*Proc. 4th Aerospace Computer Security Applications Conference, Orlando, Florida, December 1988 , pages 282-286.*

# TRANSACTION CONTROL EXPRESSIONS FOR SEPARATION OF DUTIES

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

**Abstract**

We describe a model and notation for specifying and enforcing aspects of integrity policies, particularly *separation of duties*. The key idea is to associate a *transaction control expression* with each information object. This expression constrains the transactions which can be applied to that object to occur in the specified pattern. As operations are actually executed the transaction control expression gets converted to a history. This history serves to enforce separation of duties. We distinguish *transient objects* with a short lifetime from *persistent objects* which are long lived. Separation of duties is achieved by maintaining a complete history for transient objects but only a partial history for persistent objects. This is possible because of the system enforced rule that transactions are executed on persistent objects only as a side effect of execution on transient objects.

## 1 INTRODUCTION

The issues raised by Clark and Wilson [2] have stimulated recent interest in integrity policies. Their objective was to establish that: "First, there is a distinct set of security policies, related to integrity rather than disclosure ... Second, some separate mechanisms are required for enforcement of these policies, disjoint from those of the Orange Book [5]." Others have expressed similar opinions [1, 4, 11, 14]. The conclusions appear almost self evident. Unfortunately the debate gets clouded and emotional when couched in terms of differences between commercial and military policies. So the following comment is worth noting [3]: "It is clear that the distinction between military and commercial practices should not be made a central issue ... both the military and the commercial world have clear and obvious need for both assurance of integrity and control of disclosure." We are in complete agreement with both quotations.

Our objective is to present some preliminary, but promising, results in developing a notation and model for the specific issue of separation of duties. Separation of duties is a fundamental technique for prevention of fraud and errors, known and practised long before the existence of computers. Although it is applied in computerized information systems (see [15, 17] for instance), perhaps even routinely so, there is little literature on it.

Our model generalizes access control lists in ways similar to Minsky's generalization of capabilities [13]. Our approach has many of the same advantages that access control lists have over capabilities for file protection. We also acknowledge the influence of [10, 12]. The model goes significantly beyond this previous work and enables specification of separation of duties in an intuitive and realistic way. It has a simple operational interpretation with an efficient implementation. The model is based on the standard concept of transaction [6, 8] which is reviewed in section 2 and shown to be insufficient for integrity control. Section 3 introduces transaction control expressions and illustrates their power for enforcing separation of duties. We also indicate why efficient implementation is feasible. Section 4 discusses correspondence of our model with Clark-Wilson [2].

## 2 TRANSACTIONS

The notion of a transaction as the atomic unit of activity in an information system has provided a successful base for concurrency control and recovery mechanisms. The following are two critical properties.

1. *Serializability.* The net effect of the interleaved execution of multiple transactions must be equivalent to some serial execution of the transactions.

2. *Failure Atomicity.* Either all or none of the actions of a transaction actually take effect.

Transactions also provide a useful building block for integrity; since it is assumed, perhaps even certified, that each transaction by itself preserves consistency of the database. These properties are critical for hiding temporary inconsistencies.

However, a transaction is too elementary a unit for integrity purposes. For example, consider a situation in which payment in the form of a check is prepared and issued by the following sequence of events.

1. A clerk prepares the check.
2. The check is approved by a supervisor.
3. The check is issued by a clerk, who must be different from the clerk in step 1.

From a concurrency control and recovery perspective this sequence is best viewed as three separate transactions, one for each step. The activities they represent are separated in time by unpredictable and possibly large intervals. For instance, the first two steps may be performed on line while the third is executed in batch. Moreover different users have responsibility and authorization for these activities. On the other hand, from an integrity perspective we must view this sequence as a single unit. The third step, in particular, makes explicit the constraint that the clerk executing it be different from the clerk in the first step. So it will take collusion of two clerks and a supervisor to perpetrate fraud. Since the check is presumably issued against some account the above sequence is more properly expressed as follows.

1. A clerk prepares the check and assigns an account.
2. The check and account are approved by a supervisor.
3. The check is issued by a clerk who must be different from the clerk in step 1. Issuing the check has the side effect of debiting the account assigned in step 1.

That is the check contains a reference to an account, which is another information object in the system. Strictly speaking for double entry accounting the reference should be to two accounts, one to be debited and the other credited in step 3. The important point for us is that transactions executed on the check have side effects on objects such as accounts.

A check and an account are two rather different kinds of objects. The check is a *transient object* which comes into existence, has a finite sequence of operations applied to it and then disappears (possibly leaving a record in some archive). The account on the other hand is a *persistent object* with a long life in the system with a potentially unbounded sequence of credit and debit operations performed on it. Of course, at some point the account may be closed. The key point is that we cannot prescribe its history as a finite sequence of actions. Both kinds of objects are essential to the logic and correct operation of an information system. Transient objects embody a logically complete history of transactions corresponding to a unit of service provided to the external world. Persistent objects embody the internal records required to keep the organization functioning with accurate correspondence to its interactions with the external world.

*Our fundamental thesis is that integrity can be achieved by enforcing controls on transient objects, for the most part. The crucial idea, which makes this possible, is that transactions should be executed on persistent objects only as a side effect of executing transactions on transient objects.*

# 3 TRANSACTION CONTROL EXPRESSIONS

We propose to represent the potential history of an information object by a transaction control expression. First consider transient objects. The history of the check in our example is described as follows.

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 check object only by a clerk. The semi-colon signifies sequential application of the terms. That is a supervisor can execute the approve transaction on a check only after a clerk has executed the preceding prepare transaction. Finally, separation of duties is further enforced by requiring that the users who execute different transactions in the transaction control expression all be distinct. As individual transactions are executed the expression gets incrementally converted to a history, for instance as follows.

| (a) | (b) | (c) |
|---|---|---|
| prepare $\bullet$ Tom;<br>approve $\bullet$ supervisor;<br>issue $\bullet$ clerk; | prepare $\bullet$ Tom;<br>approve $\bullet$ Dick;<br>issue $\bullet$ clerk; | prepare $\bullet$ Tom;<br>approve $\bullet$ Dick;<br>issue $\bullet$ Harry; |

The identity of the user who executes each transaction is recorded to enforce the requirement that these users be distinct. So if Tom attempts to issue that check at point (b) in this sequence the system rejects the attempt.

A transaction control expression thus contains a history of transactions executed on the object in the past and a potential history which authorizes transactions which can be executed in future. The expression begins as a constraint and ends as a complete history of the object. Transient objects will generally have a history of the order of a dozen steps at most, so this approach is viable. Moreover the information filled out as the history gets executed is an essential part of the object which should anyway be represented as part of the object state. In a manual system a transient object is represented by a form. Different transactions executed on the object are recorded by appropriate entries on the form. Identification of the user executing each transaction is achieved by signatures. In automated systems user identities must be recorded with guaranteed correctness.

Now suppose the check requires approval by three supervisors. We might specify this as follows.

prepare $\bullet$ clerk;
approve $\bullet$ supervisor;
approve $\bullet$ supervisor;
approve $\bullet$ supervisor;
issue $\bullet$ 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. We propose the following notation for expressing multiple approval.

prepare $\bullet$ clerk;
3 : approve $\bullet$ supervisor;
issue $\bullet$ clerk;

The colon is a voting constraint specifying 3 votes from 3 different supervisors in this case, without requiring the voting to be sequential. Further consider the requirement that either three supervisors approve the check or the department manager plus one supervisor approve it. The notation is easily extended to include weights for different roles as follows.

prepare $\bullet$ clerk;
3 : approve $\bullet$ manager=2, supervisor=1;
issue $\bullet$ 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. Non-weighted terms are special cases of weighted terms so we can equivalently write the above as follows.

1 : prepare $\bullet$ clerk=1;
3 : approve $\bullet$ manager=2, supervisor=1;
1 : issue $\bullet$ clerk=1;

It is tempting to introduce additional notation. For example, the votes required for approval might depend on the value of the check, say 3 votes for less than \$1000 and 5 for more. Although some degree of value dependent voting may turn out to be necessary, for the moment we forgo the temptation to include it. Instead we would prefer to identify two different kinds of checks, say small checks and big checks, and have the system enforce the integrity constraint that small checks must have value less than \$1000. We could then have two different transaction control expressions for these two types of checks, as follows.

prepare $\bullet$ clerk;
3 : approve $\bullet$ manager=2, supervisor=1;
issue $\bullet$ clerk;

(a) Small-Checks

prepare $\bullet$ clerk;
5 : approve $\bullet$ manager=2, supervisor=1;
issue $\bullet$ clerk;

(b) Big-Checks

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 • purchasing-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. We propose the following syntax to identify steps must be executed by the same user.

requisition • project-leader ⇂ x;
prepare • clerk;
approve • purchasing-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. For instance in

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

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

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

create • supervisor;
{debit • clerk + credit • clerk};
close • supervisor;

The curly parenthesis denote repetition while '+' gives a choice on each repetition. The idea is that a account gets created and is thereafter debited or credited. At some point it may be closed. Any object whose transaction control expression contains 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 is likely to be lengthy. It is clearly 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. After all 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. Our fundamental principle is that transactions are executed on persistent objects only as the side effect of executing them on transient objects. Then 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 transaction control expression of the account is modified to record this fact as follows.

create • Dick;
{debit • clerk + credit • clerk};
close • supervisor;

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 to give us the following.

create • Dick;
{debit • clerk + credit • clerk};
close • Jerry;

So 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 a subtlety concerning the actions of a supervisor in approving a check, which as a side effect debits an account, and the supervisor's action in creating a account. That is Dick might approve checks issued against an account which he himself created. There are two approaches we might take. One alternative is that when Dick attempts to approve a check for an account that he created, the system forbids it. Note the transaction control expression of an account contains the information necessary for this purpose, i,e., whether or not Dick created the account. This does complicate the enforcement mechanism somewhat. The second alternative is to recognize the potential for conflict by noting that a supervisor can create an account as well as act on transient objects referring to that account. Creation and closing of accounts can then be delegated to a separate role.

create • account-manager;
{debit • clerk + credit • clerk};
close • account-manager;

In this case we can certify there is no possibility of conflict, so approval of a check by a supervisor need not involve consideration of the transaction control expression for the related account. We can either insist we always create new roles in this manner to avoid conflicts or otherwise use this information for optimization in our enforcement mechanism. At any rate there is a clear need for formalizing these issues and developing analysis tools for detecting such cases.

# 4 CLARK-WILSON MODEL

We now consider the relationship between our model and the Clark and Wilson model [2]. Clark and Wilson identify constrained data items and transformation procedures as basic components of their model. We have the following correspondence.

| Our Model | Clark and Wilson Model |
|---|---|
| Information Object | Constrained Data Item (CDI) |
| Transaction | Transformation Procedure (TP) |

We assume the system enforces the rule that information objects can be modified only by authorized transactions. We also assume that execution of a transaction takes a valid state into another valid state. Clark and Wilson have similar requirements.

Authorization in our model is different in significant ways from the Clark and Wilson model. Clark and Wilson formulate the following rule.

> E2: The system must maintain a list of relations of the form: (User ID, $\mathrm{TP_i}$, ($\mathrm{CDI_a}$, $\mathrm{CDI_b}$, $\mathrm{CDI_c}$, ...)) which relates a user, a TP, and the data objects that TP may reference on behalf of that user. It must ensure that only executions described in one of the relations are performed.

Clark and Wilson seem to imply these relations be explicitly maintained. In our formulation these relations exist in an implicit form. Authorization is explicitly stated in terms of user roles and transaction control expressions. Our viewpoint is more natural in that the power of a user is typically derived from his position in an organization rather than being a function of his individual self. If a user's position changes so does his authority. Since such changes are inevitable, perhaps even periodically mandated for separation of duties, it is appropriate to explicitly relate authority and responsibility to positions rather than to the individuals occupying them at a given moment.

We are assuming each user has a unique identifier and no user is permitted to have more than one identity. This assumption is necessary since the system can only enforce separation in terms of distinct user identifiers. As pointed out by Clark and Wilson [3] this itself is something that can be enforced by separation of duties regarding who is authorized to register users.

In our presentation we do not consider all the rules of the Clark and Wilson model. For instance the rule that each TP writes to an append only audit log is not explicitly stated. As a transaction control expression is converted to a history we are maintaining a mini-audit log on a per object basis. We would need additional rules to require a global and complete log. More importantly we have ignored the problem of who gets to set or change the transaction control expression of an object. Our general attitude here is that the transaction control expression of an object is derived from the type of that object. However, in a complete model we would need to make such issues explicit. We have also ignored how one selects a particular supervisor to approve a check.

We present our model as an approach in trying to fulfill the goals laid out by Clark and Wilson. Lot more work needs to be done on different aspects of the integrity problem. One thing that makes the integrity problem very different from the disclosure problem as dealt with in the Orange Book [5] is that it has a lot more dynamism in that new roles get created, existing roles are reorganized, new users and transactions get created, etc. Dealing with such dynamics is known to be difficult [9]. Recent results provide the basis for a good solution [16]. We also need to consider availability issues. These can arise in subtle ways. For example, suppose all the clerks go on strike. Does it become impossible to issue checks? Or do we allow supervisors to act as clerks in an emergency mode of operation? These issues need to be formalized in a clean, coherent and usable manner.

## 5 CONCLUSION

We have presented a notation and model based on transaction control expressions for specifying and enforcing separation of duties. Our exposition has been informal, reflecting the current state of the model. We do believe the model is intuitively sound and permits efficient implementation. A key idea is to distinguish transient objects with a fixed length history from persistent ones with potentially unbounded histories. Transactions are executed on persistent objects only as the side effect of executing transactions on transient objects. This allows us to enforce separation of duties by maintaining a complete history for each transient object and an abbreviated history for persistent objects, consisting of the non-repetitive part.

Transaction control expressions constrain the pattern in which transactions can be executed on an object. Separation of duties is enforced by the rule that for transient objects different transactions must be executed by distinct users. As an exception we do allow that certain transactions in a sequence be executed by the same user. By expressing transaction control expressions in terms of user roles, we specify separation of duties in a compact and natural way without reference to individual users.

Our model addresses only a small part of the overall integrity problem. We believe that progress will be made only by attacking the overall problem in small pieces. The WIPCIS report [18] at some places expresses the sentiment that we should seek a single model to cover non-disclosure, integrity and availability. While this is a commendable objective in the long term, for the short term we recommend the opposite view that even in isolation the integrity issue is sufficiently complex to require different models dealing with different aspects. Unification of these models into a coherent whole is best postponed till we have understood different aspects of the overall problem.

Finally consider the following claim where security is meant as non-disclosure [7]: "In general, security policies are very simple, and should be easy to state in an appropriate formalism." We tend to agree but offer the following counterpoint: "In general, integrity policies are quite complicated and specific to individual organizations. However, they should be "easy" to state and analyze in an appropriate formalism." The challenge is to develop such a formalism. Our viewpoint is that this will require us to first understand pieces of the problem, coming up with different formalisms for different pieces before we can hope to arrive at a single unified formalism for all integrity concerns. Much work remains to be done.

## References


[1] Chalmers, L.S. "An Analysis of the Differences between the Computer Security Practices of the Military and Private Sectors." *IEEE Symposium on Security and Privacy*, 71-74 (1986).

[2] 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).

[3] Clark, D.D. and Wilson, D.R. "Comments on the Integrity Model." In [18].

[4] Courtney, R.J. "An Industry View of the DoD Computer Security Center Program." *6th Seminar on the DoD Computer Security Initiative*, 11-13 (1983).

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

[6] Eswaran, K.P., Gray, J.N., Lorie, R.A. and Traiger, I.L. "The Notions of Consistency and Predicate Locks in a Relational Database System." *Communications of the ACM* 8(11):624-633 (1976).

[7] Gougen, J.A. and Meseguer, J. "Security Policies and Security Models." *IEEE Symposium on Security and Privacy*, 11-20 (1982).

[8] Gray, J.N. "Notes on Database Operating Systems." In *Operating Systems: An Advanced Course*, edited by Bayer, R., Graham, R.M. and Seegmuller, G., Lecture Notes in Computer Science, Vol. 60, Springer Verlag (1978).

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

[10] Karger, P.A. "Implementing Commercial Data Integrity with Secure Capabilities." *IEEE Symposium on Security and Privacy*, 130-139 (1988).

[11] Katzke, S. (Moderator). "Panel Session — Base Spectrum of Computer Security Requirements." *6th Seminar on the DoD Computer Security Initiative*, 14-21 (1983).

[12] Kieburtz, R.B. and Silberschatz, A. "Access-Right Expressions." *ACM Transactions on Programming Languages and Systems* 5(1):78-96 (1983).

[13] Minsky, N. "An Operation-Control Scheme for Authorization in Computer Systems." *International Journal of Computer and Information Sciences* 7(2):157-191 (1978).

[14] Moffett, J.D. and Sloman, M.S. "The Source of Authority for Commercial Access Control." *Computer* 21(2):59-69 (1988).

[15] Murray, W. H. "Data Integrity in a Business Data Processing System." In [18].

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

[17] Wimbrow, J.H. "A Large-Scale Interactive Administrative System." *IBM Systems Journal* 10(4):260-282 (1971).

[18] Preliminary report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), Bentley College, MA, October 1987.