\forall M$ to signify all non-empty subsets of $\{k|k=1..N\}$. The idea here is that only supervisors can insert tickets into instances of all **lib.M**'s, and that members of project j can obtain tickets from instances of **lib.M** if and only if $j\in M$. We assume the security administrator sets up the initial state to include one subject LIB.M of each type **lib.M** and that each LIB.M will cooperate in actions required to insert and distribute tickets. To facilitate insertion of tickets into libraries we authorize supervisor-library links to be established on demand, i.e., for all i=1..N and $\forall M$,

1. **lib.M**/s $\in$ **d(sup.i)**
2. **sup.i**/r $\in$ **d(lib.M)**

We authorize subjects of type **sup.i** to demand tickets of type **pdoc.i**/vc rather than just **pdoc.i**/v. For all i=1..N and $\forall M$ we define **f(sup.i,lib.M)** to be {**pdoc.i**/vc}. It is then entirely up to the discretion of each subject of type **sup.i** to place a ticket of type **pdoc.i**/vc in the domain of any LIB.M.

For distribution of tickets from libraries we present two variations. In the first variation selectivity in distribution is enforced entirely by the filter function defined as follows, for all i=1..N and $\forall M$,

1. If $i \in M$ then
   f(lib.M,sup.i) = {**pdoc.j**/v | j=1..N}
   f(lib.M,wor.i) = {**pdoc.j**/v | j=1..N}
2. If $i \notin M$ then
   f(lib.M,sup.i) = $\emptyset$
   f(lib.M,wor.i) = $\emptyset$

Here we authorize all possible library-supervisor and library-worker links to be established on demand, i.e., for all i=1..N and $\forall M$,

1. **sup.i**/s, **wor.i**/s $\in$ **d(lib.M)**
2. **lib.M**/r $\in$ **d(sup.i)**
3. **lib.M**/r $\in$ **d(wor.i)**

Observe that many of the links which can be thus established are quite useless in that the filter function does not permit any copying of tickets across them.

In the second variation selectivity in distribution is enforced entirely by the demand function. Here we authorize only those library-supervisor and library-worker links where $i\in M$ to be established on demand, i.e., for all i=1..N and $\forall M$,

1. **sup.i**/s, **wor.i**/s $\in$ **d(lib.M)** iff $i \in M$
2. **lib.M**/r $\in$ **d(sup.i)** iff $i \in M$
3. **lib.M**/r $\in$ **d(wor.i)** iff $i \in M$

Now we can define the filter function uniformly as follows, for all i=1..N and $\forall M$,

1. **f(lib.M,sup.i)** = {**pdoc.j**/v | j=1..N}
2. **f(lib.M,wor.i)** = {**pdoc.j**/v | j=1..N}

In both variations, the net effect is that subjects of type **sup.i** or **wor.i** can obtain tickets of all types **pdoc.j**/v, j=1..N from only those instances of **lib.M**'s for which $i\in M$. There is an obvious third variation obtained by taking the filter function from the former and the demand function from the latter.

COMPARISON

The demand based solution increases the number of object types by $N*(2^{N-1}-1)$, while the library based solution increases the number of subject types by 2^N-1. The exponential factor in the number of new types in either solution is a direct consequence of the extremely detailed selectivity. In practise we would rarely need to distinguish all possible subsets of project teams. Any desired degree of selectivity can be specified by constraining the values of M. For instance, our specifications reduce to an all-or-none policy by constraining M to be either $\emptyset$ or $\{k|k=1..N,k\neq i\}$ for the demand based solution, and M to be exactly $\{k|k=1..N\}$ for the library based solution. This amounts to distinguishing two kinds of permanent documents for each project, those internal to the project and those accessible by all project teams. Note that the demand based solution introduces N new object types for this policy, while the library-based solution introduce only one new subject type. In general, for every admissible value of M the demand based solution requires N object types **pdoc.i.M**, i=1..N, while the library based solution requires one subject type **lib.M**. Hence, whatever the degree of selectivity provided there will be approximately a factor of N difference in the number of new types required for the demand based solution as compared to the library based solution.

The library based solution is more convenient regarding incremental changes in the access status of permanent documents. Let P be a permanent document of project i. To begin with the supervisors of project i may decide not to place the ticket P/vc in any library. At some later point they may decide to place P/vc in say LIB.M so P is accessible to members of all projects j in M. Thereafter, they may decide to enlarge the access by placing P/vc in LIB.N. Now P is accessible to members of projects j for j in M union N. In the demand based solution we would need to change the type of P at every step in this incremental process. Thus we would begin by creating P to be of type **pdoc.i.$\emptyset$**, then change the type of P to **pdoc.i.M**, and finally change the type of P to **pdoc.i.(M+N)** where + denotes union. This is fine, except our assumption of strong typing does not allow the type of P to be changed. The type change can be approximated by creating a new document with content identical to P, whenever the type of P needs to be changed. That is we begin by creating P to be of type **pdoc.i.$\emptyset$**, then create P1 of type **pdoc.i.M**, and finally create P2 of type **pdoc.i.(M+N)**. This fix has the drawback that tickets for P will not refer to P1 or P2 thereby presenting the potential for consistency problems. In essence the demand based solution requires a new version of a permanent document to accomplish an incremental change in its access status.

Finally, the library based solution has a possible advantage in that any number of subjects of each type **lib.M** can exist, so documents shared across projects can be grouped by some criteria, e.g., specifications, code, test data etc. Also the library based solution is easily modified to allow creation of new subjects of each type **lib.M**, by placing <**sup.i,lib.M**> in **cc** for i=1..N, so that new groupings can be created as needed. The demand based solution must rely entirely on naming conventions for these effects.

5. CONCLUSION

Consider how SSR meets our twin goals of generality and analyzability. SSR provides a convenient formalism and framework for stating value-based, selective, cooperative, and discretionary policies both in the authorization scheme and in the initial state of a system. Moreover, the ability to specify the same policy in alternate ways in SSR is a significant asset when investigating implementation trade-offs. Strong typing is a major assumption but can be circumvented by treating type changes as the creation of a new entity, i.e., a change in entity A's type from **a** to **b** is viewed as creation of a new entity B of type **b**. The problem is that tickets for A no longer refer to B. If a change in an entity's type is accompanied by revocation of existing tickets, strong typing is adequate. Indeed, in our case study when a working document is made permanent this is exactly what happens.

The policies discussed in this paper are simple enough so their specification can be understood without the need for formal analysis as discussed in Sandhu.[9] But analysis is an important objective of the model. A typical analysis question is phrased as follows: assuming complete cooperation from all subjects, can subject X acquire the ticket Y/z:c? In a variety of special cases of SSR an exact answer of "yes" or "no" to such questions can be obtained. In general, our analysis techniques will answer the question as "yes", "no", or "maybe" and are approximations because of the "maybe". A good approximation should answer "maybe" only occasionally. Approximations are useful when exact analysis is expensive or infeasible. Whether exact answers can be obtained in general is an open question. Moreover, local approximations[8] are possible since the authorization scheme is independent of a specific state and does not change.

As stated above, the analysis question assumes complete cooperation among all subjects. A significant consequence of this worst-case viewpoint is that the analysis remains unchanged in the presence of any revocation policy consistent with the restoration principle. If tickets which can be revoked can always be restored, in the worst-case we may assume no revocation occurs. Moreover in SSR the worst-case assumption can actually be relaxed without restating the question, by modifying the authorization scheme. It is immaterial whether the constraints imposed by the authorization scheme are actually enforced at run time or assumed as trusted or verified behavior on behalf of subjects of a specific type. This is a significant property of the model. Not only does this allow a given system to be studied under different assumptions concerning the behavior of subjects, but also allows a variety of implementations involving different degrees of trust for a given scheme.

The analysis question stated above is formulated in terms of specific entities. It is certainly an important goal for a protection model that such questions be efficiently answered. However, much as in the specification of a policy, in analysis the more significant concern is with classes of entities rather than specific individuals, e.g., can a subject of type **a** obtain a ticket of type **y**/x:c without cooperation of a subject of type **b**? An example in the context of our case study is: can a subject of type **wor.i** obtain a ticket of type **pdoc.i**/o without cooperation of a subject of type **sup.i**? Since our schemes were carefully constructed to avoid this possibility, obviously the answer is "no". Herein lies a crucial contribution of our work. By forcing the security administrator to confront the more significant policy aspects while specifying the scheme, we have reduced the more significant analysis questions to analysis of the scheme. Since the scheme is static its analysis is easier than the analysis of specific initial protection states.

Our final comment concerns the use of send and receive rights for defining the link relation. While there are good reasons for using these specific rights,[8,9] the concept of an authorization scheme is applicable to other sets of rights, e.g., to the the take and grant rights[5] to obtain the Schematic Take-Grant model. Indeed, the policy specifications of this paper could be rewritten for such a model with minor alterations.

REFERENCES

1. Denning, D.E., and Denning, P.J., ``Data Security,'' Comp. Surv., Vol. 11, No. 3, 1979, pp. 227-249.

2. Graham, G.S., and Denning, P.J., ``Protection - Principles and Practice,'' AFIPS, Vol. 40, 1972, pp. 417-429.

3. Linden, T.A., ``Operating System Structures to Support Security and Reliable Software,'' Comp. Surv., Vol. 8, No. 4, 1976, pp. 409-445.

4. Harrison, M.H., Russo, W.L., and Ullman, J.D., ``Protection in Operating Systems,'' Comm. ACM, Vol. 19, No. 8, 1976, pp. 461-471.

5. Jones, A.K., Lipton, R.J., and Snyder, L., ``A Linear Time Algorithm for Deciding Security,'' Proc. 17th Symp. Foundations of Comp. Sci., 1976, .

6. Snyder, L., ``Formal Models of Capability-Based Protection Systems,'' IEEE Trans. Comp., Vol. C-30, No. 3, March 1981, pp. 172-181.

7. Lockman, A., and Minsky, N., ``Unidirectional Transport of Rights, and Take-Grant Control,'' IEEE Trans. Software Eng., Vol. SE-8, No. 6, November 1982, pp. 597-604.

8. Minsky, N., ``Selective and Locally Controlled Transport of Privileges,'' ACM TOPLAS, To Appear, .

9. Sandhu, R.S., Design and Analysis of Protection Schemes Based on the Send-Receive Transport Mechanism, PhD Thesis. Rutgers University Technical Report DCS-TR-130., 1983.

10. Cohen, E., and Jefferson, D., ``Protection in the Hydra Operating System,'' Proc. 5th Symp. O.S. Princ., 1975, pp. 141-160.

11. Dennis, J.B., and Van Horn, F.C., ``Programming Semantics for Multiprogrammed Computations,'' Comm. ACM, Vol. 9, No. 3, 1966, pp. 143-155.

12. Fabry R., ``Capability-Based Addressing,'' Comm. ACM, Vol. 17, No. 7, 1974, pp. 403-412.

13. Wilkes, M.V., and Needham, R.M., The Cambridge CAP Computer and its Operating System, Elsevier North Holland, 1979.