Proc. of 8th Annual Computer Security Applications Conf., San Antonio, TX, Dec 2-4, 1992, pages 131-140.

Implementing Transaction Control Expressions by Checking for Absence of Access Rights

Paul E. Ammann & Ravi S. Sandhu*

Center for Secure Information Systems and
Department of Information and Software Systems Engineering
George Mason University Fairfax, VA 22030

Abstract

Separation of duties is an important, real-world requirement that access control models should support. In [13], Sandhu introduced the transaction control expression (TCE) for specifying dynamic separation of duties. In this paper we consider the implementation of TCEs in the typed access matrix model (TAM) recently proposed by Sandhu [16]. We show that TAM requires extension for satisfactory handling of dynamic separation of duties. In particular, dynamic separation requires the capability to explicitly test for the absence of rights in cells of the access matrix. We illustrate how TAM, extended to incorporate such tests, can implement TCEs. We also discuss the impact of checks for absence of rights on safety analysis (i.e., the determination of whether or not a given subject can acquire a given right to a given object).

1 Introduction

The need for access controls arises in any computer system that provides for controlled sharing of information and other resources among multiple users. Access control models (also called protection models or security models) provide a formalism and framework for specifying, analyzing and implementing security policies in multi-user systems. These models are typically defined in terms of the well-known abstractions of subjects, objects and access rights with which we assume the reader is familiar.

Access controls are useful to the extent they meet user's needs. In this paper we consider the implementation of separation of duties. Separation of duties is an important real-world requirement that useful access control models need to support. The particular separation of duties mechanism we implement is the transaction control expression (TCE), introduced in [13]. It was shown in [11, 13, 15] that TCEs could easily specify typical transactions in which both separation and coincidence of duties were clear requirements. In particular, TCEs go beyond the static separation of duties stipulated by Clark and Wilson [5]. Static specification of separation of duties is too restrictive. For example, in modeling the common real world scenario in which a subject takes one role with respect to object A but another role with respect to object B, completely static specification of separation of duties is inadequate.

The principal contribution of this paper is that it (informally) demonstrates that specifying *dynamic* separation of duties requires expressive power beyond that found in classical access control models. Specifically, the ability to test for the absence of a right is needed to ensure that a given subject does not perform two conflicting operations on the same object.

In this paper we show how to implement TCEs by augmenting the access control model known as TAM, or typed access matrix, recently proposed by Sandhu [16]. TAM was selected because of its expressive power and conceptual simplicity. In order to accommodate TCEs, TAM needs to be augmented to allow for checking the absence of access rights. Most access control models proposed to date do not allow such checks. The original access matrix model of Lampson [9] took the position that access should be based on presence of access rights and not on their absence. This viewpoint was reiterated as a basic principle of protection by Saltzer and Schroeder [12]. Subsequently models such as take-grant [10], SPM [14], ESPM [1, 2, 3] and TAM [16] have followed this approach. As such, these models are incapable of ex-

*The work of both authors is partially supported by National Science Foundation grant CCR-9202270. Ravi Sandhu is also supported by the National Security Agency through contract MDA904-92-C-5141.

pressing the dynamic separation of duties embodied in TCEs. Although the Orange Book [6] calls for the ability to specify discretionary denial of access, TCEs require non-discretionary denial of access based on the past history of an object, and thus go beyond Orange Book requirements.

The paper's organization is as follows. Section 2 briefly summarizes TAM and the extensions to it required for testing for the absence of a right in a cell of the access matrix. We call the resulting model augmented TAM. Section 3 translates the TCE examples given in [13] into an augmented TAM implementation. Section 4 summarizes the important points for implementing general TCE expressions in augmented TAM by automated translation. Section 5 addresses the safety problem (i.e., the determination of whether or not a given subject can acquire a given right to a given object), and gives one approach to making safety analysis feasible for TCEs. Section 6 summarizes the paper.

2 The Typed Access Matrix Model

In this section we briefly review the typed access matrix model [16], and identify the extensions required to accommodate dynamic separation of duties. For readers familiar with HRU [7], we note that the essential difference between TAM and HRU is that TAM subjects and objects are strongly typed, whereas in HRU they are not. Strong typing means that each subject or object is created to be of a specific type, which thereafter cannot change.

2.1 Access Rights and Types

There is a finite set of access *rights* denoted by R. There is a finite set of *object types* (or simply *types*) denoted by T. There is a set of *subject types* T_S, $T_S \subseteq T$. The types and rights are defined when a system is initialized, and thereafter T and R remain constant. For example, $T = \{user, so, file\}$ specifies there are three types, viz., user, security-officer and file, with, say, $T_S = \{user, so\}$. A typical example of rights would be $R = \{r, w, e, o\}$ respectively denoting read, write, execute and own. We emphasize that the types and rights are specified as part of the system definition, and are not predefined in the model.

TAM represents the distribution of rights in the system by an access matrix. The matrix has a row and a column for each subject and a column for each object. The $[X, Y]$ cell contains rights which subject X possesses for subject or object Y.

The rights in the access matrix cells serve two purposes. Firstly, presence of a right, such as r, in $[X, Y]$ may authorize X to perform, say, the read operation on Y. Secondly, presence of a right, say o, in $[X, Y]$ may authorize X to perform some operation which changes the access matrix, e.g., by entering r in $[Z, Y]$. In other words, X as the owner of Y can change the matrix so that Z can read Y. The focus of TAM is on this second purpose of rights, i.e., the authorization by which the access matrix itself gets changed.

2.2 TAM Commands and Primitives

The protection state of the system is changed by means of commands defined as follows. A TAM *command* has the following format.

$$\begin{array}{l} \textbf{command } \alpha(X_1 : t_1, \ldots, X_k : t_k) \\ \quad \textbf{if } r_1 \in [X_{s_1}, X_{o_1}] \wedge \ldots \wedge r_m \in [X_{s_m}, X_{o_m}] \\ \quad \textbf{then } op_1; \ldots; op_n \\ \textbf{end} \end{array}$$

The name of the command is α, and X_1 through X_k are formal parameters. In general TAM commands have a condition part (the **if** part) and a body (the **then** part). The body consists of a sequence of *primitive operations*, $op_1; \ldots; op_n$.

There are six primitive operations in TAM. They come in three pairs of compensating operations, itemized below. The first operation in each pair adds something to the access matrix, whereas the second operation removes the same thing from the access matrix.

- The **enter** operation enters a right into an existing cell of the access matrix. If the right is already present in the relevant cell, **enter** has no effect. The **delete** operation deletes a right, if present, from an existing cell of the access matrix.

- The **create subject** operation introduces an empty row and column for the newly created subject into the access matrix. The **destroy subject** operation removes the row and column for the destroyed subject from the access matrix.

  The **create subject** operation requires that the subject being created does not previously exist. The **destroy subject** operation similarly requires that the subject being destroyed should exist.

- The **create object** and **destroy object** operations are much like their subject counterparts, except that they work on a column-only basis.

A TAM command is invoked by substituting actual parameters of the appropriate types for the formal parameters. The condition part of the command is evaluated with respect to its actual parameters. The body is executed only if the condition evaluates to true, and the pre-conditions for all **create** and **destroy** operations are satisfied. The commands are executed atomically, i.e., there is no interleaving of operations from different commands. Alternately, we can assume an interleaved model of execution with a serializability requirement. Note that if the pre-condition for any **create** or **destroy** operation in the body is false, the entire TAM command has no effect.

A TAM *authorization scheme* consists of a finite set of rights R, a finite set of types T, and a finite collection of commands. A TAM *system* is defined by giving an authorization scheme and the initial access matrix.

2.3 Augmented TAM

For the purposes of this paper, we define *augmented TAM* to be TAM with the addition that it is possible to test for the absence of a right in a cell of the access matrix. In other words, a test of the form $r_i \notin [X_{s_i}, X_{o_i}]$ may be present in the condition part of augmented TAM commands.

2.4 Command Invocation

The formal TAM model makes no statement about who initiates a command. A command invocation is simply taken to be a substitution of actual parameters for the formal parameters of the command definition. Although this is consistent with worst-case safety analysis, for many applications, including the one discussed in this paper, it is clearly important to identify who initiates each command for a given collection of actual parameters.

For purpose of this paper, we assume that a subset of the subjects are designated as the *principals*. This is achieved in augmented TAM by explicitly specifying the *principal types* as a subset of the subject types. Principals are the unit of accountability in the system. Every command must be initiated by one or more principals. In most cases there will be only one principal involved in a command. Occasionally we will use commands involving more than one principal. In such cases we require that all principals involved as actual parameters in the command agree to the command invocation.

3 Transaction Control Expressions

In this section we show how the examples of transaction control expressions given in [13] can be expressed in augmented TAM. A transaction control expression represents the potential history of an information object. Sandhu [13] distinguished two kinds of information objects: *transient objects* which can have a bounded number of operations applied to them, versus *persistent objects* which can potentially have an unbounded number of operations. Transient objects are intuitively modeled on forms, which are filled out and after appropriate approvals lead to some action such as issuing a check. Persistent objects intuitively model books in which account balances are maintained.

3.1 Transient Objects

The classic example of a transient object is a voucher that ultimately results in a check being issued. The potential history of a voucher is represented by the following transaction control expression [13].

prepare • clerk;
approve • supervisor;
issue • clerk;

Each *term* in this expression has two parts. The first part names a transaction. The transaction can be executed only by a user with *role* specified in the second part. For simplicity in discussion assume each user has only one role. So 'prepare • clerk' specifies that the prepare transaction can be executed on a voucher only by a clerk. The semi-colon signifies sequential application of the terms. That is a supervisor can execute the approve transaction on a voucher only after a clerk has executed the preceding prepare transaction. Finally, separation of duties is specified by requiring that the users who execute different transactions in the transaction control expression all be distinct.

We now show how the given TCE is specified in augmented TAM. We make use of the following sets of types and rights:

1. Rights R={prepare, prepare$'$, approve, approve$'$, issue, issue$'$}

2. Types T={voucher, clerk, supervisor, manager}, all of which are subject types, with principal types T_P={clerk, supervisor, manager}

Rights are used as a means of keeping track of the current location in the progression of a transaction control expression. *Undecorated rights*, i.e., those rights without a trailing apostrophe, are used to indicate that current operation in the transaction control expression is in progress. *Decorated rights*, i.e., those rights with a trailing apostrophe, are used to indicate that current operation in the transaction control expression is complete. The decorated rights are useful in ensuring both separation and coincidence of duties.

The augmented TAM commands for the voucher transaction control expression are given below. Each step of the TCE is translated into two commands: the first indicating that the step in question is in progress, and the second indicating that the step has been completed.

(a) **command** begin-prepare-voucher
    $(C : clerk, V : voucher)$
        create subject V
        enter prepare **into** $[C, V]$
    end

(a$'$) **command** complete-prepare-voucher
    $(C : clerk, V : voucher)$
    if prepare $\in [C, V]$ **then**
        delete prepare **from** $[C, V]$
        enter prepare$'$ **into** $[C, V]$
        enter prepare$'$ **into** $[V, V]$
    end

(b) **command** begin-approve-voucher
    $(S : supervisor, V : voucher)$
    if prepare$' \in [V, V]$ **then**
        delete prepare$'$ **from** $[V, V]$
        enter approve **into** $[S, V]$
    end

(b$'$) **command** complete-approve-voucher
    $(S : supervisor, V : voucher)$
    if approve $\in [S, V]$ **then**
        delete approve **from** $[S, V]$
        enter approve$'$ **into** $[S, V]$
        enter approve$'$ **into** $[V, V]$
    end

(c) **command** begin-issue-check
    $(C : clerk, V : voucher)$
    if approve$' \in [V, V] \wedge$ prepare$' \notin [C, V]$ **then**
        delete approve$'$ **from** $[V, V]$
        enter issue **into** $[C, V]$
    end

(c$'$) **command** complete-issue-check
    $(C : clerk, V : voucher)$
    if issue $\in [C, V]$ **then**
        delete issue **from** $[C, V]$
        enter issue$'$ **into** $[C, V]$
        enter issue$'$ **into** $[V, V]$
    end

To control progress of the TCE, the clerk in command (a) creates a voucher subject and acquires the undecorated right prepare, indicating that the first operation of the TCE is in progress. (As will be discussed later, command (a) can be modified to tie the voucher subject to one or more particular accounts with respect to which the voucher is being prepared.) Once the voucher has been prepared command (a$'$) is invoked to indicate, via the prepare$'$ right, that voucher preparation is complete. Command (a$'$) can be invoked only by the same clerk who invoked command (a) for a given voucher. Command (a$'$) enters the prepare$'$ right in the $[C, V]$ cell to record which clerk prepared the voucher. It also enters prepare$'$ in the $[V, V]$ cell to signify that the next step of the TCE can proceed. The commands (b) and (b$'$) allow a supervisor to obtain the approve right for the voucher provided preparation is complete; and subsequently denote, via the approve$'$ right, that voucher approval is granted. The command (c) give the named clerk the issue right for the voucher provided the voucher has been approved, and the specific clerk named in the command does *not* hold the prepare$'$ right for the voucher. This is where the facility to test for absence of rights is crucial. Command (c$'$) subsequently indicates, via the issue$'$ right, that the check has been issued. At this point the voucher's TCE is complete and the voucher can be archived. (The TAM command for archival has been omitted for simplicity.)

Several points about the example warrant attention. In particular, command (c) enforces dynamic separation of duties by checking for the absence of the prepare$'$ right before allowing a specific clerk to obtain the issue right. Testing for the absence of a right in a cell in the access matrix is outside the expressive power of standard nonmonotonic access matrix formulations such HRU and TAM. Also, commands "clean up" after themselves so as to ensure that only one thread is followed. For example, once a clerk has obtained the issue right, via command (c), no other clerk can obtain the issue right (because the approve$'$ right has been deleted from $[V, V]$). Thus it is assured that two clerks will not concurrently issue the check, with the undesirable consequence that two checks get issued for the same voucher.

Now suppose the check requires approval by three supervisors. We can specify this with the following TCE.

prepare • clerk;
approve • supervisor;
approve • supervisor;
approve • supervisor;
issue • clerk;

With this expression the three approve transactions must be executed sequentially. This is appropriate in a manual system where there is one physical representation of the check, which can be accessed by only one supervisor at a time. However, in a computerized system, it should be possible to request concurrent approval. Sandhu [13] proposed the following notation for expressing multiple approval.

prepare • clerk;
3 : approve • supervisor;
issue • clerk;

The colon is a voting constraint specifying 3 votes from 3 different supervisors in this case, without requiring the voting to be sequential.

We can implement this example in augmented TAM by modifying the commands (b) and (b′) from the previous example as follows; the other commands remain as they are.

(b) **command** begin-approve-voucher
$(S1, S2, S3 : supervisor, V : voucher)$
if prepare′ $\in [V, V]$ **then**
delete prepare′ **from** $[V, V]$
enter approve **into** $[S1, V]$
enter approve **into** $[S2, V]$
enter approve **into** $[S3, V]$
end

(b′) **command** complete-approve-voucher
$(S1, S2, S3 : supervisor, V : voucher)$
if approve $\in [S1, V] \wedge$ approve $\in [S2, V]$
$\wedge$approve $\in [S3, V]$ **then**
delete approve **from** $[S1, V]$
delete approve **from** $[S2, V]$
delete approve **from** $[S3, V]$
enter approve′ **into** $[S1, V]$
enter approve′ **into** $[S2, V]$
enter approve′ **into** $[S3, V]$
enter approve′ **into** $[V, V]$
end

It is necessary to the success of the command (b′) that a given actual parameter to a TAM command be represented by at most one formal parameter. In other words we assume here that $S1$, $S2$ and $S3$ must be distinct supervisors. This assumption differs from the usual convention followed in access control models (see, for example, [3, 7, 14, 16]).

An objection that might be raised to the preceding implementation is that it requires simultaneous agreement from three supervisors to approve a voucher. It can be argued that asynchronous agreement better models organizational requirements. Fortunately, asynchronous agreement can be achieved with the introduction of additional rights. We sketch an asynchronous solution here.

The idea is to introduce one right that stands for a single approval, another that stands for two approvals, and so on, up to a right that stands for n approvals, where $n = 3$ in this example. One then defines a series of commands, one for each possible count of approvals. In the example given, these commands need only have a single supervisor and the voucher as arguments. Each begin-approve-voucher command replaces the undecorated right for n approvals in the $[V, V]$ cell with the undecorated right for $n - 1$ approvals, and enters a single undecorated approval right in an $[S, V]$ cell. Each complete-approve-voucher command replaces the decorated right for $n - 1$ approvals in the $[V, V]$ cell with the decorated right for n approvals, and replaces the undecorated approval right in the $[S, V]$ cell with a decorated approval right. By including appropriate checks for the absence of rights, it can also be ensured that a given supervisor does not grant more than one approval.

Following [13], further consider the requirement that either three supervisors approve the check or the department manager plus one supervisor approve it. The TCE notation allows weights for different roles as follows.

prepare • clerk;
3 : approve • manager=2, supervisor=1;
issue • clerk;

Approve transactions with sufficient votes are required before proceeding to the next term. In this case approve transactions executed by managers have weight 2 whereas those executed by supervisors have weight 1. If two managers approve the check we get 4 votes. It seems reasonable to allow this so we interpret the number of votes required as a lower bound. The moment 3 or more votes are obtained the next step is enabled.

Essentially the implementation must allow for progress to be made by the disjunction of various possible steps. A natural way to implement this is with a corresponding variety of augmented TAM commands for a given step, each of which is capable of enabling the following step.

For this example, a possible TAM implementation is as follows. The previous commands (b) and (b′) are still acceptable and necessary, in that they represent a possible way in which approval might be achieved. They are not sufficient, however, since the implementation must account for other ways in which votes may be collected. There needs to be a means by which a manager can combine with a supervisor, and also a means by which two managers can generate an approval. The result is two pairs of additional commands, of which one pair is shown below. (A similar pair would be required for two managers approving a voucher.)

(d) **command** begin-approve-voucher-2
  $(S : \textit{supervisor}, M : \textit{manager}, V : \textit{voucher})$
  if prepare′ $\in [V, V]$ **then**
    delete prepare′ **from** $[V, V]$
    enter approve **into** $[S, V]$
    enter approve **into** $[M, V]$
  end

(d′) **command** complete-approve-voucher-2
  $(S : \textit{supervisor}, M : \textit{manager}, V : \textit{voucher})$
  if approve $\in [S, V] \wedge$ approve $\in [M, V]$ **then**
    delete approve **from** $[S, V]$
    delete approve **from** $[M, V]$
    enter approve′ **into** $[S, V]$
    enter approve′ **into** $[M, V]$
    enter approve′ **into** $[V, V]$
  end

The concise voting notation in the TCE has been fully enumerated in the translation to augmented TAM commands. Provided the translation is automated, this expansion is not problematical for many typical cases. Also, by introducing additional rights as discussed earlier, we can decouple the supervisor and manager approvals to make them asynchronous.

3.2 Coincidence of Duties

Sometimes different transactions in an object history must be executed by the same user. Consider a purchase order with the following transaction control expression.

requisition • project-leader;
prepare • clerk;
approve • manager;
agree • project-leader;
issue • clerk;

The idea is that a project leader initiates a requisition, a purchase order is prepared from the requisition, approved by a purchasing manager, and then needs agreement of the project leader before finally being issued by a clerk. Our rule of distinct identity implies different project leaders be involved in requisitioning and agreeing, contrary to the desired policy. The following TCE syntax identifies which steps must be executed by the same user.

requisition • project-leader ⇃ x;
prepare • clerk;
approve • manager;
agree • project-leader ⇃ x;
issue • clerk;

The anchor symbol '⇃' identifies steps which must be executed by the same individual. The x following it is merely a token for relating multiple anchors, as for example in the TCE given below.

requisition • project-leader ⇃ x;
prepare • clerk;
approve • manager ⇃ y;
agree • project-leader ⇃ x;
reapprove • manager ⇃ y;
issue • clerk;

In this case there are two steps to be executed by the same project leader, and two to be executed by the same purchasing manager.

The prepare, approve and issue steps in this TCE are similar to those in the voucher example of section 3.1, and can be accomplished by similar augmented TAM commands. In addition we need commands to initiate the requisition, agree to the purchase order, and to reapprove the purchase order. To implement this TCE we define the following rights and types.

1. Rights R={requisition, requisition′, prepare, prepare′, approve, approve′, agree, agree′, reapprove′, reapprove′, issue, issue′}

2. Types T={purchase-order, project-leader, clerk, manager}, all of which are subject types, with principal types T_P={project-leader, clerk, manager}

The full set of augmented TAM commands is given below.

(a) **command** begin-initiate-requisition
  $(P : \textit{project-leader}, O : \textit{purchase-order})$
    create subject O
    enter requisition **into** $[P, O]$
  end

(a′) **command** complete-initiate-requisition
    (P : *project-leader*, O : *purchase-order*)
    if requisition $\in [P, O]$ **then**
        delete requisition **from** $[P, O]$
        enter requisition′ **into** $[P, O]$
        enter requisition′ **into** $[O, O]$
end

(b) **command** begin-prepare-po
    (C : *clerk*, O : *purchase-order*)
    if requisition′ $\in [O, O]$ **then**
        delete requisition′ **from** $[O, O]$
        enter prepare **into** $[C, O]$
end

(b′) **command** complete-prepare-po
    (C : *clerk*, O : *purchase-order*)
    if prepare $\in [C, O]$ **then**
        delete prepare **from** $[C, O]$
        enter prepare′ **into** $[C, O]$
        enter prepare′ **into** $[O, O]$
end

(c) **command** begin-approve-po
    (M : *manager*, O : *purchase-order*)
    if prepare′ $\in [O, O]$ **then**
        delete prepare′ **from** $[O, O]$
        enter approve **into** $[M, O]$
end

(c′) **command** complete-approve-po
    (M : *manager*, O : *purchase-order*)
    if approve $\in [M, O]$ **then**
        delete approve **from** $[M, O]$
        enter approve′ **into** $[M, O]$
        enter approve′ **into** $[O, O]$
end

(d) **command** begin-agree-to-po
    (P : *project-leader*, O : *purchase-order*)
    if approve′ $\in [O, O] \wedge$ requisition′ $\in [P, O]$
then
        delete approve′ **from** $[O, O]$
        enter agree **into** $[P, O]$
end

(d′) **command** complete-agree-to-po
    (P : *project-leader*, O : *purchase-order*)
    if agree $\in [P, O]$ **then**
        delete agree **from** $[P, O]$
        enter agree′ **into** $[P, O]$
        enter agree′ **into** $[O, O]$
end

(e) **command** begin-reapprove-po
    (M : *manager*, O : *purchase-order*)
    if agree′ $\in [O, O] \wedge$ approve′ $\in [M, O]$ **then**
        delete agree′ **from** $[O, O]$
        enter reapprove **into** $[P, O]$
end

(e′) **command** complete-reapprove-po
    (M : *manager*, O : *purchase-order*)
    if reapprove $\in [M, O]$ **then**
        delete reapprove **from** $[M, O]$
        enter reapprove′ **into** $[M, O]$
        enter reapprove′ **into** $[O, O]$
end

(f) **command** begin-issue-po
    (C : *clerk*, O : *purchase-order*)
    if reapprove′ $\in [O, O] \wedge$ prepare′ $\notin [C, O]$
then
        delete reapprove′ **from** $[O, O]$
        enter issue **into** $[C, O]$
end

(f′) **command** complete-issue-po
    (C : *clerk*, O : *purchase-order*)
    if issue $\in [C, O]$ **then**
        delete issue **from** $[C, O]$
        enter issue′ **into** $[C, O]$
        enter issue′ **into** $[O, O]$
end

In commands (d) and (e), a check is made to ensure coincidence of duties. Thus the same project-leader who makes the requisition agrees to the subsequent form of the requisition. Also, the same supervisor who approves the requisition does the reapproval after the project-leader has indicated agreement. In command (f), on the other hand, a check is made for absence of an access right, thus ensuring separation of duties.

3.3 Persistent Objects

We now turn our attention to persistent objects. We propose the following transaction control expression for representing the potential history of an account.

create $\bullet$ supervisor;
{debit $\bullet$ clerk + credit $\bullet$ clerk};
close $\bullet$ supervisor;

The curly parenthesis denote repetition while '+' gives a choice on each repetition. The idea is that an account is created, thereafter repeatedly debited or credited, and at some point closed. Any object whose transaction control expression contains indefinite repetition is, by definition, a persistent object. Similarly any object whose transaction control expression does not contain repetition is, by definition, transient.

The history of a persistent object may be lengthy. It is impractical to convert the transaction control expression incrementally into an history, as done for transient objects. We can realistically have only some abbreviated history for persistent objects available to the access control system. Fortunately, it is improper to require that all transactions executed on a persistent object be performed by distinct users. An account may have hundreds of debit and credit operations, while the organization employs only a few dozen clerks. Separation of duties carried to this extreme will paralyze the organization. The fundamental principle is that transactions are executed on persistent objects only as the side effect of executing them on transient objects [13]. Separation of duties can be enforced by keeping the following history information.

1. The entire history of transient objects.

2. A partial fixed length history of persistent objects for non-repetitive portions of the transaction control expression.

For the account example, assume that Dick is the supervisor who creates the account, as a side effect of executing a transaction on some transient object. The TCE of the account is modified to record this fact as follows.

$$\begin{array}{l}\text{create} \bullet \text{Dick};\\ \{\text{debit} \bullet \text{clerk} + \text{credit} \bullet \text{clerk}\};\\ \text{close} \bullet \text{supervisor};\end{array}$$

Thereafter, as debit and credit transactions are executed on the account, again as a side effect, the expression remains unmodified. Finally when the account is closed by some supervisor other than Dick, say Jerry, this fact is recorded in the TCE to give us the following.

$$\begin{array}{l}\text{create} \bullet \text{Dick};\\ \{\text{debit} \bullet \text{clerk} + \text{credit} \bullet \text{clerk}\};\\ \text{close} \bullet \text{Jerry};\end{array}$$

There is a separation of duty involved in creating and closing the account. But separation of duty in debiting and crediting it is enforced only to the extent specified in the transaction control expressions on the transient objects related to this account.

There is no great difficulty in implementing the transient object/permanent object TCE distinction in augmented TAM. The general rule is that there must be some TAM object created for each transaction on which separation of duties needs to be enforced. In the above example, the create right for an account is given to Dick, and the absence of the create right in the cell [Jerry, account] allows Jerry to obtain the close right for that same account.

For the repetitive debit or credit operations, a separate voucher subject is created each time, and transactions as illustrated in earlier examples can manipulate the column of the access matrix associated with the voucher leading up to a debit or credit on the account when the check is issued. To relate the voucher subject to the account in question, the account can be tied to the voucher subject at the time the voucher is created. The stipulation in [13], that Dick cannot approve vouchers for accounts that he has created, can be easily accommodated. The detailed augmented TAM commands required for this example are omitted due to lack of space.

4 Automatic Translation of TCEs

In this section, we provide some general observations on the implementation of TCEs in augmented TAM by automated translation. It is clear that any translation scheme, based on the examples of section 3, must accommodate at least the following.

- A subject, such as a voucher, must be created to serve as the communication channel by which the TCE proceeds. The communication subject effectively stores a "program counter" for the TCE and controls which operation can occur next. At creation time, communication subjects for transient objects can be tied to related subjects, such as accounts and responsible users, for persistent objects. When a transaction is complete, the communication subject can be destroyed, typically after audit information has been archived.

- As part of the conditional test in the TAM commands for successive operations in a transaction control expression, satisfactory completion of prior steps must be checked. Such checking is done by consulting the rights stored for the communication subject for the TCE.

- Separation of duties is enforced by explicitly checking for the absence of a particular right or set of rights. The specific checks are easily determined by examining the prior operations in the transaction control expression.

- Conversely, coincidence of duties, i.e., when the same principal must perform two or more tasks, is enforced by explicitly checking for the presence of a particular right or set of rights. Again, the specific checks are easily determined by examining the prior operations in the transaction control expression.

- Voting is achieved by either multiple TAM commands or by disjunction in the conditional test of a TAM command. In general, a concise voting expression in a transaction control expression may result in a combinatorial number of resulting TAM commands.

- The invocation of a TAM command with a given set of arguments must imply real world agreement by the users represented by those arguments. For instance, when a TAM command enters approve into the $[S, V]$ cell, there must be assurance that the action represents the supervisor's instructions, and not some malicious party. Briefly, authentication issues require attention in an actual implementation.

5 Safety Analysis of TCEs

In this section, we explain the general problems of safety analysis, and the complications that checking for the absence of a right in a matrix cell can introduce into safety analysis.

The *protection state* of a system is defined by the privileges, which we equate here with rights, possessed by the individual subjects. Once the initial state of a system has been established, the state evolves by the autonomous activity of subjects. A security model, such as HRU or TAM, provides a framework for specifying the dynamics of the protection state. Such a collection of rules is called an *authorization scheme*, or simply a *scheme*.

To understand the implications of a scheme, it must be possible to determine the cumulative effect of authorized incremental changes in the protection state. The incremental state changes authorized by a scheme may appear innocent enough in isolation, but their cumulative effect turns out to be undesirable. For a given initial state and authorization scheme, we need to characterize protection states that are reachable.

This problem was first identified in [7] where it is called the *safety problem*. In its most basic form, the safety question for access control asks: is there a reachable state in which a particular subject possesses a particular privilege for a specific object? It is the fundamental question which an access control model must confront. Since in most access control models subjects are usually authorized to create new subjects and objects, the system is unbounded; and it is not certain that such analysis will be decidable, let alone tractable, without sacrificing generality.

There is an essential conflict between the expressive power of an access control model and tractability of safety analysis. The access matrix model as formalized by Harrison, Ruzzo, and Ullman (HRU) [7] has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties. Safety is undecidable for most policies of practical interest, even in the monotonic version of HRU [8].

The main contribution of access control models such as take-grant [10], SPM [14], ESPM [1, 2, 3] and TAM [16] is that they offer tractable safety for schemes of practical interest. It is easy to add expressive power to such models, and in many cases, the need for additional expressive power can be readily argued. For example, in this paper we argue for the need to check for the absence of access rights. The difficulty is ensuring that the increased expressive power does not destroy the ability to analyze safety.

It is known that the general inclusion of checks for the absence of rights greatly complicates the safety analysis. For example, in [4] it is shown how to embed satisfiability in an access matrix model in which it is possible to test for the absence of rights. Thus safety if general checks on the absence of rights are allowed is NP-hard. What is required is some tolerable restriction that avoids exponentially difficult analysis.

In the context of this paper, we propose to address the safety problem by restricting the usage of checking for the absence of rights. In particular, if access is specified with transaction control expressions, then we allow checking for the absence of access rights in the (mechanical) translation to augmented TAM. Otherwise, we prohibit checking for the absence of rights. We also require that the rights used in implementing TCEs be disjoint from TAM rights used for other purposes.

The advantage of this approach is that safety analysis for TCEs can be carried out separately from safety analysis for the rest of the TAM authorization scheme, a process which is explained in [16]. Safety analysis for TCEs by themselves is relatively straight forward. From inspection, it is clear which types of users can receive particular TCE-related rights. For instance, in the first example presented earlier, it is clear the clerks can obtain the undecorated and decorated versions of the prepare and issue rights for a given voucher, and that the same holds for supervisors with respect to the approve rights.

6 Conclusion

In this paper we have analyzed the implementation for transaction control expressions in the augmented typed access matrix model. Transaction control expressions are important because they provide a natural mechanism for the specification of separation of duties applications. The main result of this paper is that, to implement transaction control expressions in the access matrix model, it appears necessary to allow checks for the absence of access rights in cells of the access matrix. Such checks are outside the expressive power of nonmonotonic HRU and (unaugmented) TAM. Examples of translations of transaction control expressions into the augmented access matrix (augmented TAM) were given, and general considerations in the translation were outlined. The complications of the increased expressive power of augmented TAM on safety analysis were discussed, along with a way of handling such complications.

Acknowledgments

The authors are grateful to Nathaniel Macon, Howard Stainer, and Mike Ware for making this work possible.

References

[1] Ammann, P.E. and Sandhu, R.S. "Extending the Creation Operation in the Schematic Protection Model." *Proc. Sixth Annual Computer Security Applications Conference*, 340-348 (1990).

[2] Ammann, P.E. and Sandhu, R.S. "Safety Analysis for the Extended Schematic Protection Model." *Proc. IEEE Symposium on Research in Security and Privacy*, 87-97 (1991).

[3] Ammann, P.E. and Sandhu, R.S. "The Extended Schematic Protection Model." *Journal of Computer Security*, to appear.

[4] Budd, T.A. "Safety in Grammatical Protection Systems." *International Journal of Computer and Information Sciences* 12(6):413-431 (1983).

[5] 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).

[6] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria.* DoD 5200.28-STD, (1985).

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

[8] Harrison, M.H. and Ruzzo, W.L. "Monotonic Protection Systems." In DeMillo et al (Editors). *Foundations of Secure Computations.* Academic Press (1978).

[9] Lampson, B.W. "Protection." *5th Princeton Symposium on Information Science and Systems*, 437-443 (1971). Reprinted in *ACM Operating Systems Review* 8(1):18-24 (1974).

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

[11] Nash, M.N. and Poland, K.R. "Some Conundrums Concerning Separation of Duty." *IEEE Symposium on Security and Privacy*, 201-207 (1982).

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

[13] Sandhu, R.S. "Transaction Control Expressions For Separation Of Duties." *Proc. Fourth Annual Computer Security Applications Conference*, 282-286 (1988).

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

[15] Sandhu, R.S. "Separation Of Duties In Computerized Information Systems." *Proc. Database Security IV: Status and Prospects* S. Jajodia and C.E. Landwehr, eds., Elsevier, 179-189 (1991).

[16] Sandhu, R.S. "The Typed Access Matrix Model." *Proc. IEEE Symposium on Research in Security and Privacy*, 122-136 (1992).