****** SUBMITTED FOR PUBLICATION ******

Technical Report CS81005-R

Modeling of MULTISAFE Protection Enforcement Processes with Extended Petri Nets*

H. Rex Hartson

and

Earl J. Balliet**

March 1981

Department of Computer Science

Virginia Polytechnic Institute and State University

Blacksburg, VA 24061

---

* The work reported herein was supported, in part, by the National Science Foundation under Grant Number MCS-7903936.

** Present address: Bell Telephone Laboratories, 6 Corporate Place, Room 1D-217, Piscataway, NJ 08854.

Modeling of MULTISAFE Protection Enforcement Processes with Extended Petri Nets*

H. Rex Hartson
and
Earl J. Balliet**

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

ABSTRACT

The various kinds of access decision dependency within a predicate-based model of database protection are classified according to cost of enforcement. Petri nets and some useful extensions are described. Extended Petri nets are used to model the flow of messages and data during protection enforcement within MULTISAFE, a multimodule system architecture for secure database management. The model demonstrates that stated criteria for security are met within MULTISAFE. Of particular interest is the modeling of data dependent access conditions with predicates at Petri net transitions.

Keywords and phrases: Database security, Petri nets, predicate-based protection, modeling, data dependent checking, MULTISAFE, enforcement

---

* The work reported herein was supported, in part, by the National Science Foundation under Grant Number MSC-7903936.
** Present address: Bell Telephone Laboratories, 6 Corporate Place, Room 1D-217, Piscataway, NJ 08854.

1. INTRODUCTION

A continuing problem in computing systems is the need for controls on the access to stored data. Database access controls have commonly been ad hoc, inflexible, and sometimes circumventable, pointing out the need for a more structured approach. In particular, modular system architectures and security mechanism modeling are indicated. The Database Security Research Project at VPI & SU has developed a modular system architecture called MULTISAFE, discussed in [TRUER81]. The chief conceptual feature of MULTISAFE is the logical (and, possibly, physical) isolation of all protection and enforcement mechanisms from the users and from the operations on the database. This was accomplished by separating the database system into three distinct modules. Criteria for security have been presented in terms of the flow of inter-module messages [TRUER81].

Modeling of MULTISAFE has been done from several points of view, including cost and performance (based on access condition classifications), implementation with a relational database approach to storing and retrieving authorization information, and distributed protection in a distributed database environment. This paper describes a Petri net model of MULTISAFE which is descriptive, which helps verify security conditions, and which provides insight into implementation.

Petri nets have been shown to be a useful tool in the study of concurrent activities and the modeling of the flow of control throughout a system. However, Petri nets do not possess enough structure to efficiently model message flow in MULTISAFE. Several extensions to the

theory of Petri nets exist in various forms in the literature. These extensions include the association of identifiers with tokens, so that tokens can be distinguished, the association of tokens with messages which have data content, the queueing of tokens in a place, and the resolution of conflicts in a net by the association of a predicate with a transition. In this paper these extensions are gathered and enhanced in a novel application of Petri nets to database protection. In particular, it is the first time that Petri net transition predicates have been used to model data dependent access conditions. The concept of queues of predicates at transitions is introduced to facilitate the evaluation of those access conditions.

A conflict-free Petri net model of MULTISAFE message flow is presented. First, this model is useful as a descriptive model of MULTISAFE data security, especially as it provides insight into possible implementations of a predicate-based model of database protection. The model is also shown to preserve the criteria for security in MULTISAFE. Binding time, access condition evaluation, and enforcement policy are analyzed in the process of demonstrating that the conditions for security are maintained. The use of a counting function at a transition is introduced to verify the completedness of a Petri net.

The predicate-based model of authorization and enforcement of [HARTH76] provided the motivation for studying types of access condition dependencies. In this paper, access conditions are categorized into three distinct types: system, query, and data dependent. The data dependent access conditions are further refined into several categories of increasing complexity and increasing performance demands.

2. CLASSES OF ACCESS DECISION DEPENDENCY

A predicate-based protection model has been described in [HARTH76]. The generality introduced there by the use of predicates as conditions of access implies a variety of ways in which the access decision can be dependent on different kinds of system information. Three broad categories of access conditions which can be placed on a user of a database are:

1. System dependent
2. Query dependent
3. Data dependent

2.1. System Dependent Access Conditions

An access condition (predicate) is a system dependent condition if its Boolean value can be ascertained from information available about the general system state. Such a condition might require that the time of day be between 8 a.m. and 5 p.m., allowing database operations only during regular working hours, or only on certain days of the week or month.

2.2. Query Dependent Access Conditions

A condition is a query dependent condition if its Boolean value can be ascertained from the query itself. A SEQUEL-type [CHAMD76] language is used to illustrate. A simple SEQUEL request takes the form:

```
SELECT <attribute-list>
FROM   <relation-name>
WHERE  <selection-predicate>
```

A query dependent access condition can limit the relations upon which the user can operate, the attributes from which the user can SELECT, and the attributes which can be used in the WHERE clause.

The following database of three relations is used to illustrate the various kinds of dependency. (Much of this discussion of access decision dependency is taken from [BALLE81, HARTH80].)

EMP (EMP#, NAME, SALARY, BIRTH_YEAR, DEPT, YRS_SERVICE)
containing information about employees with attributes of employee number, name, annual salary rate, year of birth, department, and years of service

TAX (EMP#, NBR_DEPS, EARNED)
containing tax information about employees, with attributes of employee number, number of dependents, and year-to-date earnings

MGR (DEPT, MGR_NAME, STATUS)
with the manager names for each department and information on "clearance status"

Managers, being employees as well, are also listed in the EMP relation. EMP and TAX are separate relations for operational reasons and not, for example, because of third normal form requirements. The clearance status information in the MGR relation is provided by each manager for his/her department; the semantics are unimportant here.

As an example of a query dependent condition, consider an authorization which allows the selection of NAMEs from the EMP relation as long as the WHERE predicate does not specify SALARY values. Similarly, a selection of SALARY from the EMP relation may be disallowed if NAME also appears in the SELECT clause attribute list.

2.3. Data Dependent Access Conditions

A condition is a data dependent condition if its value cannot be ascertained without a retrieval (or perhaps several retrievals) from the database. The classes of data dependency are presented in approximate order of increasing complexity.

2.3.1. Class A Data Dependency

A condition has a class A data dependency if it has truth values in a one-to-one relationship with the tuples retrieved in response to a query, and the values depend on the retrieved fields of the retrieved tuples. With such a dependency, an access decision must be made for each tuple retrieved. Consider the following simple, unqualified query, Q:

```
SELECT NAME, SALARY
FROM   EMP
```

For Q, an example of a class A access condition is:

SALARY < 20,000

2.3.2. Class B Data Dependency

A condition has a class B data dependency if the condition has values in a one-to-one relationship with the tuples retrieved in response to a query, but the information needed to evaluate the condition is not present in the retrieved fields. Instead, it is in another field (or other fields) of the retrieved tuples. For Q, this access condition (which must be computed from BIRTH_YEAR) is class B:

age < 40

2.3.3. Class C Data Dependency

A condition has a class C data dependency if the condition has values in a one-to-one relationship to the elements retrieved in response to a query, but the information needed to evaluate the condition is not present in any fields of the relation. Instead, the information is available from another relation (or relations) in the database.

The following access condition is of class C with respect to query Q:

NBR_DEPS <= 3

2.3.4. Class D Data Dependency

A condition has a class D data dependency if the condition has values in a one-to-n relationship with the tuples retrieved in response to a query, and the information needed to evaluate the condition requires only one database retrieval. This information may be:

1. in the response to the query,
2. in the non-response attributes of the response tuples, or
3. found by making a single additional request to the database, independent of the user's query.

For simplicity, in all cases, an extra access from the database is conservatively assumed to be needed.

Given query Q, this access condition is of class D:

name of manager of finance department ≠ 'Joe'

An access to the MGR relation is needed. If the name (MGR_NAME) of the finance department manager is Joe, access is immediately denied to all tuples selected in response to Q. If the name of the finance department manager is not Joe, then access is allowed to all of the response tuples.

2.3.5. Class E Data Dependency

A condition has a class E data dependency if the condition involves aggregate information (sum, count, average, or derived data). Five subclasses of E are discussed in [BALLE81], but are not detailed here. The subclasses are distinguished by the kinds of data used in the aggregates. The subclasses of dependencies in the aggregate information,

approximately parallel the classes of the non-aggregate dependencies described above. Some examples of aggregate- and derived-data-dependent access conditions are:

average age of employees retrieved < 30

average salary of all employees in finance department < 25000

age of employee when hired < 21

2.3.6. Summary of Data Dependency Classes

For easy reference, a summary table of the data dependency classes is presented. In the table below, 1-1 means one decision per tuple, whereas, 1-n means one decision per query. Retrieved tuples mean tuples of the original relation retrieved in response to the query, before being projected down to the list of attributes named in the SELECT clause.

Summary of Data Dependency Classes

CLASS	RELATIONSHIP WITH RETRIEVED TUPLES	ACCESS CONDITION IS DEPENDENT ON:
A	1-1	retrieved fields of retrieved tuples
B	1-1	non-retrieved fields of retrieved tuples
C	1-1	not in retrieved data
D	1-n	data anywhere, often independent of query
E	1-n (usually)	aggregate
Ea	1-n	retrieved fields of retrieved tuples
Eb	1-n	non-retrieved fields of retrieved tuples
Ec	1-n	data anywhere, often independent of query
Ed	1-1	derived from tuples retrieved
Ee	1-1	derived from other data

2.4. Dependency and Binding

The time at which each part of an access condition can be shown to be satisfied for a given query is called the binding time for that part. The time at which a complete access decision can be reached depends upon the type of dependencies within the access condition.

2.4.1. Binding of System Dependent Access Conditions

If a condition has only a system dependency, it can be evaluated as soon as the system state can be determined. A system dependent condition can be applied at any time from the logging in of the user (possibly even earlier), to the time the results of a service request are returned to the user. As an example, a policy may limit database access to times between 8 a.m. and 5 p.m. For most purposes that can be interpreted to mean that database operations may be allowed only if the request for service occurs in the prescribed time range. Or, if timing is critical, it may mean that the response be returned to the user only within the same time interval. These two interpretations imply two different binding times, one at the start of the service and one at the end.

2.4.2. Binding of Query Dependent Access Conditions

If a condition has only a query dependency, then enforcement can be performed upon receipt of a query, leading to a relatively early binding time. If the query is:

```
SELECT   NAME, SALARY
FROM     EMP
WHERE    DEPARTMENT = 'finance'
```

and the access policy for this user and the EMP relation is that NAME is allowed as long as SALARY is not also requested, then the query can be immediately rejected.

A query dependent access condition may involve combinations of the SELECT attributes, relations used, and attributes in the WHERE clause. All forms of query dependent access conditions can be bound at query processing time.

2.4.3. Binding of Data Dependent Access Conditions

If a condition has a data dependency, the binding time depends upon the class of the dependency. Class A or B conditions (being one-to-one with retrieved tuples) must be bound repeatedly as the responses to a query return from the database. A class C condition requires one additional database retrieval to be made for each response to a query. Thus, class C conditions require a later (and more costly) binding time than that of classes A and B. Upon completion of this retrieval, a decision can be made which affects only one response. A class D condition requires that one additional database retrieval be made for the entire query. Upon completion of this retrieval, a decision can be made which is in effect for all responses to the query. Therefore, although the class D binding time is very late, it is not as costly (because it is not as frequently needed) as that of class C conditions. A class E condition requires that aggregate information be computed before the condition can be evaluated.

3. ENFORCEMENT AND DISCLOSURE POLICIES

Policy considerations at a high level are fundamental to the behavior of the system. Two enforcement resolution policies and two disclosure policies are considered. An enforcement resolution policy, introduced in [HARTH77], is a policy which determines how much data is to be returned to the requester when only part of the response to his/her query is accessible according to the authorization information. A disclosure policy, introduced here, is a policy which decides how informed the user should be of the authorizations that pertain to his/her database operations. Enforcement policy applies only to how the enforcement process is performed, whereas disclosure policy is used only in formulating user messages which explain the response or lack of response.

3.1. Definitions

If any part of any response to a query fails to meet the authorization requirements for access, a full enforcement policy requires that no information at all be passed to the user. Full enforcement is an 'all-or-nothing' policy. In contrast, partial enforcement is a policy which permits the user to be passed all the individual responses to the query that pass all applicable access conditions. Only the responses that fail an access condition are withheld.

Associated with full and partial enforcement are two disclosure policies, complete and null disclosure. Complete disclosure is a policy under which a user is made aware of all the authorizations that control his/her database operations. Null disclosure is a policy whereby the user is kept completely unaware of the authorizations placed on his/her database activities.

In a database system none of the four possible policy combinations may be universally applicable. A different policy may be chosen depending upon the individual user and on the information that is being protected. Determining which of the four policies best suits a particular user over a given part of the database is generally left to the discretion of the authorizer.

If, under a full disclosure policy, the user is told s/he may get the NAME and ADDRESS of all employees, s/he, nevertheless, is still not told about the existence of other information, possibly in the same relation. The user merely knows that there may or may not be other attributes of the relation that s/he cannot query. Full disclosure can also lead to illegal inference of data values. For example, if the user asks for the SALARY of the employee named Jones and the access condition is to allow SALARY if SALARY < $20,000, the user upon being informed of the access condition that is violated will know that Jones earns more than $20,000. Further discussion of the inference problem is beyond the scope of this paper, but is treated extensively in the literature (as examples, see [DENND79, DOBKD79]).

3.2. Partial Enforcement

The INGRES database system is an interesting example of a partial enforcement policy, around which the entire protection system is built. The protection mechanism of INGRES [STONM74] is used to modify incoming queries so that the resulting query is guaranteed to request only authorized data. The modified request is then processed by the database retrieval programs without further need for enforcement of access rules. To greatly oversimplify (details are found in [STONM74]), data dependent access conditions are ANDed to the search qualification predicate (in the WHERE clause) of the incoming query. For example, if a given employee of the ABC Company has access only to records of Department A, his/her general request for information about the entire company is modified with a predicate such as: DEPT = 'A'. Thus, there is a response to the request, but it contains no information from tuples having a Department value other than A.

It is possible for this policy to cause difficulties if the requester does not understand that the response corresponds to a query other than the one s/he submitted. It is particularly serious if the requester asked for, say, the average salary in Company ABC. If the user was unaware that the result was an average for only Department A, s/he would unknowingly be dealing with false information. Of course, a full disclosure policy would provide the information necessary to know that the results are inaccurate and why they are inaccurate. On the other hand, a null disclosure policy may be suitable if a user does not need to be aware of some parts of the database. For example, if a user, whose view

of the world includes only a certain department, asks for the average salary, s/he gets only the average salary of the employees that s/he has the authorization to access, and need not be concerned as to the restrictions placed on his/her request by an authorizer. This approach is exemplified by the early ASAP security measures, based on user views [CONWR72].

3.3. Full Enforcement

In contrast, consider a full enforcement access policy which allows all employee salaries except the salary of a manager. If the user asks for the average salary of all employees, s/he is told that no response is allowed. Only a correct (from a global viewpoint) query, requesting the average salary of all employees except the manager, is allowed, and it produces a correct response.

4. PETRI NETS

The following section reviews Petri nets only briefly to provide some background, for the unfamiliar reader, to other sections that follow. The reader interested further in the theory and application of Petri nets should begin with Peterson's survey [PETEJ77] and pursue some of the references therein.

4.1. Introduction to Petri Nets

A Petri Net [AZEMP78] is a four tuple (P,T,I,O) where: P is a set of places, drawn as circles; T is a set of transitions, drawn as bars; I is an input function and O is an output function. Places and transitions are connected variously with arcs.

$$I: P \times T \longrightarrow \{ 0, 1 \}$$

$$O: P \times T \longrightarrow \{ 0, 1 \}$$

If pi $\in$ P and tk $\in$ T, then:

$$I(pi,tk) = \begin{cases} 1, & \text{if there is an arc from place pi to transition tk} \\ 0, & \text{otherwise} \end{cases}$$

and

$$O(pi,tk) = \begin{cases} 1, & \text{if there is an arc from transition tk to place pi} \\ 0, & \text{otherwise} \end{cases}$$

In the standard graphical representation of Petri nets, the direction of the arcs that connect the places and transitions is generally not indicated, it being the assumption that all flow in a net is in a downward direction. Any ambiguity is, of course, resolved by the input and output functions. However, in this paper, arrows are drawn on the arcs, whenever an ambiguity is possible.

A Petri net is populated by the presence of tokens in the places. A token is represented as a dot. A place can contain zero or more tokens. Given a transition t, a place p for which $I(p,t) = 1$ is called an input place for t. Similarily, a place p for which $O(p,t) = 1$ is called an output place for t.

A transition t is said to be firable if each place p, for which $I(p,t) = 1$, has at least one token. Once a transition is firable, it can fire, removing one token from each of its input places and adding one token to each of its output places. By this movement of tokens, the Petri net exhibits dynamic behavior. Although a firable transition transition can wait a non-zero time before firing, the firing itself is considered to occur in zero time. Thus the state of the net, which is represented at any instant of time by the number of tokens in each place, is always well defined. Further, it follows that the probability of more than one transition firing at any given instance of time is zero.

Throughout the paper, the following notation is adopted:

$\forall$	'for all' (the universal quantifier)	&	'logical AND'
$\neq$	'not equal to'	|	'logical OR'
$\in$	'is a member of' (set membership)	~	'logical NOT' (negation)
$\oplus$	'exclusive-or'		

The state of a net is indicated by its marking. A marking of a net is a function, M,

$$M : P \longrightarrow Z$$

where P is the set of places and Z is the set of non-negative integers, such that $\forall p \in P$, M(p)= the number of tokens in place p. A marking M shall be denoted by a vector with dimension equal to the cardinality of the set P. The coordinate values of the vector are the values of M(p) for each place. The net changes states by the firing of transitions, which produces new markings.

Given a net and an initial marking of the net, M0, a Marking M is derivable from M0 if there exists a sequence of transitions t1, t2, ..., tk which when successively fired, result in the marking M. The forward marking class of a net given M0 is the set of all markings derivable from M0. The forward marking class of a bounded net with initial marking M0 is finite.

The forward marking class of a net can be graphically represented using a tree diagram. A reachability tree of a net, given an initial marking M0, is a tree, the root which is an initial marking M0 and each other node of which is a marking derivable from its ancestor by the firing of one or more transitions. The arcs connecting the nodes of the tree are labeled with the transitions that have been fired.

A transition t which has no input places (i.e., $I(p,t)=0$, $\forall p \in P$) is called a source. A source acts as a communication port to the external environment through which the net can receive information. A transition which has no output places (i.e., $O(p,t)=0$, $\forall p \in P$) is called a sink. A sink allows information to pass as output to the external environment.

Petri nets have been successfully used to model concurrency in a system, as several paths in a net may be active concurrently [AZEMP78, PETEJ77]. This concurrency is illustrated in the reachability tree by labeling the arc, that connects two nodes, with the set of transitions that have been fired concurrently. Here, 'concurrent' does not mean 'exactly simultaneously,' but that the transitions fire essentially in parallel; their order is immaterial.

A firable transition tj is disarmed by a transition ti if the firing of ti removes a token from an input place shared with tj, with the result that tj is no longer firable. If two or more transitions are firable in a given marking, and the firing of one disarms the others, then the transitions are said to be in conflict. Transitions t1 and t2

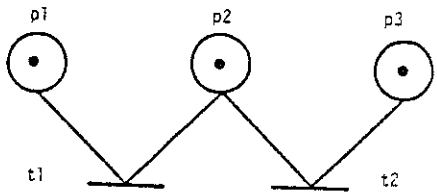

Figure 1. A Conflict in a Petri Net

of the net in Figure 1 are in conflict. Since it cannot be ascertained which transition will fire, the net is nondeterministic. Firing orders [PETEJ77] have been introduced in an attempt to eliminate this nondeterminism. However, this approach greatly complicates the model.

4.2. Extended Petri Nets

'Pure' Petri net theory is not adequate to model the concepts discussed in this paper. This lack of modeling power has been noted by other investigators [PETEJ77, AZEMP78, ELLIC77, NUTTG72]. It is necessary to completely resolve conflicts, attach meaning to a token as a message which may contain data, order the tokens in a place (in a queue), develop a mechanism to determine when all entered tokens have arrived at a sink of the net, and be able to fire a transition when no token is at any of its input places. Most of these extensions are scattered in the literature in various forms. This paper attempts to gather, enhance, and unify them.

4.2.1. Predicates for Conflict Resolution

In [AZEMP78], conflicts are resolved by the association of a predicate, p, with each transition, t. The predicate is a function of a vector of program variables, x. Its association with a transition, t, is denoted by pt(x). (The x in this notation is omitted unless it is necessary for clarity.) For a transition to be firable now, each input place must have at least one token, as before. However, the transition will not fire unless, in addition, pt = true.

Consider a predicate associated with each of transitions t1 and t2 in Figure 1. No conflict exists, if either pt1 or pt2 has a value of false. Given a marking, the existence of a conflict now depends upon the values of the predicates associated with the transitions.

The following is proposed as a condition to ensure that a net is conflict-free.

Proposition 1: No conflict exists in a Petri net provided that

$\forall$ pi $\in$ P, if Ti = {tj| I(pi,tj) = 1}

and if ptk = true for some tk $\in$ Ti,

then ptm = false, $\forall$ m $\neq$ k.

(Note: For all k, ptk is the negation of the logical OR of the ptm, where m $\neq$ k). That is, given a place pi, the predicates associated with those transitions for which pi is an input place are such that at most one predicate can be true at any given time.

4.2.2. Information Content of a Token

Tokens, so far being nothing more than marks in places, are indistinguishable one from another. In order to associate the movement of specific tokens with events in the modeled system, it is useful to attach an identifier to each token. This first step in assigning data content to a token leads directly to the generalization in which an entire message is associated with each token. Data is transported via these messages. This leads to conflict resolution based on the information that is passed as the transition fires. The predicate associated with a transition can be a condition whose value depends on a wide variety of elements. In particular, a predicate can depend on the system state, as well as data content of tokens that are in input places to the transition. This also leads to the ability for output transitions to

channel tokens into different places based on the information content of the tokens.

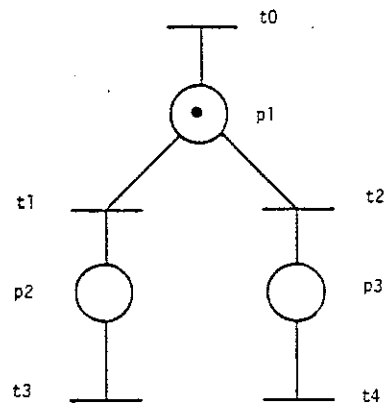

Figure 2. Data Dependent Predicates to Resolve a Conflict

For example, if a token represents a NAME and a SALARY value, in the context of a database relation about employees, then in the net of Figure 2 with pt1 = "SALARY < 20,000" and pt2 = "SALARY $\geq$ 20,000," the given net separates all tokens into two categories: the NAMEs and SALARYs of all people who earn less than $20,000 and those that earn equal to or greater than $20,000. Clearly in the above example, pt1 $\oplus$ pt2 is true, where $\oplus$ designates the 'exclusive-or' between pt1 and pt2. Thus, t1 and t2 are never in conflict, according to Proposition 1.

A place, now, contains tokens which have information content. As tokens build up in a place, transitions fire to remove the tokens one at a time. When tokens were indistinguishable, there could be no ordering to the tokens in a place, and it could not be determined which token had been removed by the firing of a transition. In [ELLIC77, NUTTG72],

evaluation nets, which allow incoming message queues as well as places in a net model, were introduced as useful extensions of Petri nets. Here, now that tokens are distinguishable, each place shall be considered to be a queue, which preserves the sequencing of the tokens in that place.

4.2.3. The t_count Function for Completedness

Consider a net that is conservative (the number of tokens is invariant as the transitions fire) and which does not trap tokens (in endless cycles through the same transitions). In such a net the sum of the number of tokens appearing at sinks will eventually equal the sum of the number of tokens entered at sources. When those sums are equal, the net is said to be completed. The determination of when all inputs have been processed through such a net is now a matter of counting the tokens at the sinks. Clearly, completedness of a net holds specifically for some fixed time period, with a given start time and end point.

Consider the following queue, Q1, of five NAMEs and salaries.

Joe	10 K
Harry	15 K
Mary	20 K
Rita	8 K
Jane	9 K

Let the source transition t0, in Figure 2, have the queue Q1, given above, as as its input. After transition t0 fires (five times), place p1 has all five of the tokens. If transition t1 has the predicate pt1 = "SALARY < 10 K" and transition t2 has the predicate pt2 = "SALARY $\geq$ 10 K," then transition t1 delivers into place p2, the queue Q2:

Rita	8 K
Jane	9 K

while transition t2 produces in place p3, the queue Q3:

Joe	10 K
Harry	15 K
Mary	20 K

Now that queues have been implanted in each place and the tokens have been given order, it is desirable to be able to tell when all tokens in a queue have been processed by the net or subnet that is currently under consideration.

In order to determine when all the tokens in a queue have been processed, a t_count function is introduced.

$$t\_count : T \longrightarrow Z$$

where T is the set of transitions and Z is the set of non-negative integers. The function t_count associates with each transition ti, an integer (initially zero) which is incremented by one each time that the transition is fired.

Assuming that the original length of Q1 is known to be five, the net in Figure 2 will have the entire queue put into place p1 as soon as t_count(t0) = 5. Now, either transition t2 or t3 fires, depending upon the value of the conditions pt2 and pt3 associated with t2 and t3. The queue, Q1, has completely been partitioned into Q2 and Q3 when

$$t\_count(t1) + t\_count(t2) = 5$$

Thus, it can be determined by a numerical check of the t_count values, that the entire queue, Q1, has been processed and that no new tokens

have been created and that none of the tokens from Q1 have been lost within the net (i.e., that the net is conservative) and completed. This is important in verifying the security of message flow.

4.2.4. An Inhibitor Arc

A transition can fire only if each input place has at least one token. In [PETEJ77] the concept of an inhibitor arc which allows for zero-testing is discussed. An inhibitor arc is an arc from a place p to a transition t which allows the transition to fire only if the place has zero tokens. The inhibitor arc is denoted by a small circle at the end of the arc which connects the place to the transition. For a more complete description of the power added to Petri net theory by the introduction of this concept, see [PETEJ77].

4.3. A Module of a Petri Net

The concept of a subnet of a Petri net, which allows a net to be analyzed at several levels of abstraction, is presented in [AZEMP78]. This subnet is called a module.

The behavior of a Petri net is analyzed by the evolution of the successive markings. The input and output conditions are often the main points of interest of a particular module. The reduction abstraction consists of merging the internal transitions of a module into a new set of transitions (generally this set has only one element) which directly

connect input places to output places. Reduction preserves the input/output behavior of the module while suppressing its internal details.

The concept of modules can be used to greatly simplify a complex Petri net by compacting a module into one transition. It can also be used to expand a transition into a Petri net whenever the current view of the net requires, respectively, less or greater detail. Thus different portions of a net may be viewed at several different levels of abstraction at one time.

In [AZEMP78], Petri nets have been used to model communication protocols. A modular approach was used to produce a bottom-up analysis of the global behavior of a module with respect to its environment, once its modeling and verification have been performed at a lower level. In the next section, using a similar approach, the major functional parts of MULTISAFE are abstracted into Petri net modules, hiding the internal details of MULTISAFE implementation and highlighting the intermodule communication structure.

5. PETRI NET MODEL OF MULTISAFE

5.1. Introduction to MULTISAFE*

A MULTImodule system for supporting Secure Authorization with Full Enforcement (MULTISAFE) for shared database management is being developed [TRUER79, TRUER81] by Robert Trueblood at the University of South Carolina and Rex Hartson at Virginia Polytechnic Institute and State University. Performance improvements are expected to be achieved by a combination of multiprocessing, pipelining, and parallelism. The MULTISAFE protection processor can meet complex policy requirements with flexible, generalized protection mechanisms.

The system configuration is based on functional division into three major modules:

1. the user and application module (UAM)
2. the data storage and retrieval module (SRM)
3. the protection and security module (PSM)

Each module is implemented on one or more processors forming the multiprocessor system. In MULTISAFE all three modules function concurrently. The UAM coordinates and analyzes user requests at the same time that the SRM generates responses for requests. Simultaneously, the PSM continuously performs security checks on all activities. Figure 3 illustrates the logical relationships among the three modules. All processing within MULTISAFE is initiated and controlled by events occurring within

---

* This section is adapted from [HARTH81].

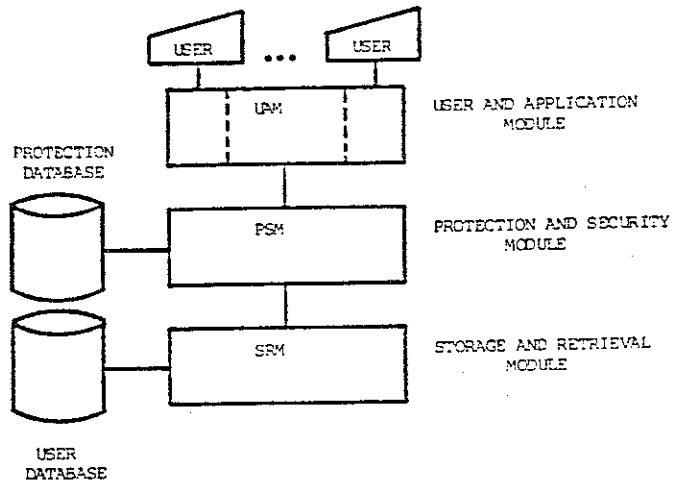

Figure 3. Logical Relationship Among the Three Modules

the message flow, including such events as the transmission of data to and from the database.

Typically, the concepts of isolation and separation have been considered important for supporting data security. However, the protection question can still be considered to be open, because physical isolation is not a guarantee of security. It is at the logical level that evidence must be given that an architecture does indeed support data security. Unless communication among the system components can be shown to

be logically secure (in terms of both message control and message content), security is not necessarily gained by isolation. Consequently, special attention is given to intermodule communication in [TRUER81] and in this paper.

In MULTISAFE, messages are sent between modules via an encapsulated data type. Its contents are set and checked by protected procedures which are invoked parametrically. No user or user process can directly access these message objects. The primary mechanisms are structured and verifiable. For a single user, it is shown in [TRUER81] that the only message path between the UAM and the SRM is established by a sequence of carefully defined operations on abstract objects in the PSM. It is also shown that the message sequences from multiple users, introduced for the sake of concurrency, can be effectively serialized, leaving intact the security of the single user case.

Messages from either authorizers or users are subject to two kinds of security checking: 1) checking specific to the request, and 2) system occupancy checking. System occupancy checks relate to overall permission to be an active user of the system, without regard to how the system is being used. The system occupancy check is always made in conjunction with login. For example, the conditions (separate from user identification) for a given system user may be that occupancy is allowed only between 8:00 a.m. and 5:00 p.m. System occupancy checking at data request time and other times provides (optional)

5.2. The Petri Net Model of MULTISAFE

In the Petri net model of MULTISAFE the places correspond to the three modules and the two databases. The transitions correspond to the lines of communication between modules and databases. Tokens correspond to messages flowing from module to module.

Using Petri net modules to represent the major parts of MULTISAFE, the interpreted Petri net model, in Figure 4, has the form (P,T,I,O) where:

P = {UAM, PSM, SRM, DB, PDB}

T = {t0, t1, t2, t3, t4, t5, t6, t7, t8}

and with the indicated values of I and O.

5.3. Resolution of Conflicts in the MULTISAFE Petri Net

In the Petri net for MULTISAFE (Figure 4), it is observed that the place PSM has three transitions that are in conflict. Whenever the PSM has a token the transitions t2, t4, and t8 are all firable, and the firing of any one disarms the other two. However, there are conditions to govern the firing order:

Transition t2 is meant to be fired if the token represents a request for a Protection Data Base, PDB, operation.

Transition t4 is meant to be fired if the token represents a request for a Data Base, DB, operation; t4 cannot be fired unless the PSM has made all required data independent checks.

Transition t8 is meant to be fired if the token represents a message to be returned to the UAM; t8 cannot be fired unless the PSM has made all required data dependent checks.

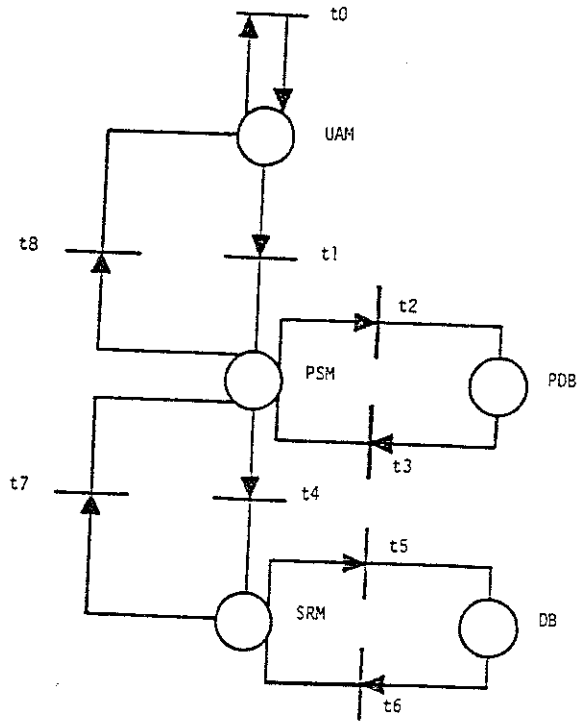

Figure 4. Petri Net Model of MULTISAFE

If a predicate reflecting the above mutually exclusive conditions is attached to each of the transitions t2, t4, and t8, then no conflict can occur. Different lines of communication between the modules are not needed if all messages contain a target module code [TRUER81], so that each transition can determine if a given message is a token upon which it should act. In other words, token content is used to evaluate the transition predicates.

Similarly, the place SRM is connected to two transitions that are in conflict whenever the SRM has a token. Predicates are also used at t5 and t7 to ensure the following conditions.

Transition t5 is meant to be fired if a token represents a request for a database operation on the database, DB.

Transition t7 is meant to be fired if a token represents a message to be be returned to the user or a result from the database, DB, to be passed to the PSM.

5.4. Assumptions and Definition of Security*

Several assumptions which help to focus this work are as follows:

Assumption 1. Controlled Physical Access
The system is accessed only via terminals. Consider that the system is physically protected by an impenetrable wall with small holes through which wires protrude to terminals in the outside world. Any information or signals can be sent in through those wires. If the modules of MULTISAFE are physically distributed, the equivalent of the impenetrable wall can be provided by anti-eavesdropping techniques such as encryption.

Assumption 2. PSM Programming Impervious to Modification
PSM programming is built into a Programmable Read-Only Memory (PROM). It is physically impossible for PSM programs to be modified by a user, via a terminal.

Assumption 3. Correct User Identification
User identification (authentication) is assumed to be done correctly. User identification is being attacked elsewhere as a completely separate problem [EVANA74, PURDG74, COTTI77]. Further, it is assumed that user identification can be reauthenticated whenever necessary or desirable, so that the relationship between user and terminal remains constant to MULTISAFE.

Assumption 4. Separation of Users in UAM
The UAM provides ordinary primary memory protection, so that multiple users are prevented from interfering with each other's processes, data, or messages.

Assumption 5. Limitation of Scope
Security in this work refers to access controls, and not information flow controls [DENND76] or inference controls [DENND79]. The flexibility of a generalized PSM processor, of course, admits to future addition of these and other controls.

Assumption 6. Discretionary Access Control

---

* This section is adapted from [TRUER81].

Discretionary authorization is assumed, being a more general case than non-discretionary (security levels), but not ruling out non-discretionary policies. An important implication is that many users are typically also authorizers.

All that follows, particularly the definition of security and its constituent conditions, applies to systems subject to the constraints of these assumptions. The definition of data security in this work emphasizes security explicitly as a relationship between authorization and enforcement:

Definition 1: A system is data secure if, in that system, the enforcement process allows the system to perform only those access operations which are specified by the authorizers.

At this point, it is useful to have a clear understanding of the term "access." In this work the following definition of access is used.

Definition 2: Access includes all operations used for reading, writing, or modifying data stored in the system.

Definition 1 can be restated as a set of four conditions:

Condition 1. Correctness of Authorization Process
All authorizations specified by the authorizers (and only by proper authorizers) are properly stored in the PSM Protection Data Base.

Condition 2. Correctness of Enforcement Process
All access decisions made by the PSM are correct with respect to (1) the access request, (2) the stored authorization information, and (3) the system state, including data values, at the time of the decision.

Condition 3. Complete Mediation
All access requests (authorization and database requests) are subject to enforcement (an access decision by the PSM).

Condition 4. Prohibition Against Spurious Data Transmission
No data may move between a user and either database (in either direction), except as a correct response to an access request.

These conditions are intuitively shown in [TRUER81] to completely embody Definition 1 as follows. (The informality of the definition and the conditions precludes a formal proof of completeness.) Condition 4 states that every data access is in response to an access request. (The requirement that it be a correct response also eliminates secondary trickery, such as a "Trojan Horse" in the SRM trying to deceive the PSM by sending prohibited data disguised as the response to some other request.) By condition 3, then, every data access that occurs is subject to an enforcement decision. Condition 2 implies that every data access is subject to an access decision that is correct with respect to the stored authorization information. Finally, by condition 1, every data access is subject to an access decision that is correct with respect to a proper authorizer's specifications of access privileges, and this is a restatement of Definition 1.

Condition 1 is assumed to be true; as the verification that authorization information, as specified by authorizers, is properly stored is outside the scope of this paper. Similarly, the correctness of the enforcement process (condition 2) is assumed. That is, it is assumed that the enforcement algorithm given in [HARTH76, HARTH80] is a correct algorithm and is implemented correctly.

The modeling power of Petri nets allows, by inspection of the net model in Figure 4, conditions 3 and 4 to be verified.

Condition 3 (all access requests are subject to enforcement) is verified since:

1. By inspection of Figure 4, $I(UAM,t1) = 1$, and $I(UAM,ti) = 0$, $\forall\ ti \in (T-\{t0,t1\})$. If t0 fires immediately following a user request, it

amounts to a trivial echo of the user's own message. Therefore, the only meaningful way a token bearing a user request can leave the UAM is for transition t1 to fire.

2. Since O(PSM,t1) = 1, upon the firing of transition t1, the token from the UAM must enter the PSM, where the request is subjected to enforcement.
3. Also, since O(pi,t1) = 0, $\forall$ pi $\in$(P-{PSM}), the token can go nowhere else but the PSM.
4. Therefore, all access requests are subject to enforcement by the PSM.

Condition 4 (no spurious data movement) is also easily established. Consider the binary relation "<" which imposes a partial order on the set of transitions of a Petri net in the following way: a < b if and only if the first firing of "b" is preceded by the first firing of "a" [AZEMA78], given an initial marking. (Technically, the relation "$\leq$" is required for a partial order. However, the equality condition is needed only to satisfy the reflexivity property of the partial order ($a \leq a$). In fact, a = b implies that a is identical to b, because no two transitions can fire at the same time. Therefore, the relation a < b is sufficient for partially ordering the transitions of a Petri net.)

A transition may fire only if all its input places have received a token. In the net of Figure 4, because every transition has only one input place, no transition can fire unless a message (a token) has arrived in that place. Thus, given an initial marking with no tokens in the net, the only way a message can originate is by the firing of transition t0, which is the source for the net. And transition t0 fires only upon the submission of a request by the user.

It is established above that, when t0 does fire, the token must arrive at the UAM and must then fire t1 (except for the trivial "echo" back to t0). Since $I(pi,t1) = 0$, $\forall\ pi \in (P-\{UAM\})$, the only way transition t1 can fire is by a token from the UAM. Therefore, any firing of t1 must be preceded by a firing of t0. By similar arguments the following partial ordering of transitions is established. For login and authorization operations:

$$t0 < t1 < t2 < t3 < t8$$

and for data access operations:

$$t0 < t1 < t2 < t3 < t4 < t5 < t6 < t7 < t8$$

These orderings depend, in part, on the predicates associated with the transitions (discussed in the previous section). For example, the requirement that t8 cannot fire until the PSM has made all of its data dependent checks ensures that $t2 < t3 < t8$ for data access requests. The state information in these conflict resolving predicates could, in fact, be represented by adding places and transitions to the Petri net of Figure 4. In such a case, the Petri net structure alone might be enough to determine the partial ordering.

Since data can move to or from the PDB only by the firing of transition t2 or t3, respectively, and to or from the DB by the firing of transition t5 or t6, respectively, and transition t0 precedes the firing of these transitions, no data may move between a user and either database, except by a sequence originating at t0, whose firing is always initiated by a user access request. Thus, no data (or any messages) will move spuriously within the system.

5.5. Message Flow in the Petri Net Model

In [TRUER81], the description of MULTISAFE message flow is divided into two cases: the single terminal (but multiple user) case and the multiple terminal case. It was established there that the multiple terminal case added only concerns relating to the proper separation and synchronization of user messages and resulting concurrent processes, and these problems are well solved elsewhere in the literature. Thus, analysis of message flow for the general case is reduced to analysis for the single terminal case. Therefore, for simplicity, it is here assumed that the system does not support more than one terminal and/or active user connected to the UAM. The nets of Figures 5, 6, and 7 are used to describe message flow for various database activities, from login to database requests and the responses to those requests.

5.5.1. Login Request

In Figure 5, once a user submits a login request at the terminal to the UAM (fire t0), a user work area is created within the UAM and the login message is sent to the PSM (fire t1). This message causes the PSM to set up a work area in the PSM memory for this transaction and, concurrently, to send a request to the PDB for the retrieval of the user's access franchise (fire t2) [HARTH80]. The user's access franchise is the user's effective access rights as dynamically computed for a given query from the authorization information. Upon completion of this PDB retrieval (fire t3), the PSM now contains the user's login request, as

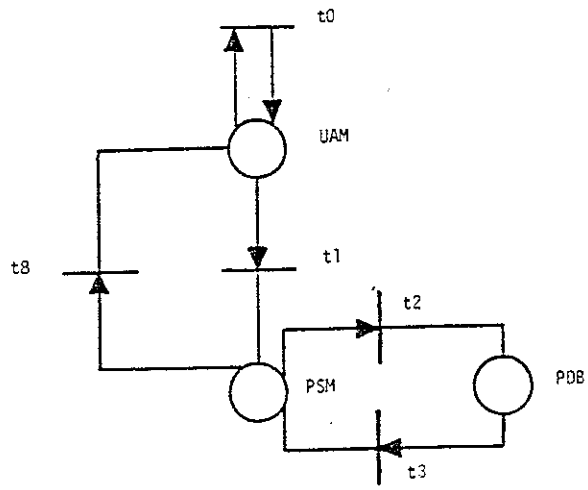

Figure 5. The Net for PDB Retrievals

well as the user's access franchise. Should the user have no access franchise, and hence have no rights whatever to access the system, a message is returned to the UAM denying login access (fire t8). The net terminates at the sink (transition t0). The SRM has not been entered, and no information held in either database has been violated.

5.5.2. Query and System Dependent Condition Checking

Further requests may be submitted by a user who has an access franchise stored in the PDB, and who passes all login access conditions. The user may be prompted to submit a request for an operation on the database, DB, or if the user is in an authorizer role, on the PDB (in either case, fire t8). Upon submission of a request to the PSM (fire t1), system dependent and query dependent access checks can now be made.

The database, the requested operation, and the names of the relation(s) to be operated upon are all available in the statement of the query. The access franchise relations are checked to determine if the query dependent access conditions are satisfied. If so, access is tentatively allowed (that is, no grounds have yet been found to deny access, but further analysis of the data dependent access conditions must be done before a final access decision can be made).

5.5.3. Authorization Request

If the user has requested an authorization operation, that is, a database operation on the PDB, the SRM module is not contacted and no information resident in the DB is accessed. Data dependent access conditions on the requested operation are now evaluated, should there be any such conditions on a PDB operation. This requires access to the PDB (fire t2), with information being retrieved from the PDB into the PSM (fire t3). If authorized, the required change in the PDB is effected or the requested information is returned to the user (fire t8). If the access operation is not authorized, the PSM sends a message to that effect to the UAM (fire t8). In either case, the net is depicted in Figure 5.

5.5.4. Data Base Access Request

For a request for an operation on the database, DB, the procedure that is followed depends upon the requested operation. If an update,

insert, or delete operation is requested, all data dependent access conditions must be evaluated prior to physical data access. For a retrieval operation, data dependent access checks are made after the physical access occurs, but before logical access is completed (i.e., before the data is passed back to the UAM and the user.) In order to evaluate the data dependent access conditions, several additional accesses to DB could be required. The net for this sequence is depicted

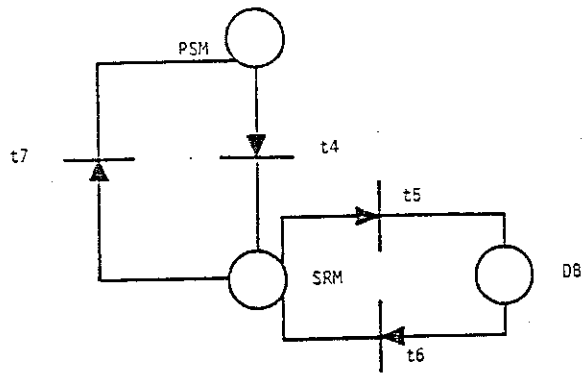

Figure 6. The net for DB Retrievals

in Figure 6. A separate working area in the PSM is created for the evaluation of these conditions (fire t4, t5, t6 and then t7).

If the enforcement process determines that the required operation is not authorized, it is not performed and a message is returned to the UAM to that effect. Otherwise, the operation proceeds as requested and the UAM is notified upon completion of the operation (fire t4, t5, t6, t7, and t8). The net depicting this sequence is shown in Figure 7.

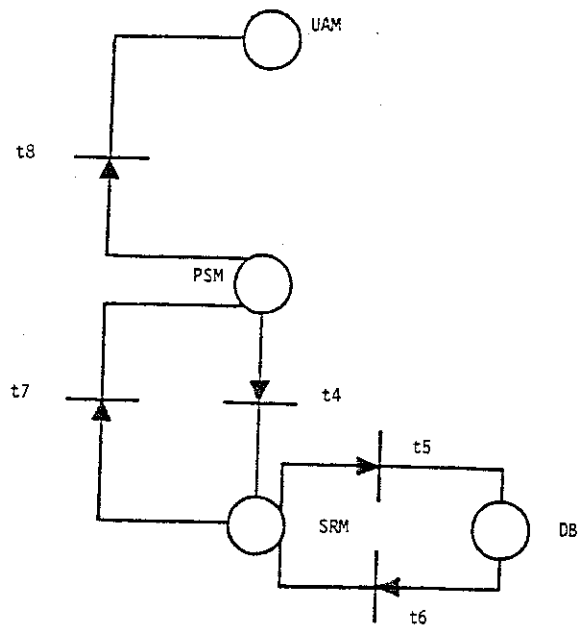

Figure 7. The Net for Requests after Successful Login

For a retrieval operation, the SRM can be sent the database request and can begin the retrieval operation while the PSM concurrently checks the data dependent access conditions. The PSM may submit additional requests for database retrievals to the SRM in order to evaluate the data dependent access conditions. These PSM-originated DB requests are queued at the SRM, the same as any other requests are, as described in [TRUER81]. The Petri net model is well suited for illustrating this asynchronous evaluation of access conditions. This aspect of Petri nets is seen again in Figures 8 and 9.

5.5.5. PSM-SRM Interplay for Data Dependent Checking

A retrieval request can cause considerable interplay between the PSM and the SRM. request is now considered. The PSM has a work area assigned to the query and a work area assigned to the evaluation of the data dependent access conditions. A semi-private memory configuration [TRUER81] can be used to ensure that no data used to evaluate access conditions is passed to the area of memory whose contents can be passed from the PSM to the UAM. Also, as the data retrieved in response to the query is sent from the SRM to the PSM, it can be placed in an area of memory whose contents can never be passed to the UAM until an access decision has been reached.

The Problem of Evaluating Data Dependent Conditions

The effective access condition with respect to a given user and a specific query can be a logical combination of the access conditions from several authorizations [HARTH80]. Thus, it is potentially complex, although in practice it usually is not. In general, one can assume that there are n different data dependent authorization conditions (predicates) say C1, C2, ..., Cn, each of which must be satisfied (i.e., they are ANDed) for the query currently under consideration. Each of these predicates is a boolean combination--using & (AND), | (OR), and ~ (NOT)--of simple data dependent conditions as classified in section 2.3. ('Simple' here is meant in the sense that each such condition corresponds to one dependency classification.) For example:

$$Ci = c(i,1)\ \&\ c(i,2)\ |\ c(i,3)\ \&\ c(i,4)\ |\ c(i,5)$$

The problem, then, is to evaluate (assign a truth value to) each of the Ci predicates. The problem of evaluating a Ci predicate reduces to the evaluation of all of its component c(i,j) simple conditions.

In the PSM, a template is constructed for each Ci, with the dependency class of each c(i,j) designated. A "slot" exists for the truth value of each c(i,j). The time at which each slot can be "filled" (assigned a truth value) depends on the binding time requirements of the corresponding c(i,j). The main criterion for binding at this point is whether the c(i,j) in question is one-to-one or one-to-n with the query responses. The one-to-n condition can be evaluated once for the entire query and, therefore, has constant (for the query) truth values. The one-to-one c(i,j)'s must remain variables because their truth values are determined by the individual response tuples (or by other tuples that are one-to-one with the response tuples).

Variable Simple Conditions

Several sources of data are used to evaluate the one-to-one variable conditions. First, there is the data directly associated with the query responses. The SRM, upon querying the database, DB, to fulfill the user's request, stores the responses into two queues. One queue (the query response queue, or QRQ), contains the attributes specified in the SELECT clause of the query (i.e., the target attributes), and is used to evaluate Class A and Class Ea data dependent access conditions (which depend only on those attributes requested).* The other queue (the query response tuple queue, or QRTQ), contains the entire tuples

---

* The reader may find the summary table of data dependency classes, given in section 2.3.6, a useful reference for what follows.

selected in response to the query, including values for the attributes that were not requested. These attribute values are kept in order to evaluate any data dependent access conditions whose value depends upon this information, that is, Class B and Class Eb data dependent access conditions.

There is one more source of information for evaluating varible conditions: other (non-response) data that is one-to-one with the response tuples. A separate query must be sent by the PSM to retrieve these values in the SRM. There are, therefore, often several concurrent database requests in operation at one time.

The variable simple conditions are summarized as follows:

1. Class A--dependent on QRQ
2. Class B--dependent on QRTQ
3. Classes C, Ed, and Ee--dependent on other data that is one-to-one with the tuples of the QRQ and QRTQ

Constant Simple Conditions

Constant conditions can depend on aggregate values of the QRQ (class Ea), on aggregate values of the QRTQ (class Eb), on aggregate values from other sources (class Ec), or can be dependent on data values other than any of these sources (class D). As mentioned earlier, queries may be broadcast from the PSM to the SRM for this information. The responses to these queries return at their own rate, allowing the truth values of the constant c(i,j)'s to be assigned and put into the corresponding slots in the template, for the Ci conditions.

Evaluation of Variable Simple Conditions

Because the variable simple condition truth values are one-to-one with the individual query response tuples, each Ci must have its variable simple conditions evaluated against each response tuple. In what follows, the basic procedure is to select an access condition whose constant simple conditions have already been evaluated, and apply it against each of the response tuples. The variable simple conditions of Ci are evaluated with respect to the tuple, and then the entire condition Ci can be evaluated for that tuple. Ci is then evaluated against the next tuple, and so on. The process is repeated for each other Ci after its constant simple conditions have become evaluated. The outcome of the procedure varies according to the enforcement policy.

Evaluation Under a Partial Enforcement Policy

Recall that under a partial enforcement policy an individual response to the user query is acceptable, if and only if it satisfies all the access conditions. Since the conditions Ci are ANDed, a response causing any Ci to be false is deleted from the QRQ. At the end of the process the QRQ contains only response tuples satisfying all Ci. Thus, the original QRQ may become reduced in size as the process progresses, decreasing the amount of checking to be done against each succeeding Ci. This evaluation of variable simple conditions proceeds concurrently with the evaluation of constant conditions for other Ci's. Since the constant simple conditions become evaluated in an unpredictable order, the Ci's become ready for variable evaluation in an unpredictable order. Because the Ci's are ANDed together, however, their order of application is immaterial.

The process of checking variable simple conditions, c(i,j), and their overall conditions, Ci, against response tuples (and other data that is one-to-one with the response tuples) is terminated in two different ways:

1. The QRQ becomes empty, resulting in no allowed responses to the query. This case can result from having every tuple fail to satisfy at least one access condition because of its variable parts, or from a single Ci that has no variable simple conditions and whose value is false. Checking is terminated immediately and an access denial is sent to the user.
2. The QRQ has been subjected to all the conditions C1, C2, ..., Cn, and what remains is a queue of responses that have satisfied all data dependent access conditions.

A Petri Net Model of Variable Simple Condition Evaluation under a Partial Enforcement Policy

Figure 8 is a net depicting this sequence of events. The data upon which variable simple conditions are dependent is gathered into three types of queues in the SRM: QRQ, QRTQ, and OTHER (as described earlier). These queues, having their elements in one-to-one with query responses, each have the same number of elements and each have them in the same order.

There are several steps in the process:

1. The data in the three queue types (QRQ, QRTQ, and OTHER) is sent in parallel from the SRM to the PSM through transition t0. It can be assumed that the parallel transmission of one tuple from each of the

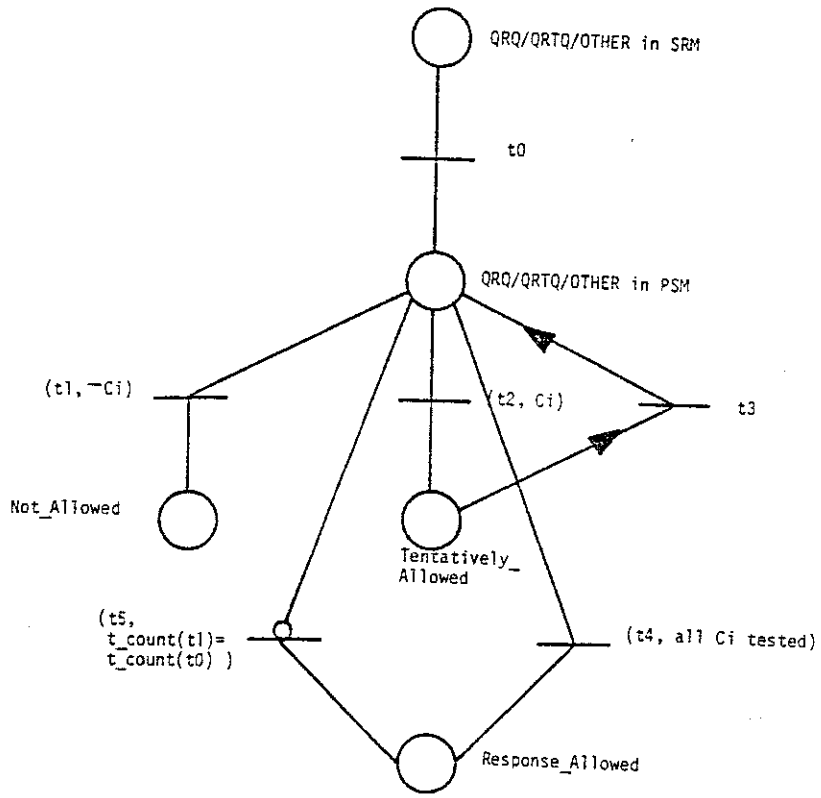

Figure 8. A Petri Net Model of Variable Simple Condition Evaluation under Partial Enforcement

queues corresponds to the movement of one token in the net. Thus, when t_count(t0) = the cardinality of QRQ, all the data is in the corresponding set of queues in the PSM.

2. At this point it is useful to introduce the concept of a queue of predicates at a transition of a Petri net. Their control is very much the same as that of the data queues at the places. The Ci's and their templates form a queue of predicates at t2. (There is a corresponding queue of predicates, ~Ci, at t1.) Those Ci's having their constant slots filled with truth values are selected one at a time to evaluate against the QRQ/QRTQ/OTHER data. One more slot per Ci can be provided and tagged to indicate which Ci's have been tested against the QRQ.
3. The tuples from QRQ/QRTQ/OTHER are considered one at a time, and either transition t1 or transition t2 fires for each tuple. The predicate for transition t2 allows it to fire only if the current condition Ci is true, placing the response in the Tentatively_Allowed place. Transition t1 fires if Ci is false, shunting the response off to the Not_Allowed place (a dead end).
4. As each tuple passed by t2 arrives in the Tentatively_Allowed place, it then fires t3 and goes back to the queues QRQ/QRTQ/OTHER. The first element of the queues is specially marked, so that a new Ci is selected at t1 and t2 each time it comes up.
5. If, at any time, there are no tokens in QRQ/QRTQ/OTHER and no tokens in Tentatively_Allowed, transition t5 (which has an inhibitor arc) fires, sending an empty queue to the Response_Allowed place. This causes an access denial to be sent to the user.
6. When the first element of QRQ appears as an input to t2 and there are no Ci's left untested in the predicate queue, transition t4 fires, delivering all the contents of QRQ to the Response_Allowed place to be sent to the user.

Implementation Aspects

Although this paper is not about implementation, a brief discussion of implementation with respect to Figure 8 might help the reader better to visualize the process. The moving of the data from the set of queues in the SRM to the corresponding set in the PSM may not actually require physical movement of data. In [TRUER81], a scheme is described to accomplish this by electronically switching segments of primary memory. The QRQ, QRTQ, and OTHER queues can be implemented, as an example, by arrays or linked lists, and they are traversed in parallel. The tuples to be deleted can simply be tagged as Not_Allowed.

Evaluation Under A Full Enforcement Policy

Under a full enforcement policy an individual response to the user query is acceptable if and only if all of the responses satisfy all the access conditions. If any individual response has a false value for any Ci, the entire access evaluation process can stop and a "no response allowed" message returned to the user. If all individual responses have a true value for all Ci, the process stops and the entire QRQ is allowed. Only a few simple changes are necessary to the Petri net of Figure 8 to illustrate the full enforcement policy.

Should a mix of full and partial enforcement be used, the enforcement policy used for any condition Ci must be attached to that condition by appending the enforcement policy to the stored authorization information in the PDB. If a condition Ci has a partial enforcement attribute, and an individual response to the query in the QRQ is evaluated and found to be unacceptable, the response is deleted from the QRQ, and the

process continues. If a condition Ci has a full enforcement attribute, and an individual response in the QRQ is evaluated and found to be unacceptable, the evaluation process can terminate and the QRQ emptied. Thus a mix of enforcement policies presents no new difficulties to the enforcement process.

6. CONCLUSION

A review of Petri nets and their properties has shown that Petri net theory is a valuable tool for the study of concurrent processes. Extensions to the basic structure of Petri nets were used to remove some limitations. These extensions were primarily in the area of conflict resolution by predicates associated with transitions, information content of tokens, and the ordering of tokens by considering places to be queues. Counting functions and queues of predicates at transitions were also introduced as further extensions, respectively for detecting completion of data flow through the net and for evaluating data dependent access conditions. Conditions on transition predicates were stated for ensuring that a net is conflict-free.

Access conditions were classified on the basis of various kinds of dependencies and binding time requirements. These classes facilitated the analysis of checking enforcement. A Petri net model of protection and enforcement in MULTISAFE was shown to be conflict-free and to preserve some of the the stated criteria for security. Due to the modeling

power of Petri nets, the satisfaction of these criteria for security was easily demonstrated upon inspection of the net. The use of Petri nets also provided insight into how access conditions can be implemented and evaluated.

Petri nets were found to be a powerful modeling tool for providing insight into system structure and into the relationships among system components. Although the decision power of Petri nets is an area of current research elsewhere [PETEJ77], the decision power of Petri nets is not of crucial importance in this work, since the reachability set is finite.

The modular nature of MULTISAFE makes extension into a distributed database environment a logical step. Research in the area of distributed protection of distributed data is now being conducted by the Database Research Group at VPI & SU.

ACKNOWLEDGEMENTS

The authors acknowledge earlier work on Petri net modeling by

Robert P. Trueblood.

REFERENCES

AZEMP78 Azema, P., Ayache, J.M., and Berthomieu, B., "Design and Verification of Communication Procedures: A Bottom-up Approach," 3rd International Conference on Software Engineering (Proceedings), May 10-12, 1978, Atlanta, Georgia, 168-174.

BALLE81 Balliet, Earl J., "Modeling of MULTISAFE Protection Enforcement Processes with Extended Petri Nets," M.S. Thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 (January 1981).

CHAMD76 Chamberlin, D. D., et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," IBM Journal of Research and Development, 20, 6 (November 1976), 560-575.

CONWR72 Conway, Richard W., W.L. Maxwell and Howard L. Morgan, "On the Implementation of Security Measures in Information Systems," Communications of the ACM, 15, 4 (April 1972), 211-220.

COTTI77 Cotton, Ira W., and Paul Meissner, "Approaches to Controlling Personal Access to Computer Terminals," in Tutorial on Computer Security and Integrity, pub. EH 1124-8 by IEEE Computer Society (1977).

DENND76 Denning, Dorothy E., "A Lattice Model of Secure Information Flow," Communications of the ACM, 19, 5 (May 1976), 236-243.

DENND79 Denning, Dorothy E., Peter J. Denning, and Mayer D. Schwartz, "The Tracker: A Threat to Statistical Database Security," ACM Trans. on Database Systems 4, 1 (March 1979), 76-96.

DOBKD79 Dobkin, David, Anita K. Jones, and Richard J. Lipton, "Secure Databases: Protection Against User Inference," ACM Trans. on Database Systems 4, 1 (March 1979), 97-106.

ELLIC77 Ellis, C.A., "A Robust Algorithm for Updating Duplicate Databases," Proceedings, 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, 1977.

EVANA74 Evans, Arthur, Jr., and William Kantrowitz, "A User Authentication Scheme Not Requiring Secrecy in the Computer," Communications of the ACM, 17, 8 (August 1974), 437-442.

HARTH76 Hartson, H. Rex, and Hsiao, David K., "A Semantic Model for Data Base Protection Languages," Proc. of the International Conf. on Very Large Data Bases Brussels (September 1976).

HARTH77 Hartson, H. Rex, "Dynamics of Database Protection Enforcement—A Preliminary Study," Proc. of the IEEE Computer and Software Applications Conf. Chicago (November 1977), 349-356.

HARTH80 Hartson, H. Rex, "Implementation of Predicate-Based Protection in MULTISAFE," Technical Report CS80010-R, Department of Computer Science, V.P.I.&S.U., Blacksburg, Va. 24061

HARTH81 Hartson, H. Rex, "Database Security—System Architecture," accepted for publication in Information Systems.

NUTTG72 Nutt, G.J., The Formulation and Application of Evaluation Nets, Ph. D. Dissertation, Univ. of Washington, 1972.

PETEJ77 Peterson, J.L., "Petri Nets," ACM Computing Surveys, 9, 3 (September 1977), 222-252.

PURDG74 Purdy, George B., "A High Security Log-in Procedure," Communications of the ACM, 17, 8 (August 1974), 442-444.

STONM74 Stonebraker, Michael, and Eugene Wong, "Access Control in a Relational Data Base Management System by Query Modification," Proceedings of the ACM Annual Conference (November 1974), 180-186.

TRUER79 Trueblood, Robert P., Multiprocessor Architectures for Supporting Secure Database Management, Ph. D. Dissertation, Department of Computer Science Virginia Polytechnic Institute and State University, Blacksburg, VA (June 1979).

TRUER81 Trueblood, R.P., Hartson, H.R., and Martin, J.J., "MULTISAFE - A Modular Multiprocessing Approach to Secure Database Management," Submitted for publication.