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. 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. Insert 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 is the 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...
The next step is to define the right approach to the problem. The approach is critical, and the strategy for implementing it can determine the success or failure of the project. The next step is to define the scope and objectives of the project. This includes identifying the stakeholders, their needs, and the resources available. The project must be scoped to meet these needs and align with the organization's goals.

The next step is to define the scope and objectives of the project. This includes identifying the stakeholders, their needs, and the resources available. The project must be scoped to meet these needs and align with the organization's goals. The next step is to define the scope and objectives of the project. This includes identifying the stakeholders, their needs, and the resources available. The project must be scoped to meet these needs and align with the organization's goals.

The next step is to define the scope and objectives of the project. This includes identifying the stakeholders, their needs, and the resources available. The project must be scoped to meet these needs and align with the organization's goals. The next step is to define the scope and objectives of the project. This includes identifying the stakeholders, their needs, and the resources available. The project must be scoped to meet these needs and align with the organization's goals.
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.,

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

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

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

where dom denotes the set of tickets possessed by the indicated subject. The existence of \(\text{link}(A,B)\) 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 \(t(Y)/x/c\) are different ticket types. This is an important distinction because of the role of the copy flag. The entire set of ticket types is \(\text{TXY}\).

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 \(\text{TS, TO, and TXR}\). 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.,

$$f: \text{TS} \times \text{TS} \rightarrow \text{2TXR}$$

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

1. \(Y/x/c \in \text{dom}(A)\),
2. \(\text{link}(A, B)\) exists,
3. \(t(Y)/x/c \in f(t(A), t(B))\).

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.,

$$d: \text{TS} \rightarrow \text{2TXR}$$

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

1. If \(b/s \in d(a)\) then every subject \(A\) of type \(a\) can demand \(B/s\) for every subject \(B\) of type \(b\). The definition of \(\text{link}(A,B)\) for such subjects effectively reduces to \(A/r \in \text{dom}(B)\).
2. Correspondingly, if \(a/r \in d(b)\) then for subjects \(A, B\) of types \(a, b\) respectively the definition of \(\text{link}(A,B)\) effectively reduces to \(B/s \in \text{dom}(A)\).
3. If both \(b/s \in d(a)\) and \(a/r \in d(b)\) then for subjects \(A, B\) of types \(a, b\) respectively \(\text{link}(A,B)\) 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 \text{can-create-relation} \(cc\) which relates subject types to types, i.e.,

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

The interpretation is that subjects of type \(a\) are authorized to create entities of type \(b\) if and only if \(\langle a, b \rangle \in cc\). 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 \(\langle a, b \rangle\) create rule tells us which tickets for \(B\) are placed in \(A\)'s domain as a result of this operation. If \(B\) is a create rule, 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 introduce the concept of "local incremental impact on the state" and show how it can be used to identify specific points in the process that need to be changed. This allows us to focus on the most critical aspects of the system, rather than making changes that are unlikely to have a significant impact.

In summary, the SRE model requires the introduction of a set of security policies, each with its own set of rules, to manage the interactions between different parts of the system. The SRE model is designed to provide a clear and consistent way to manage these interactions, and to ensure that the system remains secure and reliable.

2. The set of security policies: The set of security policies is fixed, as are the set of security policies of the entities in the system. The set of security policies can be modified, however, by adding new policies or removing old ones.

3. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

4. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

5. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

6. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

7. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

8. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

9. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

10. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

11. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

12. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

13. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

14. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

15. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

16. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

17. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

18. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

19. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

20. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

21. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

22. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

23. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

24. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

25. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

26. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

27. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

28. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

29. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.

30. The demand for tickets: The demand for tickets is determined by the set of entities and the set of tickets that are required to access the system. The demand for tickets can be satisfied by either creating new tickets or by using existing tickets.
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.,

\[ RL = \{ v, o, vc, oc \} \]

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 \in d(sup)
2. pdoc/v \in 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 \in d(sup)
2. sup/sr \in 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) = \emptyset

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 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) = \emptyset
5. cc = \{ sup/wdoc, sup/pdoc, sup/sdoc, sup/sdoc, sup/wor, wdoc, 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. SSS 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 if the filter function does not allow copying of any tickets from one worker to another. Then 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. \( \text{sup/src, wor/src} \leftrightarrow \text{d(sup)} \)

2. \( \text{sup/src, wor/src} \leftrightarrow \text{d(wor)} \)

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 \( f(\text{wor, wor}) \) to whatever value we find convenient. In particular if we set \( f(\text{wor, wor}) \) to \( \text{wdoc/voc} \), the modified scheme has the property that \( \text{wdoc/voc} \) 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 \( \text{wdoc/voc} \) from one subject to another.

As a third and final variation, observe that since tickets of type \( \text{pdoc/v} \) can be demanded by all subjects there is no need to copy a ticket of type \( \text{pdoc/v} \) from one subject to another. But then, it does not matter if allow supervisors to demand tickets of type \( \text{pdoc/vc} \) rather than \( \text{pdoc/v} \). Combining this with our earlier observation that the copy flag on control tickets has no significance we can modify \( d(\text{sup}) \) to be \( \text{sup/src, wor/src, pdoc/voc} \). Now all ticket types not in \( f(\text{sup, sup}) \) 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 \( f(\text{sup, sup}) \) to be \( \text{TR} \), 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 different 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 copyable control tickets for workers and transfer these to the domains of workers. The following additions to our scheme account for this change.

1. \( \text{wor/src} \leftrightarrow \text{d(sup)} \)

2. \( \text{wor/src} \leftrightarrow \text{f(sup, wor)} \)

3. \( f(\text{wor, wor}) = \text{wdoc/voc} \)

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 \( \text{wdoc/voc} \) can then be copied from \( X \) to \( Y \). However, \( Y \) cannot obtain copyable tickets from \( X \). The net effect is that, without further supervisor intervention, only those workers with direct links between them can share \( \text{wdoc's} \). A more liberal policy is obtained by setting \( f(\text{wor, wor}) \) to \( \text{wdoc/voc} \), so that tickets for \( \text{wdoc's} \) can be copied along a sequence of links between workers. We might also modify the \( \text{<wor, wor> create rule} \) 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 varia-
tion does not preserve the modified policies since new worker working links have some significance. The second variation is subsumed by the more liberal modification where \( f(\text{wor}, \text{wor}) \) is \((\text{wdoc}/\text{voc})\) but is inconsistent with the less liberal modification where \( f(\text{wor}, \text{wor}) \) is \((\text{wdoc}/\text{woc})\). 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.

**REVCATION**

Consider the consequences of the restoration principle on revocation policies in the context of our scheme. Since tickets of type \( \text{pdoc}/\text{v} \) 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 \( \text{pdoc}/\text{v} \) from every domain. The situation with regard to tickets of type \( \text{pdoc}/\text{c} \) 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 \( \text{pdoc}/\text{c} \) 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 \( \text{wdoc}/\text{voc} \). The creator \( X \) of a \( \text{wdoc} \) \( W \) gets the \( W/\text{voc} \) tickets immediately on creation. Tickets \( W/\text{voc} \) in the domain of any subject \( Y \) other than the creator \( X \) are obtained by copying. Any policy for revocation of the tickets \( W/\text{voc} \) 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 \( \text{wdoc} \) from the domain of any subject other than the creator of the \( \text{wdoc} \). Realistically, perhaps only the creator of \( \text{wdoc} \) \( W \) 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 \( \text{wdoc} \)’s and \( \text{sdoc} \)’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. \( \text{TS} = \{\text{sup}_i, \text{wor}_i | i=1..N\} \)
2. \( \text{TO} = \{\text{wdoc}_i, \text{pdoc}_i, \text{sdoc}_i | i=1..N\} \)
3. \( \text{RI} = \{\text{v}, \text{c}, \text{vc}, \text{oc}\} \)
4. For \( i=1..N \), \( d(\text{sup}_i) = \{\text{sup}_i/\text{sr}, \text{wor}_i/\text{sr}, \text{pdoc}_i/\text{v}, \text{pdoc}_i/\text{oc}\} \)
5. For \( i=1..N \), \( f(\text{sup}_i, \text{sup}_i) = \{\text{wdoc}_i/\text{voc}, \text{sdoc}_i/\text{voc}\} \)
6. For \( i=1..N \), \( f(\text{sup}_i, \text{wor}_i) = \{\text{wdoc}_i/\text{voc}, \text{pdoc}_i/\text{oc}\} \)
7. For \( i=1..N \), \( f(\text{wor}_i, \text{sup}_i) = \{\text{wdoc}_i/\text{voc}\} \)
8. For \( i=1..N \), \( f(\text{wor}_i, \text{wor}_i) = \emptyset \)
9. For \( i=1..N \), \( f(\text{sup}_i, \text{sup}_j) = \emptyset \)
10. For \( i=1..N \), \( f(\text{sup}_i, \text{wor}_j) = \emptyset \)
11. For \( i=1..N \), \( f(\text{wor}_i, \text{sup}_j) = \emptyset \)
12. For \( i=1..N \), \( f(\text{wor}_i, \text{wor}_j) = \emptyset \)

The create rule for creation of objects is that when an object \( X \) is created the creator gets the \( X/\text{vc} \) and \( X/\text{oc} \) 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. \( \text{pdoc}_i/\text{v} \in d(\text{sup}_j), i,j=1..N \)
2. \( \text{pdoc}_i/\text{v} \in d(\text{wor}_j), i,j=1..N \)

The policy regarding the creation and modification of \( \text{pdoc}_i/\text{v} \) remains unchanged and under control of corresponding \( \text{sup}_i/\text{v} \)'s.
COMPARISON

The demand based solution increases the number of object types by $N^2(2^{N-1})$, 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 practice 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 0 or $(k-1)\ldots(N-1)$ 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 introduces only one new subject type. In general, for every admissible value of $M$ the demand based solution requires $N$ object types $pdoc.i,M$. 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 document in any library. At some later point they may decide to place $P$ 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 $P$ 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 $P$ of type $pdoc.i,0$, 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 $P$ of type $pdoc.i,0$, then create $P_1$ of type $pdoc.i,M$, and finally create $P_2$ of type $pdoc.i,(M+N)$. This fix has the drawback that tickets for $P$ will not refer to $P_1$ or $P_2$ 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 $cpriv.i,lib.M$ in $M$ for $i=1\ldots N$, 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. But analysis is an important objective of the model. A typical analysis question in the form of asking complete cooperation from all subjects, can subject $X$ acquire the ticket $Y$ if $Z$? 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 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, so that whenever revocation occurs, Moreover in SSR the worst-case assumption is 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.