Extending the Creation Operation in the Schematic Protection Model

P.E. Ammann R.S. Sandhu

Department of Information Systems and Systems Engineering
George Mason University
4400 University Drive
Fairfax, VA 22030-4444

Abstract

Protection models provide a formalism for specifying control over access to information and other resources in a multi-user computer system. Useful protection models must balance expressive power with the complexity of safety analysis, *i.e.* the determination of whether or not a given subject can ever acquire access to a given resource. We argue that, in terms of expressive power, a joint creation operation is a natural candidate for inclusion in a access control model, particularly in the context of integrity considerations. We extend the Schematic Protection Model (SPM) [15] to allow for groups of subjects to jointly create other subjects and objects. We show that the extended model, which we call ESPM, is equivalent in expressive power to the monotonic access matrix model of Harrison, Ruzzo, and Ullman [6]. Additionally, in contrast to [16], we conjecture that SPM is in fact *less* expressive than the monotonic access matrix. Thus the joint creation operation appears to have fundamental expressive power lacking in the SPM model. We discuss the safety properties of ESPM. Despite the increase in expressive power, ESPM retains tractable safety analysis for many cases of practical interest.

1. INTRODUCTION

The *protection problem* is how to control access to information and other resources in a multi-user computer system. *Protection models* provide a formalism for specifying this control. Protection models must satisfy two conflicting requirements:

(1) The need for expressive power sufficient to describe authorization schemes of practical interest.

(2) The need for tractable safety analysis of the propagation of access rights, *i.e.* the determination of whether or not a given subject can ever acquire access to a given resource.

One tolerable restriction on expressive power that can have substantial benefits for safety analysis is that of monotonicity. Since monotonic models do not permit the deletion of access privileges, backtracking in analysis can be avoided.

It should be noted that a strictly monotonic model is too restrictive to be of much practical use, since the ability to delete access privileges is an important requirement. We are really interested in models which can be reduced to monotonic models for purpose of safety analysis. In particular, we can ignore deletion of an access privilege P whenever the deletion can itself be undone by regranting P. This is by far the most common form of revocation and it is indeed fortunate that monotonic models can accommodate such deletion.

Monotonicity by itself is insufficient for tractable safety analysis. The monotonic version of the access matrix model of Harrison, Ruzzo, and Ullman [6] (HRU) has very broad expressive power; unfortunately, despite its monotonicity, it has very weak safety properties [7]. A variety of monotonic[†] models with more desirable safety properties have been proposed [2, 8, 11, 12, 13]. These are all subsumed by Sandhu's Schematic Protection Model [15, 16] (SPM). SPM has remarkably strong safety properties and has been shown to represent a wide variety of cases of practical interest.

In this paper, we argue for the fundamental importance of a mechanism by which subjects can jointly create other subjects and objects. We argue that, in terms of expressive power, a joint creation operation is a natural candidate for inclusion in an access control model, particularly in the context of integrity considerations. We also argue that the inclusion of a joint creation

[†] These models generally include the kind of revocation mentioned earlier where the revocation itself can be undone, i.e., they are monotonic only in the technical sense of being reducible to monotonic operations for the purpose of safety analysis.

operation still permits tractable safety analysis for many cases of practical interest.

The organization of the paper is as follows. In section 2 we provide practical examples to motivate the need for a joint creation operation. These examples are largely driven by integrity considerations (although confidentiality does figure in one instance). Then in section 3 we extend SPM to allow for groups of subjects to jointly create other subjects and objects. In section 4, we show that the extended model, which we call ESPM, is exactly equivalent in theoretical expressive power to the monotonic HRU model. In section 5, we conjecture, in contrast to [16], that SPM is in fact *less* expressive than monotonic HRU. Thus the joint creation operation appears to have fundamental expressive power lacking in the SPM model. Section 6 concludes the paper.

2. MOTIVATION FOR JOINT CREATION

In this section we motivate the utility of a joint create operation from a practical point of view by showing how this operation naturally supports a range of protection policies that have been proposed in the literature. These policies have mostly been proposed in the context of integrity considerations, although there are some aspects that are concerned with confidentiality issues.

Our first example is that of *mutual suspicion*. This was one of the earliest protection problems identified in the literature [5]. The problem arises whenever two users, say A and B, who do not trust each other have to cooperate in achieving some task. The task requires that B has the ability to exercise a subset of A's privileges, and vice versa. The standard solution to this problem [5] is as follows.

- (i) A creates a subject A',
  (ii) A gives A' the privileges that B needs, and
  (iii) A gives B' the indirect privilege for A'.

Similarly:

- (i) B creates a subject B',
  (ii) B gives B' the privileges that A needs, and
  (iii) B gives A' the indirect privilege for B'.

At this point A' and B' act as agents for A and B in achieving their cooperative objective. The idea is that A' can indirectly exercise the privileges of B' and vice versa.

This solution depends critically on indirect privileges and as such requires a new concept and mechanism for its implementation. This concept is not particularly easy to formalize, e.g., we must consider whether or not the indirect privilege can itself be indirectly exercised? The solution is correct only with the specific assumption that chaining of indirection is disallowed. It is clear that weak restrictions on the indirect privilege have disconcerting implications for access review and safety analysis, but strong restrictions amount to building more policy into our model than we really wish. Moreover, the algorithm described above does not give B any unique access to A' since A is free to send indirect privileges for A' to some other subject C.

On the other hand, the joint create operation provides an ideal solution for the mutual suspicion problem. A and B can jointly create a subject C such that A and B become the joint owners of C. Any pattern of communication from A and B to C and from C to A and B can then be specified. In the case of the mutual suspicion problem, A and B can be allowed to contribute privileges to C freely, but must be restricted in their ability to take privileges from C.

It is important to appreciate that once the joint create operation has occurred, the restrictions needed to solve the mutual suspicion problem can all be stated in terms of an operation which copies privileges from one subject to another. Since the copy operation is one of the fundamental operations that any access control model must support, the required ability to specify restrictions on the copy operation is independent of the mutual suspicion problem and joint creation.

Our second application of joint creation is in solving the well-known *protected subsystem problem* [4, 10]. In this problem we again have two parties A and B, where A is a user and B is a service that A wishes to use for some purpose. We model the invocation of B by A as the joint creation of subject C by A and B. Note that B is a passive participant in this act, while A and C are active subjects. Our requirements are as follows:

(1) A can only give data to C and receive results from C. In particular A cannot obtain the rights to directly modify the internal data structures of C (i.e., we have information hiding in the sense of data abstraction).

(2) C can obtain data and code from B.

This differs from our mutual suspicion problem only in regard to what C can obtain from A and B. That is we no longer have a symmetry between A and B. Given the ability to restrict the copy operation it is a simple matter to specify these constraints in a sufficiently general access control model.

The *confinement problem* as originally formulated by Lampson [9] gives us our next application. This problem is actually a particularly stringent variation of the protected subsystem problem with the following added requirement.

(3) C cannot leak to anyone the data given it by A.

This is clearly a confidentiality requirement. Since the code executed by C can only be obtained from B, A is threatened by Trojan Horses in B's code that might leak A's confidential data. To solve this problem we need to ensure that C cannot write to any object other than its internal data structures and objects provided by A.[†]

From the joint creates perspective the three examples above are all variations of the same requirement. The joint child C of A and B is restricted with respect to privileges it can obtain from A or B as well as privileges A or B can obtain from it. In general the restrictions applied to parent A are different from those applied to parent B. A key point is that these restrictions are specified in terms of the copy operation to move privileges from one subject to another. The facility to do so should certainly be available in any access control model that claims generality.

In these examples the critical role of joint creation lies in binding C to its parents A and B at the moment of creation. The question naturally arises whether or not joint creation can be reduced to more primitive operations. In section 4 we will present formal arguments which strongly indicate that joint creation is fundamentally more powerful than the usual single-parent creation employed in access-control models. To appreciate why this happens consider an attempt to mimic the effect of joint creation of C by A and B as follows.

(1) Let A create C (using the usual single-parent creation). This binds A and C in a unique manner.

(2) Establish the B to C binding by copying privileges for C to B and/or vice versa.

The problem with this approach is that the means to achieve step 2 inevitably implies that a similar binding can also be established between C and, say, B'. In other words it appears impossible to uniquely bind C to B without introducing some new mechanism, such as joint create,[‡] for this purpose.

Our final example of joint creation stems from the *separation of duties* concept of Clark and Wilson [3]. Joint creation turns out to be a particularly effective way of specifying separation of duties with respect to creation of new users. As noted in [14, 17] separation of duties is often best expressed in terms of roles such as manager, security-officer, clerk, etc. For simplicity assume that each user has a unique role in the system. The following rules show how joint creation can specify the involvement of distinct users with different roles in the process of enrolling new users in the system.

(1) A manager and a security-officer can jointly create a new clerk.

(2) A senior-manager and a security-officer can jointly create a new manager.

(3) Senior-managers and security-officers can only be created by the system-owner.

In this case the joint creation is more concerned with involving multiple parties in the decision to effect the creation, rather than our earlier examples where the focus was on the unique binding between the child and its multiple parents. Note that while the joint creation operation offers an elegant solution for this last example, it is not strictly necessary. For example, the techniques described in [18] can be used to achieve separation of duties within the conventional framework of single-parent creation.

3. THE ESPM MODEL

In this section we define a formal model which includes a joint creation operation. The model is based on Sandhu's Schematic Protection Model [15] or SPM. By way of introduction, we first review SPM, and then we describe the extensions.

3.1. Review of SPM

In response to the relatively weak safety properties of the access matrix model formalized by Harrison, Ruzzo, and Ullman [6, 7] (HRU), a number of more restricted models with efficient safety analysis were proposed [2, 8, 11, 12, 13]. However, a substantial gap in expressive power exists between these models and HRU. The schematic protection model (SPM) was developed to fill this gap in expressive power while sustaining efficient safety analysis.

SPM is based on the key principle of protection types, henceforth abbreviated as types. SPM subjects and objects are strongly typed, *i.e.*, the type of an entity (subject or object) is determined when the entity is created and does not change thereafter. Types are an abstraction of the intuitive notion of properties that are protection relevant. An SPM scheme is to a large extent, but not exclusively, defined in terms of types. The dynamic privileges in SPM are tickets of the form Y/r where Y identifies some unique entity and r is a right. The notion of type is extended to tickets by defining

[†] Of course, we also need a covert channel analysis to achieve a high level of assurance. In other words we need to make sure that not only the explicit write operations but also the implicit ones have been accounted for.

[‡] We are aware that there are mechanisms other then joint create which can also give us this effect. See our discussion in section 5.4.

type(Y/r) to be the ordered pair type(Y)/r. That is the type of a ticket is determined by the type of entity it addresses and the right symbol it carries.

SPM has only two operations for changing the protection state, viz., create and copy.[†] These operations are authorized by rules which comprise the scheme defined by specifying the following (finite) components.

(1) Disjoint sets of subject types TS and object types TO. Let $T = TS \cup TO$.

(2) A set of rights R. The set of ticket types is thereby $T \times R$.

(3) A can-create function $cc : TS \rightarrow 2^T$.

(4) Create rules of the form $cr_p(u, v) = c/R_1 \cup p/R_2$ and $cr_c(u, v) = c/R_3 \cup p/R_4$

(5) A collection of link predicates $\{link_i\}$

(6) A filter function $f_i : TS \times TS \rightarrow 2^{T \times R}$ for each predicate $link_i$.

An SPM scheme is itself static and does not change.

The Create Operation. Creation is authorized exclusively by types. Subjects of type u can create entities of type v if and only if $v \in cc(u)$. Tickets introduced as the side effect of creation are specified by create-rules. Each create-rule has two components shown above, where p and c respectively denote parent and child and the R_i are subsets of R. When subject U of type u creates entity V of type v the parent U gets the tickets V/R_1 and U/R_2. The child V similarly gets the tickets V/R_3 and U/R_4. For example, $file \in cc(user)$ authorizes *users* to creates *files*. And $cr_p(user, file) = c/rw$ and $cr_c(user, file) = \varnothing$ gives the creator r and w tickets for the created file.

The Copy Operation. A copy of a ticket can be transferred from one subject to another leaving the original ticket intact. Permission to copy a ticket Y/r depends in part on possession of the SPM copy flag, c, for that ticket, denoted Y/rc. Possession of Y/rc implies possession of Y/r but not vice versa. It is possible to copy Y/r only, or to copy Y/rc, in which case the ticket may be further copied. Let $dom(U)$ signify the set of tickets possessed by U. Three independent pieces of authorization are required to copy Y/r from U to V.

(1) $Y/rc \in dom(U)$, *i.e.*, U must possess Y/rc for copying either Y/rc or Y/r.

(2) There is a link from U to V. Links are established by tickets for U and V in the domains of U and V. The predicate $link_i(U, V)$ is defined as a conjunction or disjunction, but not negation, of one or more of the following terms for any $r \in R$: $U/r \in dom(U)$, $U/r \in dom(V)$, $V/r \in dom(U)$, $V/r \in dom(V)$, and **true**. Some examples from the literature are given below [11, 13, 16, respectively]:

$$link_{tg}(U, V) \equiv V/g \in dom(U) \vee U/t \in dom(V)$$
$$link_t(U, V) \equiv U/t \in dom(V)$$
$$link_{sr}(U, V) \equiv V/s \in dom(U) \wedge U/r \in dom(V)$$
$$link_u(U, V) \equiv \textbf{true}$$

(3) The last condition is defined by the filter functions f_i, one per predicate $link_i$. The value of $f_i(u, v)$ specifies types of tickets that may be copied from subjects of type u to subjects of type v over $link_i$. Also f_i determines whether or not the copied ticket can have the copy flag. Example values are $T \times R$, $TO \times R$, and $\varnothing$ respectively authorizing all tickets, object tickets and no tickets to be copied.

In short Y/r can be copied from U to V iff there exists some $link_i$ such that:

$$Y/rc \in dom(U) \wedge link_i(U, V) \wedge y/r \in f_i(u, v)$$

where the types of U, V and Y are respectively u, v and y. To copy Y/rc from U to V, it must also be the case that $y/rc \in f_i(u, v)$.

3.2. Adding Joint Creation to SPM

We extend the creation operation in the SPM model above to enable groups of subjects to jointly create new subjects and objects. We call the extended model ESPM, and we refer to the extended creation operation as *joint creation*, or simply *creation* in those cases where no confusion arises. Joint creation includes the SPM creation operation as a special case. In all other respects, ESPM is identical to SPM.

The (joint) can-create function for ESPM is a mapping:

$$cc : TS \times TS \times \cdots \times TS \rightarrow 2^T$$

In ESPM the domain of cc is an N-tuple of subject types as opposed to a single subject type in the SPM case. ESPM imposes no bound on the maximum value of N, although for any given scheme this value is of course a constant. Further, if type constraints are met, we allow a subject to redundantly participate as more than one parent in a joint create operation. The option of forcing the parents to be unique was rejected because it is contrary to the tone of the SPM copy operation where the

[†] We note that the original definition of SPM [15] included a third operation called demand that has since been shown to be redundant [19].

same subject can participate as the source, destination and reference of the transported ticket. It also seems natural to follow the lead taken in programming languages, where it is the usual practice to allow a single object to replace multiple formal parameters.

It remains to extend the create rules cr_p and cr_c of SPM to describe the distribution of tickets that results from joint creation. We have a variety of choices as to which tickets parents are allowed to acquire as a result of a joint create operation. The most general choice is to allow the cr_p function to supply a parent with arbitrary tickets, not only for itself and the child, but also for any other parent. This option has the undesirable side effect of duplicating the functionality of the links and filter functions. Since we do not want the joint create operation to be a substitute for the copying of tickets, we restrict the cr_p operation such that a parent X does not acquire tickets of the form Y/r for any other parent Y.

With the above restriction in mind, the most general choice for specifying the distribution of tickets as a result of creation is to give a separate cr_p rule for each parent-child pair and a single cr_c rule that allows the child to acquire a different set of tickets from each parent. Formally, we define N create rules of the form:

$$cr_{p_i}(t_{p_1}, t_{p_2}, \ldots, t_{p_N}, t_c) = c/R_1^i \cup p_i/R_2^i \text{ for } i = 1..N$$

and one rule of the form:

$$cr_c(t_{p_1}, t_{p_2}, \ldots, t_{p_N}, t_c) = c/R_3 \cup p_1/R_4^1 \cup p_2/R_4^2 \cup \cdots \cup p_N/R_4^N.$$

The t_{p_i} are the types of the N parents, and the t_c is the type of the child. In all of the create rules c is the name of the jointly created entity and p_i is the name of the *ith* parent. For the ESPM create rules, note that the sets R_1, R_2, and R_4 from the SPM create rules have each been expanded into N sets, R_1^i, R_2^i, and R_4^i, for $i = 1..N$. The set R_3 is unchanged.

4. EXPRESSIVE POWER OF ESPM

We now turn to evaluation of the expressive power of ESPM. The most general monotonic protection model to date is the monotonic access matrix model of Harrison, Ruzzo, and Ullman [6], which we refer to as monotonic HRU. It turns out that ESPM is precisely equivalent to monotonic HRU in its expressive power.

The equivalence result is established by simulating monotonic HRU in ESPM and vice versa. The simulation of HRU in ESPM is by far the more difficult part of this proof. The details are intricate and inevitably tedious and lengthy [1]. Here we only outline the construction, paying special attention to the key role played by the joint creation operation. Because it is straightforward to implement ESPM in HRU, we do not discuss the reverse construction.

4.1. The Monotonic HRU Access Matrix Model

In the monotonic HRU scheme that we model, we consider I HRU commands, each denoted $HRU_i, i = 1..I$, structured as follows.

```
$HRU_i(P_1, \ldots, P_{J_i}, P_{J_{i+1}}, \ldots, P_{J_i+M_i})$
    **if** $T_1^i \wedge T_2^i \wedge \cdots \wedge T_{K_i}^i$ **then**
        $C_1^i$
        $C_2^i$
        ...
        $C_{M_i}^i$
        $E_1^i$
        $E_2^i$
        ...
        $E_{N_i}^i$
    **end**
```

in which:

(1) The P_j, where $j = 1..J_i$, are formal parameters representing existing HRU entities. The P_j where $j = J_{i+1}..J_i+M_i$, are the names of entities to be created by the HRU command.

(2) The T_k^i, where $k = 1..K_i$, are terms of the form "$r \in [P_X, P_Y]$". Note that $[P_X, P_Y]$ is the HRU matrix entry for row P_X and column P_Y. X and Y are in the range $1..J_i$. The absence of disjunctions in the conditional entails no loss of generality since disjunctions can be simulated with multiple HRU commands. Note, however, that negation is disallowed.

(3) The C_m^i, where $m = 1..M_i$, are HRU primitives of the form "Create P_X", where $X = J_i+m$. Since we consider only the monotonic access matrix model, there is no "Delete P_X" operation, and we are free to order the HRU primitives so that all "Create" operations precede all "Enter" operations.

(4) The E_n^i, where $n = 1..N_i$, are HRU primitives of the form "Enter right r in $[P_X, P_Y]$". X and Y are in the range $1..J_i+M_i$.

4.2. Reduction of HRU to ESPM

There are three basic problems in implementing HRU in ESPM. They correspond to the various parts of the HRU command:

(1) Parameter List Generation: HRU commands are invoked with a particular set of entities as parameters. For a valid simulation, it must be

possible to manipulate exactly the set of entities that correspond to any possible HRU parameter list. Thus one task for any simulation of HRU is to mimic the gathering of arbitrary existing HRU entities into sets of cardinality J_i.† However, the joint creation operation of ESPM is ideally suited for the task.

(2) Validating the Conditional: The basic process of an HRU command is to permit the specified operations only if the conjunction of certain conditions is evaluates to **true**. A mechanism is required to determine the validity of each term in the conditional. Another mechanism is required for combining the values of the individual terms.

(3) Implementing Primitive HRU Operations: Simulating "Enter" operations is straightforward in ESPM; it is easy to arrange that the ticket simulating the right in question only be copied only if the conditional evaluates to **true**. However, "Create" operations are another matter. ESPM does not contain a conditional creation operation. Therefore, in showing that an ESPM scheme can simulate an HRU scheme, we must simulate conditional creation. Simulating conditional creation can be accomplished in various ways; we do it by augmenting the HRU scheme with an additional right that indicates that an entity is "alive". Thus, even though we cannot conditionally control the creation of ESPM entities, we can conditionally control the presence of tickets indicating liveness. We must also ensure that entities not marked as being alive do not participate in changing the protection state. Again, various options are available; we choose to augment the conditional expression in an HRU command to ensure that all relevant entities hold a "live" ticket.

4.2.1. Construction Outline

The entities used in the ESPM simulation can be grouped into various categories. Each category is used to mimic part of the evaluation of an HRU command. The categories are defined below.

(1) Entities that mimic HRU entities. We call these entities **proxies**. The simulation arranges that a **proxy** P_X can hold a ticket of the form P_Y/r for some HRU right r iff the HRU access matrix entry $[P_X, P_Y]$ can contain r. The single **proxy** type is p.

(2) Entities to represent existing **proxies** in each possible parameter position of a single HRU command. We call these entities **agents**. The number of types of **agents** is $J_{\max}$, which is the maximum value over the J_i, where $i = 1..I$. The set of **agent** types is $\{a_j \mid j = 1..J_{\max}\}$.

(3) Entities to represent the collection of J_i existing HRU entities in the HRU_i command. We call these entities **validators**. The creation of **validators** is the step that requires the joint creation operation. **validators** assume a coordinating role in the simulation; they are responsible first for overseeing the simulation of HRU conditional evaluation, and then for enabling the simulation of "Create" and "Enter" primitives. There are I types of **validators**, one for each HRU command, HRU_i. The set of **validator** types is $\{v^i \mid i = 1..I\}$.

(4) Entities to collectively determine the truth of the entire conditional expression in an HRU command by examining each conjunct of the conditional in turn. We call these entities **terms**. For each command HRU_i, there are K_i types of **terms**. The set of **term** types is $\{t^i_k \mid k = 1..K_i, i = 1..I\}$.

(5) Entities to implement the HRU primitive "Create P_X". We call these entities **creates**. For each command HRU_i, there are M_i types of **creates**. The set of **create** types is $\{c^i_m \mid m = 1..M_i, i = 1..I\}$.

(6) Entities to implement the HRU primitive "Enter r in $[P_X, P_Y]$". We call these entities **enters**. For each command HRU_i, there are N_i types of **enters**. The set of ESPM types for **creates** is $\{e^i_n \mid n = 1..N_i, i = 1..I\}$.

The simulation operates informally as follows. Existing **proxies** create **agents** to represent them in various parameter positions. For each HRU command HRU_i, groups of J_i **agents** jointly create a **validator**; the effect is to gather **agents** for existing **proxies** into sets of cardinality J_i. These two actions simulate the invocation of an HRU command.

It remains to implement the execution of the HRU command. Each **validator** creates a set of **terms** to determine the validity of the HRU conditional with the selected set of parameters. In addition, the **validator** creates **creates** and **enters** to simulate, respectively, the HRU "Create" and "Enter" primitives. The **enters** are

† We conjecture that SPM is unable to provide this grouping operation; hence, in comparing the expressive power of SPM to ESPM, this is the key stage in the simulation of HRU. This point is discussed further in section 5.2.

enabled only if the entire conditional (comprised of the individual terms) evaluates to **true**. For each **create**, there is an **enter** whose responsibility it is to copy the P_X/e enabling ticket to P_X.

In summary, the types required are $TS = \{p, a_j, v^i, t^i_k, c^i_m, e^i_n\}$. Without loss of generality, we may ignore passive objects and only consider active subjects, and thus set $TO = \varnothing$. The creation relations among these types of entities is shown in fig. 1. Arrows in fig. 1 point from parent types to child types. Formally:

$cc(p) = \{a_j \mid j = 1..J_{\max}\}$
$cc(a_1, a_2, ..., a_{J_i}) = \{v^i\}$, for $i = 1..I$
$cc(v^i) = \{t^i_1\} \cup \{c^i_m \mid m = 1..M_i\} \cup \{e^i_n \mid n = 1..N_i\}$, for $i = 1..I$
$cc(t^i_k) = \{t^i_{k+1}\}$, for $k<K_i$ and $i = 1..I$
$cc(c^i_m) = \{p\}$, for $m = 1..M_i$ and $i = 1..I$.

Note that the joint create operation is only required for the construction of the **validators**.

The links and filter functions necessary to implement the scheme are not shown here; the definitions are available in [1]. Instead we have concentrated on the the role of the creation operation. Two points warrant notice: First, a joint creation capability is (apparently) necessary to achieve the expressive power of monotonic HRU. Second, since the create structure is cyclic (entities of type p can indirectly create other type p entities), the safety of the scheme is outside the cases known to be decidable [15, 20]; this characteristic is consistent with the weak safety properties of HRU. It should be also noted that we have sketched theoretical equivalence, but we have not compared the ease of expressing explicit policies in the two models. This issue is beyond the scope of this paper. Finally, no example is given because the construction is not the most natural way to implement policies. Due to the general nature of the construction, even simple policies, such as Take/Grant, are transformed into lengthy schemes. For instance, a straightforward implementation of Take/Grant in SPM requires at most two links [15]. However, defining Take/Grant in HRU, and then applying the construction outlined above results in over twenty links.

5. DISCUSSION

In this section we outline some important questions regarding ESPM in particular and joint creation in general. We summarize our results and conjectures which are reported in further detail in [1].

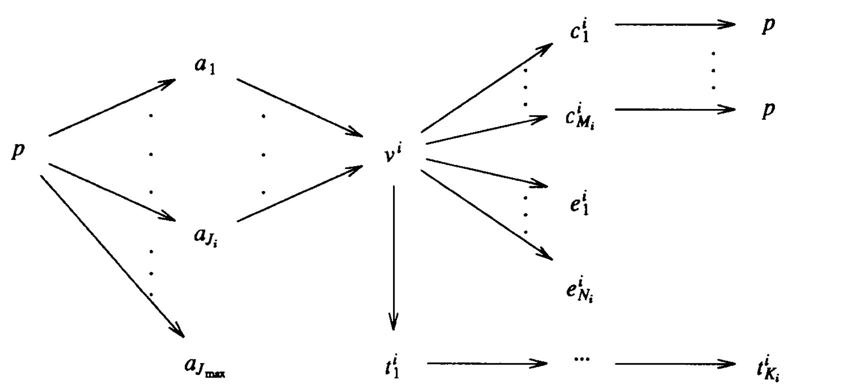

Fig. 1. Create And Joint Create Graph

5.1. Safety Properties of ESPM

Safety analysis for ESPM is similar to safety analysis for SPM. Most of the techniques in [15] can be applied directly, and, even where the machinery of [15] is inadequate, the underlying ideas remain valid. As in SPM, the basic result of our safety analysis for ESPM is that safety is decidable for attenuating acyclic schemes. Attenuating acyclic schemes for ESPM are schemes in which the create and joint create graph is acyclic, except for cycles of length one in which the child entity receives a subset of the tickets that each parent receives. Attenuating acyclic schemes appear to be adequate for expressing practical access control policies. A further result is that safety analysis is prohibitively expensive if the use of joint create operation is not carefully controlled. The cost of analyzing joint creation is exponential in the size of the initial state. Thus the expressive power of joint creation in the ESPM model carries a substantial cost in computational complexity.

5.2. Expressive Power of SPM vs. ESPM

In [16], it was conjectured that SPM has equivalent expressive power to monotonic HRU based on the fact that both allow schemes with undecidable safety. In this paper, we have implicitly argued that SPM is less expressive than monotonic HRU, and have shown that the joint create operation of ESPM is a sufficiently expressive mechanism to yield equivalence with monotonic HRU. We conjecture that SPM and ESPM have different expressive power for undecidable schemes. At present, we are unable to prove this conjecture. There is no difference in expressive power between SPM and ESPM for decidable schemes. For schemes in which safety is known to be decidable, any ESPM scheme can be simulated by an SPM scheme by the addition of a suitable number of types. In essence, the unique type names can be used instead of unique entity names if only a finite number of entities need be created for safety analysis. We note that for *practical* purposes, the ESPM joint create operation clearly cannot be simulated directly by an SPM scheme. At the very least, simulating ESPM's joint create in SPM requires an additional number of types that is exponential in the size of the initial state.

5.3. Binary vs. N-ary Joint Creates

As it turns out [1], any multiple joint creation can be implemented by a set of pairwise joint creates and the introduction of a fixed number of additional types, rights, links and filter functions. Thus, an ESPM model with N-ary creation is no more expressive than one with binary creation. However, the implementation of N-ary joint creation with binary joint creates has undesirable effects on safety analysis; when N-ary ESPM schemes with tractable safety analysis are converted to binary ESPM schemes, the constraints required by our safety analysis algorithm are no longer satisfied.

5.4. Other Mechanisms

Joint creation is not the only mechanism that can be added to SPM to achieve the expressive power of monotonic HRU. For example, a mechanism that allows a ticket to be copied only some fixed number of times before it expires can be used to implement the parameter selection mechanism of HRU. Our choice of joint creates in preference to this alternate mechanism is motivated by our strong desire to stay within a monotonic framework.

6. CONCLUSIONS

In this paper we have shown that a joint create operation is a useful abstraction from a pragmatic point of view, since it provides for convenient solution of a variety of protection problems taken from the existing literature. Our examples cover a range of classic problems such as mutual suspicion, protected subsystems, confinement and separation of duties. These problems are motivated by integrity considerations, although confinement does concern confidentiality.

We have also argued that from a theoretical point of view joint creation appears to confer additional expressive power not available with the traditional single-parent creation. In particular we have shown that extending the create operation of SPM [15] in this manner gives us a model (called ESPM) which is equivalent to the monotonic HRU model [7]. We conjecture that SPM is strictly weaker than ESPM. Finally, we have sketched arguments to demonstrate that ESPM retains the strong safety properties of SPM.

References

[1] Ammann, P.E. and Sandhu, R.S. "The Extended Schematic Protection Model", Technical Report, George Mason University, 1990.

[2] Bishop, M. and Snyder, L. "The Transfer of Information and Authority in a Protection System", *7th ACM Symposium on Operating Systems Principles*, 45-54 (1979).

[3] Clark, D.D. and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies", *IEEE Symposium on Security and Privacy*, 184-194 (1987).

[4] Cohen, E. and Jefferson, D. "Protection in the Hydra Operating System." *5th ACM Symposium on Operating Systems Principles*, 141-160 (1975).

[5] Graham, G.S. and Denning, P.J. "Protection - Principles and Practice." *AFIPS Spring Joint Computer Conference* 40:417-429 (1972).

[6] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D. "Protection in Operating Systems", *CACM*, 19(8):461-471 (1976).

[7] Harrison, M.H. and Ruzzo, W.L. "Monotonic Protection Systems", In *Foundations of Secure Computations*, DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J. (Editors), Academic Press (1978).

[8] Jones, A.K., Lipton, R.J. and Snyder, L., "A Linear Time Algorithm for Deciding Security", *17th IEEE Symposium on the Foundations of Computer Science*, 337-366 (1976).

[9] Lampson, B.W. "A Note on the Confinement Problem." *Communications of ACM*, 16(10):613-615 (1973).

[10] Linden, T.A. "Operating System Structures to Support Security and Reliable Software." *ACM Computing Surveys*, 8(4):409-445 (1976).

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

[12] Lipton, R.J. and Budd, T.A. "On Classes of Protection Systems", In *Foundations of Secure Computations*, DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J. (Editors), Academic Press (1978).

[13] Lockman, A. and Minsky, N. "Unidirectional Transport of Rights and Take-Grant Control", *IEEE Transactions on Software Engineering*, SE-8(6):597-604 (1982).

[14] Nash, M.J. and Poland, K.R. "Some Conundrums Concerning Separation of Duty." *IEEE Symposium on Security and Privacy*, 201-207 (1990).

[15] Sandhu, R.S. "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes", *JACM*, 35(2):404-432 (1988).

[16] Sandhu, R.S. "Expressive Power of the Schematic Protection Model", *Computer Security Foundations Workshop*, 188-193 (1988).

[17] Sandhu, R.S. "Transaction Control Expressions for Separation of Duties", *4th Aerospace Computer Security Applications Conference*, 282-286 (1988).

[18] Sandhu, R.S. "Transformation of Access Rights." *IEEE Symposium on Security and Privacy*, 259-268 (1989).

[19] Sandhu, R.S. "The Demand Operation in the Schematic Protection Model", *Information Processing Letters*, 32(4):213-219 (1989).

[20] Sandhu, R.S. "Undecidability of the Safety Problem for the Schematic Protection Model with Cyclic Creates", *JCSS*, to appear.