Technical Report CS80010-R

Implementation of Predicate-Based Protection in MULTISAFE

H. Rex Hartson

October 1980

Revised May 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.

Implementation of Predicate-Based Protection in MULTISAFE*

H. Rex Hartson

Virginia Polytechnic Institute and State University

Blacksburg, VA 24061

Abstract: This paper reports some implementation work done within the MULTISAFE database protection research project group at Virginia Tech. It describes the evolution of an approach to database security from a formal model of predicate-based protection, through an implementation model, to an on-going implementation. The implementation model is based on a relational database approach to the management of protection information (stored representations of authorizations). Classes of access decision dependency are reviewed. Protection policies, design decisions, and special implementation problems are discussed. Detailed examples are used to illustrate the use of this flexible and generalized approach to database security within the MULTISAFE system architecture.

Keywords and phrases: MULTISAFE, database management, protection, security, implementation, relational, data dependency, predicate based

---

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

1. INTRODUCTION

Throughout the field of data management, the areas of security and protection continue to evolve and improve, in response to the continuing demand for more flexible and effective data security. The predicate-based model of protection in [HARTH76a] was one of the first in which access decisions were generally sensitive to the state of the system, including a dependency on data content. MULTISAFE [TRUER80] is a system architecture for secure data management. The discussion of MULTISAFE in [TRUER80] gives a detailed account of intermodule communication in terms of procedure calls and returns and proposes in detail an intermodule message system as a vehicle to carry those calls and returns. No indication is given there, however, as to how access decisions are implemented. This paper brings three things together in an implementation model:

a) the predicate-based model of protection,

b) the MULTISAFE architectural approach to database security, and

c) a relational database approach to storing, retrieving, and combining authorization information in order to make access decisions.

Many people in the MULTISAFE project group have contributed to this implementation model. The work reported here is an effort providing a point of departure from which two other research efforts have proceeded in very different directions. Deaver [DEAVM81] deals with a cost/performance model of MULTISAFE protection and Balliet [BALLE81] with a Petri net model of MULTISAFE protection. Talbott is presently implementing the work described in this paper [TALBT81] and has contributed significantly to the syntax of the commands and the structure of the system directories and authorization information.

2. A PREDICATE-BASED PROTECTION MODEL*

Hartson's predicate-based model of protection began as a semantic model for protection languages [HARTH75, HARTH76a, HARTH76b]. It has both an authorization process and an enforcement process. Protection requirements (authorizations) are presented to the authorization process. Subsequently, access requests are presented to the enforcement process which, by consulting the authorization information, renders an access decision.

An emphasis on authorization draws several other concepts into the model. For example, the "authorizer" emerges as an entity distinct from the "user." Authority can now be decentralized, if desired, from the bottleneck of a single database administrator. The model introduces a general dependency of the access decision on system state, rather than using the fixed relationship between users and data afforded by the more conventional access matrix. Also, the concept of "ownership" acquires a

* This section is adapted to an implementation model from the formal model of [HARTH76a]. The following notation will be used. Underscored upper case letters will denote sets while lower case letters, possibly subscripted, will denote set elements. Finally, subsets will be denoted by upper case letters, possibly subscripted. Subscripts will follow the associated variable, delineated by square brackets when necesary for clarity. for example, s, s[1], or s1 is an element of S, while S = {s1,s2} is a subset of $\underline{S}$ = {s1,s2,s3}.

role more general than its traditional one.

2.1. Sets and Predicates

Several sets are involved in the model. An important set is $\underline{U}$, the set of individual users, who will be making access requests. Sets of users will be called user groups. There is also a set of individuals who will be granting authorizations, the authorizers, $\underline{A}$. An authorizer is an owner of data and has the responsibility of making authorizations, i.e., of determining who may share in the access of that data and in what manner. In the literature the term "user" is almost universally used to denote both users of the database system and users of the protection system. The distinction is emphasized here by using the term "authorizer."

The next set of the model is the one that contains data. $\underline{D}$ is the set of all data in the database. Real world access requests and data definitions deal with subsets of the database, as well as with individual data elements. Since this protection model is not tied to a particular model of data, the way in which elements and subsets of $\underline{D}$ are defined will be left unspecified until the specifics of the implementation are discussed in section .

The operations a user may perform on the data also comprise a set, $\underline{O}$. Examples of operations are: OPEN, CREATE, READ, WRITE, APPEND, UPDATE, DELETE, EXECUTE, COMPARE, RETRIEVE, OWN, and SUBOWN. The model is extensible in that it allows new data types and operations to be added.

A set can be described explicitly by enumeration of its members or implicitly by a predicate, or condition, used as its characteristic function. For example, let c(b,B) be a condition defining set B. Membership of any potential element b in set B is dependent of c(b,B) in the following way:

$$B = \{b \mid c(b,B) \text{ holds}\}$$

Since the truth value of c(b,B) can, in general, vary with time (with database system state), membership in B is a dynamic property.

If the set in question is U, a group of users, then c(u,U) determines whether or not an individual user u is a member of user group U at a specific instant in time and c(u,U) is called a <u>user group defining condition</u>. Similarly, c(d,D) is called a <u>data defining condition</u>, where d is a data element (such as a single record, tuple, or field value) and D is some subset of $\underline{D}$.

Another important use of a predicate is an an <u>access condition</u>, specifying the condition (involving the system state) under which a given type of access is to be allowed. The set of all access conditions known to the system at any time will be denoted by $\underline{C}$. Within this model there are no restrictions on what type of variables can be used in any of the conditions. However, in the implementation model which follows each condition type has been restricted to certain classes of variables without any loss of generality.

A user group can be defined by a list of user identifiers. The examples which follow illustrate a second way--by the use of predicates. The simplest of these definitions is the one for a standing group called <u>GENERAL</u>. Its defining predicate is:

c(u,GENERAL): "true"

Thus, because this predicate has a constant "true" value, every user is implicitly and unconditionally a member of the GENERAL group, sharing its minimum level of privileges. This is an answer to the problem of wasting storage to represent, for example, the rights of everyone to use a public file [LAMPB71].

The following is typical of the user group defining conditions which can be dynamically evaluated.

c(u,ul): dept(u)=13 v project(u)='design'

When a user logs in, if s/he is either in department 13 or part of the design project, s/he becomes a member of user group Ul (until s/he logs off or until his/her membership in Ul is tested again) and is given its access rights. (See the footnote in section regarding symbols denoting logical operators in predicates.)

References to data, whether for access or for access control, are very dependent upon the data model and its method of data definition. In most real systems it will often be desirable to use a combination of explicit naming and implicit predicate-based definitions. The explicit naming, such as is done by providing relation and attribute names, describes a domain in which a data dependent predicate is to be applied. Following is an example which uses both the relation name "personnel" and content values for attributes named "dept" and "salary:"

```
c(d,D3): RELATION(d)='personnel' & dept(d)=7
                   & salary(d)<20000
         & (ATTRIBUTE(d)='name' v ATTRIBUTE(d)='address')
```

This data definition refers to a "fragment" of the PERSONNEL relation, restricted to tuples with a DEPT value of 7 and SALARY values of less than 20,000, projected to the NAME and ADDRESS atttibutes.

Access conditions--provisos which must be met before certain accesses are allowed--can be associated with global system variables such as the time-of-day clock, modes of operation, status indicators, flags, and internal codes. Many of these indicators contain information known to the database system about the current transaction. As an example, suppose that certain accounting information can be entered only on Fridays. For 1981, this is represented by the following access condition:

$$C: \text{MOD}(\text{day},7) = 2$$

where MOD is the modulo function and day is the Julian day number on the system calendar. Access conditions can also be data dependent predicates.

2.2. Access Requests and Authorizations

The purpose of a database system is to serve the user in response to an <u>access request</u>. An access request, or query, is a triple:

$$q = (u,o,D)$$

where $u \in \underline{U}$, $o \in \underline{O}$, and $D \subset \underline{D}$. This represents a request by an individual user u for a single operation o to data subset D. Of the three elements, the user actually provides only o and D. The user identifier, u, is supplied by the system in an "unforgeable" manner.

An instance of an <u>authorization</u> is a 5-tuple:

$$p = (a,U,O,D,c)$$

where $a \in \underline{A}$, $U \subset \underline{U}$, $o \in \underline{O}$, $D \subset \underline{D}$, and $c \in \underline{C}$. This 5-tuple represents a declaration by authorizer a that each and every individual user in user group U may do any or all of the operations in O to any or all of the data elements in D only if the access condition (predicate) c has a value of "true." For reasons of accountability, each authorization is marked with the identity of the authorizer who created it. Only that authroizer can modify or delete that authorization. However, a subowner (one to whom ownership has been explicitly granted by the owner who created the data) of a given subset of data, D, can issue other authorizations governing the access to D. Thus, several different authorizers can each create an authorization for the same user group, operation, and subset of the database, but perhaps with different access conditions. A subowner cannot grant OWNership to others.*

Consider this protection requirement, taken from [CONWR72]: A user may see and update only "financial" parts of each record in a given personnel file, and only between 9 a.m. and 5 p.m. from a specific terminal in the payroll department. The user group is defined by:

c(u,U6): dept(u)='payroll' & terml(u)='payoffice'

The data to be protected is:

c(d,D5): RELATION(d)='personnel' &

(ATTRIBUTE(d)='salary' v ATTRIBUTE(d)='rate')

assuming the salary and rate attributes are the "financial" parts of the tuples. The access condition is:

c7: TIME > 0900 & TIME < 1700

---

* These ownership rules are a matter of high level policy, and other policies are possible. See section , later, for a discussion of policies.

The authorization which then ties these definitions together is:

p4 = (a,U6,{READ,WRITE},D5,c7)

2.3. Authorization and Enforcement

Before the authorization process translates the protection language expressions of authorizers into internal representations of access control information, it first validates the right of the authorizer to make the authorization. The process controls granting, as well as revocation, of rights; keeps lists of authorizer-created definitions; controls the display of access control information; and keeps a journal of all transactions with the protection system.

Before proceeding with the enforcement process, it is convenient to define some projection functions which operate on n-tuples. In general, where p = (x1,x2,...,xn), a set of projection functions i(p) = x[i], is defined for i = 1,2,...,n. Since the tuples of this model carry specific element names, mnemonic meaning is better served by the use of the names of the elements instead of a general scheme with subscripts. Therefore, for p = (a1,U3,O7,D4,c6): a(p)=a1, U(p)=U3, O(P)=O7, D(p)=D4, and c(p)=c6. Also, let $\underline{P}$={p1,p2,...,pk} be the net collection of valid authorizations received up to a given time. The enforcement process is now stated in the context of a request, q = (u,o,D), and the access decision is computed. (A similar set of projection functions is used on q: u(q), o(q), and D(q)). An example will be developed concurrently to illustrate the application of enforcement.

Assume that, at a given point in time, the following authorizations and the associated definitions exist in the system:

$\underline{P}$ = { p1=(a,U1,o,D2,"true"), p2=(a,U1,o,D3,c1),
p3=(a,U2,o,D1,c2), p4=(a,U2,o,D4,c3),
p5=(a,U3,o,D1,c4), p6=(a,U4,o,D1,c5),
p7=(a,U4,o,D2,c6), p8=(a,U4,o,D3,c7) }

Without loss of generality, a single authorizer a and a single operation o are assumed. The remaining information in $\underline{P}$ can be represented as the matrix of access conditions in Figure 1.

	D1	D2	D3	D4
U1	F	T	c1	F
U2	c2	F	F	c3
U3	c4	F	F	F
U4	c5	c6	c7	F

Figure 1. Matrix of Access Conditions

Now, consider a request q = (u,o,D). The enforcement algorithm follows.

(1) Determine from the set of known user groups those groups to which the requesting user u(q) belongs.

Example. This step is done by searching lists of user identifiers (explicit user group definitions) for u(q) and by evaluating the predicates of the implicit user group definitions. Assume that it is determined that u(q)∈U2 and u(q)∈U4.

(2) Collect from $\underline{P}$, the set of all authorizations received to date, those authorizations which have the user groups determined in step (1) as elements. The result, called the <u>franchise of the user</u>, is given formally as:

$$F(u) = \{p \in \underline{P} \mid u(q) \in U(p)\}$$

Steps (1) and (2) can be done once per terminal session, at log-in time.

Example. For the example, this is {p3,p4,p6,p7,p8}, since these are the authorizations which mention U2 or U4.

(3) Determine from the set of known data subsets (already defined for purposes of authorization) those data subsets which have data in common with the data subset requested in q.

Example. This determination is dependent on the data model and its method of data definition. Suppose, for the example, it is found that D(Q) is found to have elements in common with D1 and D3.

(4) Determine the set of authorizations that name data subsets found in step (3). Formally, this set of authorizations is denoted as:

$$\{p \in \underline{P} \mid D(q)\ \underline{XN}\ D(p) \neq \{\}\ \}.$$

Example. Here, the set which mentions D1 or D3 is {p2,p3,p5,p6,p8}.

(5) Determine the set of authorizations which specified o(q) as a data operation. Formally, this is denoted as:

$$\{p \in \underline{P} \mid o(q) \in O(p)\}.$$

Example. Here, this is all of $\underline{P}$, since only one operation is being considered.

(6) Determine those authorizations common to the sets found in steps (2), (4), and (5). As this is the set of authorizations which pertain to this specific request, it is called F(q), the franchise (of the user) with respect to the query q. Formally,

$$F(q) = \{p \in \underline{P} \mid u(q) \in U(p)\ \&\ D(q)\ \underline{XN}\ D(p) \neq \{\}\ \&\ o(q) \in O(p)\}$$

All of the enforcement process so far is summarized in this one expression. F(q) is the set of authorizations which are applicable to q (i.e., which participate in the access decision for q).

Example. In the example, the franchise for q is {p3,p6,p8}.

(7) The set of data subsets named in the authorizations of F(q) is called D*(q), the "data reference" of q. Let D*(q) = {D1*, D2*,...,D[d]*}. This step is to determine if the requested data subset D(q) is covered by the data reference D*(q); i.e., it must be true that $D(q)\ \underline{c}\ \underline{UN}\ D[i]^*$, $\forall i \in [1,d]$. That is, every element of D(q) must be contained in some member of D*(q).

Explanation. Having some overlap with authorized data is not enough to allow all of the requested data to be accessed. All of the requested data must be subject to some authorization. At this

---

Throughout the paper, the following notation is adopted:

$\underline{UN}$	'set union'
$\underline{XN}$	'set intersection'
$\underline{\&}$	'logical AND operator'
v	'logical OR operator'
⊕	'OR over a bit-by-bit AND' (used in section)
{}	'the null set'
∈	'is a member of' (set membership)

point, under a policy of "full enforcement" [HARTH77] (all data may be accessed or none is), failure of the covering check terminates the enforcement process with a flat rejection. Under a "partial enforcement" policy (those parts covered can be further considered for access), the non-covered parts are eliminated by setting:

$$D(q) = D(q)\ \underline{XN}\ D^*(q)$$

(8) Partition the set F(q) into equivalence classes based on the relation such that two authorizations are in the same class if and only if they specify the same data subset. Formally, this is accomplished as follows. Partition F(q) into d equivalence classes such that the i-th class is:

$$F[i](q) = \{p \in F(q) \mid D(p) = D[i]^*\}$$

$\forall i \in [1,d]$. Construct a temporary composite authorization for each class F[i](q):

$$p'[i] = (-, u(q), o(q), D[i]^*, v\{c(p[j] \mid p[j] \in F[i](q)\})$$

where "v" denotes the logical OR of the set of access condition predicates, c(p[j]), over all j such that $p[j] \in F[i](q)$. (A "-" in the authorizer position of the 5-tuple indicates a "don't care" value.) The franchise for q is now the collection of all these composite authorizations:

$$F(q) = \{p'[i] \mid i=1,2,\ldots,d\}$$

Example. The partitions are {p3,p6}, corresponding to D1, and {p8}, corresponding to D3. For the example, F(q) becomes {(-,u(q),o(q),D1,c2vc5), (-,u(q),o(q),D3,c7)}. The result of this step, for each data subset in D*(q), is an OR of the user's rights over all groups of which s/he is a member. This may be easier to see in the matrix of access conditions, Figure 2.

	D1	D2	D3	D4
U1	F	T	c1	F
U2	c2	F	F	c3
U3	c4	F	F	F
U4	c5	c6	c7	F

```
            |          |
            |          |
            V          V

         c2 v c5      c7
```

Figure 2. The OR Function Over User Groups

(9) Find EAC(q), the <u>effective access condition corresponding to the request q</u>, by performing the logical AND over the access conditions of the members of F(u,q):

$$EAC(q) = \&(c(p'[i])),\ \text{over } i=1,2,\ldots,d$$

Example. EAC(q) = (c2 v c5) & c7

(10) Evaluate the effective access condition and render an access decision: permit the requested access if EAC has a value of "true"; deny if otherwise.

The above sequence of ten steps, while useful for explaining the enforcement process, is not followed directly by the implementation. Some short-cuts will be described later in section . Also, the relational model has turned out to be an ideal environment in which to implement this model. The collection of authorization tuples is a relation. Many of the steps of the enforcement algorithm are nothing more than relational calculus descriptions of queries over the authorization relation, operating on subsets of its tuples.

In [HARTH76b] this basic model of protection is extended to history keeping (access decision dependency on previous database events) and auxiliary program invocation (procedures triggered by database events).

3. CLASSES OF ACCESS DECISION DEPENDENCY

The generality introduced in the model of [HARTH76a] 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. In [BALLE81] access conditions (predicates) are classified into three broad categories:

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

The truth value of a system dependent condition is 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. A query dependent access condition can limit the relations upon which the user can operate, the attributes which the user can retrieve, and the attributes which can be used in the selection predicates. Since the names of the requested relations and attributes are given in the query, the truth values of the predicates are ascertained from the queries themselves. As an example of a query dependent condition, consider an authorization which allows the selection of names from an employee relation as long as the selection predicate does not specify salary values.

A condition is a data dependent condition if its value cannot be ascertained without a retrieval (or perhaps several retrievals) from the database. In [BALLE81] several classes of data dependency are identified depending on which of the following sources provides the data necessary for evaluating the condition:

1) retrieved attributes of retrieved tuples
2) non-retrieved attributes of retrieved tuples
3) tuples in relations other than those being queried by user
4) aggregate data (sum, count, average, or data otherwise derived from, but not directly stored in, the database)

An example of an aggregate dependent access condition is one that requires the average age of employees retrieved to be less than 30.

Dependency has an effect on the binding time requirements of predicates. The time at which each part of an access condition can be evaluated 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. System dependent conditions 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 (and even earlier), to the time the results of a service request are returned to the user. Evaluation of query dependent conditions can be performed upon receipt of a query, leading to a relatively early binding time. If a condition has a data dependency, the binding time depends upon the class of the dependency. For example, conditions dependent upon retrieved tuples must be bound repeatedly as the responses to a query return from the database.

4. INTRODUCTION TO MULTISAFE

A MULTIprocessor system for supporting Secure Authorization with Full Enforcement (MULTISAFE) for shared database management is being developed [TRUER80] by Trueblood at the University of South Carolina and Hartson at Virginia Tech. 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)

The UAM can support many different types of user interfaces. The SRM can support the popular data models (relational, network, or hierarchical), or it can be a specialized database machine. MULTISAFE can be implemented on one or more processors. In an ordinary single processor system these three modules function sequentially in an interleaved fashion. In a multiprocessor system all three modules can 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 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. The physical isolation of modules in MULTISAFE constrains all intermodule communication to well-defined channels and eliminates all "back door" access paths in the software. However, physical isolation by itself is not a guarantee of security. Security depends upon the correctness of the mechanisms within the PSM and upon the correctness of intermodule communication. Verifying the correctness of the authorization and enforcement processes of the PSM can now be isolated as a separate endeavor. Methodology is

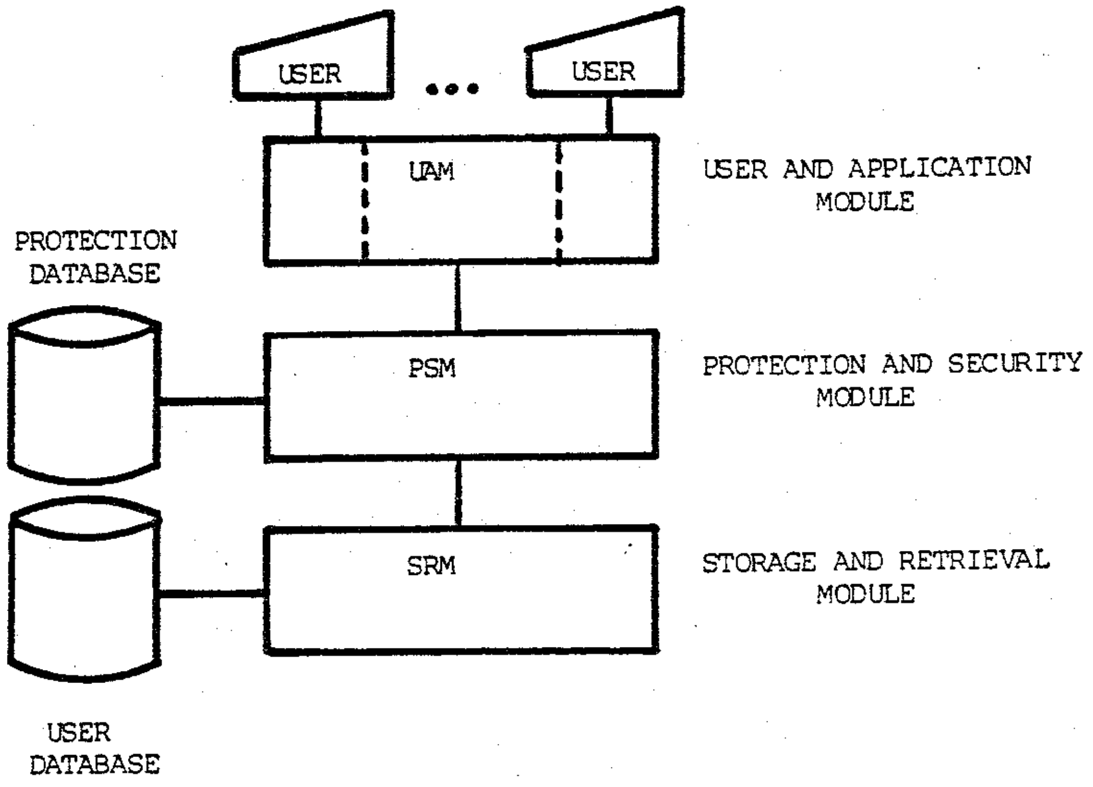
USER
USER
UAM
USER AND APPLICATION MODULE
PROTECTION DATABASE
PSM
PROTECTION AND SECURITY MODULE
SRM
STORAGE AND RETRIEVAL MODULE
USER DATABASE

Figure 3. Logical Relationships Among MULTISAFE Modules

currently available to show that the specifications of these processes do not violate protection policies (axioms) and that the programs are faithful to the specifications. Special attention is given to intermodule communication in [TRUER80].

In MULTISAFE, messages are sent between modules via encapsulated data types. 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 [TRUER80] 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 log-in. 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) additional binding times [HARTH77] for these conditions.

In this paper the concern is with the implementation of the authorization and enforcement mechanisms within the PSM of MULTISAFE.

5. A DATABASE APPROACH TO PROTECTION IMPLEMENTATION

5.1. The Relational Makeup of the Protection Database

The database system used by the accessors of data is a relational DBMS, called the Mini Data Base (MDB), implemented at VPI & SU on the VAX 11/780 computer. The authorization information (called the Protection Data Base, or PDB) is stored in relational form, as is the authorization information of System R [GRIFP76]. The PDB uses a fixed set of relations and a specially modified version of the MDB for storage and retrieval. The fixed set of relations (initially in the system and, thereafter, never created or destroyed) in the PSM includes USERS and AUTHS. The USERS relation contains user identification and account information for each individual user, as well as definitions for all user groups. The AUTHS relation contains the authorization information, i.e., who has the right to do what operations on what data and under what access conditions. In other words, AUTHS contains the set <u>P</u> of section 2.3. SCHEMAS, a fixed relation in the SRM, provides the database directory containing the definitions of the user-created data relations.

5.1.1. Data Definitions

The database system directory is contained in the relation SCHEMAS. SCHEMAS conceptually consists of variable length tuples of the form

SCHEMAS (RELATION, NBR_ATTRS,

NAME_OF_ATTR_NO_1, TYPE_OF_ATTR_NO_1, ATTR_NO_1_INDEXED,

NAME_OF_ATTR_NO_2, TYPE_OF_ATTR_NO_2, ATTR_NO_2_INDEXED, ...,

NAME_OF_ATTR_NO_N, TYPE_OF_ATTR_NO_N, ATTR_NO_N_INDEXED).

This conceptual view of SCHEMAS is what the user sees. The implementation details, described in the following paragraph, are transparent to the user.

The underlying implementation of the relation SCHEMAS has been simplified by separating it into two parts. The Data Base System Directory (DBSD) contains the relation name (RELATION) and number of attributes (NBR_ATTRS) for each relation in the system. In addition, the DBSD contains a pointer to a Relation Directory (RD) which contains the other attributes of the SCHEMAS relation. There is one DBSD for the entire system and one RD for each data relation. In addition to these attributes from the SCHEMAS relation, the RD contains a count of the number of tuples in the relation and pointers, into the data storage area and to secondary index B-trees, which are needed to retrieve the data tuples.

5.1.2. User and User Group Definitions

In MULTISAFE individual users are identified by the standard attributes: userid, account number, project name, password, etc. Users can also be grouped together to simplify the authorization of access to shared data. User groups are a generalization of the Multics (and others) "project" concept. The USERS relation is a 6-tuple of information about system users and groups of users with the form:

USERS (GROUP_NAME, USER_ID, ACCT_NO, TERM_NO, PROJ_NAME, PASSWORD).

In MULTISAFE, users can be grouped either explicitly by enumerating the members of the set or implicitly by stating a set-defining predicate. At log-in time, all such predicates are tested against the user's characteristics. In this particular implementation, user group defining predicates are restricted to the use of attributes found in the USERS Relation (e.g., user_id, account number, terminal number, and project name). The user becomes a member of each group for which the corresponding predicate is true. More discussion of the use of these predicates is given in section . Implicit membership holds for the duration of the terminal session until logout, or until such time that the group defining conditions need to be re-evaluated.

5.1.3. Authorization Information

The AUTHS relation contains the set of all authorizations. Each row of AUTHS is a 6-tuple of information governing which users or groups of users have access to any of the system relations. The AUTHS relation has the form:

AUTHS (AUTHORIZER, GROUP_NAME, OPERATIONS,

RELATION, ATTRIBUTES, ACCESS_CONDITION).

AUTHS contains a dynamic record of the current state of authorization information as a result of the cumulative effect of all valid authorizations received up to a given point in time. In each tuple of AUTHS the attribute RELATION and the bit-coded attribute ATTRIBUTES denote a data relation or a projection of a data relation. The correspondence of the bits of ATTRIBUTES to attributes of the data relation is given by the order of the attributes as defined when the relation was created. A value of '1' in a bit position means that the corresponding attribute is included in the projection defined by that AUTHS tuple. A value of '0' means that it is not. Since the operation set is fixed, for convenience of implementation, the OPERATION attribute is bit-coded, too. To illustrate the semantics of AUTHS, assume that a data relation called EMP has been defined by a user named SMITH. Consider the con-

AUTHS:

	AUTHO-RIZER	GROUP_NAME	OPERATIONS	RELATION	ATTRI-BUTES	ACCESS_CONDITION
1	SMITH	SMITH	OWN,RETRIEVE,UPDATE,...	EMP	111111	T
2	SMITH	GROUP1	RETRIEVE	EMP	010010	C1
3	SMITH	GROUP2	RETRIEVE	EMP	011000	C2

tents of AUTHS shown below.

For the relation EMP, the ATTRIBUTES bits represent the attributes of EMP in the order they were defined:

EMP (EMP#, NAME, SALARY, BIRTH_YEAR, DEPT, YRS_SERVICE)

The first tuple of AUTHS denotes unconditional ownership (including full access rights) of EMP, the relation created by Smith. This tuple was entered into AUTHS as one result of Smith's command that created EMP. The OWN right is what allowed Smith to grant RETRIEVE rights to GROUP1 and GROUP2, represented by the second and third tuples in AUTHS. The second tuple defines a projection of the EMP relation containing only attributes NAME and DEPT; the third, a projection over NAME and SALARY. C1 and C2 are the names of the access conditions for the two grants. A value of "T" for ACCESS_CONDITION denotes an unconditionally "true" truth value in the first tuple of AUTHS.

To simplify checking of access conditions when retrieving data, the access conditions are stored in a separate relation with the ACCESS_CONDITION attribute of AUTHS containing pointers into the CONDITIONS relation. The method of evaluation for a condition is determined by its dependency class (see section 3). The treatment of access conditions is further discussed in section .

5.2. Philosophy of Syntax Design

The MULTISAFE system uses a relational database system, the Mini Data Base (MDB), for the SRM (and part of the UAM). In an initial design, the MDB command language, modeled after SEQUEL [CHAMD76], included the usual commands for creating and dropping relations; inserting, deleting, and updating tuples in data relations; and loading and storing data files to and from relations. The authorization part of the language provided the means to create and drop new users and groups of users and to grant, revoke, display, and modify access rights of users and user groups.

The use of a relational database approach to protection drew the syntax of the authorization and data definition commands close to that of the commands for manipulating the data relations. The logical conclusion was a redesign of the command language taking the point of view that the authorizer is just a user of the system who is using authorization relations instead of data relations. Previously specialized commands to create and drop relations (tuple types) are now viewed as ordinary INSERT and DELETE operations on tuple instances in the special relation SCHEMAS, which contains the database directory information. Thus, the right to create relations appears in AUTHS as the right to do an INSERT operation on the SCHEMAS relation. And the right to grant the right to create and drop relations is represented by the OWN (ownership) operation on the SCHEMAS relation. This redesign produced a much simpler language with fewer commands, rather than a different set of commands for each of the three functional areas (data definition, data manipulation, and authorization). Intuitively, these simplifications would seem to enhance both "string simplicity" and "structural consistency" described by Reisner in [REISP81]. The latter, a measure of whether tasks that are perceived to be similar by a user are described by similar sequences of commands, is especially important to the usability of a language. The system interactively prompts the inexperienced user, who deals only with data relations, for all needed information. Authorizers and users defining new relations, being more experienced, can easily extend their view to include the added system relations, without having to learn new commands.

Another advantage to the database approach is that it offers data independence to the protection enforcement processes. Because AUTHS and USERS are accessed via database operations, information about their structure is not put into the accessing programs, but is in the data definition of these relations. Many later changes to the make-up of these tables can be limited to their definitions. For example, suppose there are x bits in ATTRIBUTES of AUTHS, and it is desirable to double the number of attributes in an MDB relation. Now, 2x bits are needed in the ATTRIBUTES field of AUTHS. This change requires only that AUTHS be redefined and reloaded. Existing tuples of AUTHS and existing programs that deal with AUTHS are not affected.

5.3. Summary of Commands

This section contains some incomplete examples which give just an overview of the form of the command language interaction. More detailed examples and explanations of the meanings of the commands are given in section .

In the following examples the output produced by the database system is marked with S and the part typed by the user is marked with U. The user starts each interaction by typing the command, which is offset to the left above the interaction.

5.3.1. The INSERT Command

```
INSERT INTO SCHEMAS
       S: ENTER VALUES AS THEY ARE REQUESTED.
          RELATION = ?
       U: EMP
       S: NBR_ATTRS = ?     (NUMBER OF COLUMNS IN THE RELATION)
       U: 4
       S: NAME_OF_ATTR_NO_1 = ?
       U: NAME
       S: TYPE_OF_ATTR_NO_1 = ?     (C = CHARACTER, I = INTEGER)
       U: C
       S: ATTR_NO_1_INDEXED = ?     (Y = THIS ATTRIBUTE IS TO HAVE A
                                     SECONDARY INDEX CREATED FOR IT,
                                     N = NOT TO BE INDEXED)
       U: Y
              .
              .
              .

INSERT INTO EMP
       S: ENTER VALUES AS THEY ARE REQUESTED.
          NAME     = ?
       U: SMITH
              .
              .
              .

INSERT INTO USERS
       S: ENTER VALUES AS THEY ARE REQUESTED.
          GROUP_NAME = ?   (USER_ID, IF ADDING A NEW USER TO THE SYSTEM)
       U: GROUP1
       S: USER_ID = ?  (ENTER NULL LINE TO END THE LIST)
       U: TALBOTT
              .
              .
              .

INSERT INTO AUTHS
       S: ENTER VALUES AS THEY ARE REQUESTED.
          GROUP_NAME = ?
       U: GROUP1
              .
              .
              .
```

5.3.2. The DELETE Command

```
DELETE FROM EMP WHERE DEPT = 'D3'
```

```
DELETE FROM EMP WHERE CURRENT
     (Any command with a WHERE clause of CURRENT will operate
     on the last tuple retrieved.
     This allows users to make sure they are deleting the tuple
     they wish to DELETE by entering RETRIEVE commands with
     various WHERE clauses until the appropriate tuple is isolated,
     and then it is deleted with the WHERE CURRENT clause.)
```

```
DELETE FROM USERS
WHERE GROUP_NAME = 'GROUP1' AND USER_ID = 'TALBOTT'
```

```
DELETE FROM USERS WHERE USER_ID = 'TALBOTT'
```

```
DELETE FROM SCHEMAS WHERE REL_NAME = 'EMP'
```

```
DELETE FROM AUTHS
WHERE GROUP_NAME = 'GROUP1' AND OPERATIONS = 'UPDATE' AND RELATION = 'EMP'
AND ATTRIBUTES = 'NAME, DEPT' AND ACCESS CONDITION = 'DEPT IN ('D1','D2','D3')'
```

5.3.3. The RETRIEVE Command

```
RETRIEVE FROM SCHEMAS WHERE RELATION = 'EMP'
        (Omitted attribute list implies all attributes of the relation
         are to be returned.)
```

```
RETRIEVE NAME AND SALARY
FROM EMP
WHERE DEPT = 'D3'
```

```
RETRIEVE FROM USERS WHERE GROUP_NAME = 'GROUP1'
```

```
RETRIEVE AUTHS WHERE GROUP_NAME = 'GROUP1'
     (Prints all authorizations for the group, using a
      combination of the AUTHS and CONDITIONS relations.)
```

```
RETRIEVE UNIQUE NAME FROM EMP
WHERE DEPT = 'D3' ORDER BY SALARY
     (The user must be authorized to RETRIEVE both NAME and SALARY.)
```

5.3.4. The UPDATE Command

```
UPDATE NAME AND DEPT
IN EMP
WHERE DEPT = '45'
       S: ENTER VALUES AS THEY ARE REQUESTED.
          NAME      = ?
       U: SMITH
       S: DEPT      = ?
       U: 10
```

```
UPDATE OPERATIONS IN AUTHS
WHERE GROUP_NAME = 'GROUP1' AND RELATION = 'EMP'
       S: ENTER VALUES AS THEY ARE REQUESTED.
          OPERATIONS = ?
       U: OWN, RETRIEVE, INSERT
```

```
UPDATE ACCT_NO, PROJ_NAME
IN USERS
WHERE GROUP_NAME = 'GROUP1'
       S: ENTER VALUES AS THEY ARE REQUESTED.
          ACCT_NO    = ?
       U: 503
       S: PROJ_NAME  = ?
       U: DESIGN
```

5.3.5. The STORE Command

```
STORE FROM SCHEMAS
       S: ENTER THE RELATION NAME.
       U: EMP
       S: ENTER THE FILE NAME OF THE DATA.
          USE NEW FILE FOR STORE COMMAND; OLD FOR LOAD COMMAND.
       U: EMPSCH.DAT
```

```
STORE FROM EMP
       S: ENTER THE FILE NAME OF THE DATA.
          USE NEW FILE FOR STORE COMMAND; OLD FOR LOAD COMMAND.
       U: EMPLOY.DAT
```

5.3.6. The LOAD Command

```
LOAD INTO SCHEMAS
       S: ENTER THE RELATION NAME.
       U: EMP
       S: ENTER THE FILE NAME OF THE DATA.
          USE NEW FILE FOR STORE COMMAND; OLD FOR LOAD COMMAND.
       U: EMPSCH.DAT

LOAD INTO EMP
       S: ENTER THE FILE NAME OF THE DATA.
          USE NEW FILE FOR STORE COMMAND; OLD FOR LOAD COMMAND.
       U: EMPLOY.DAT
```

5.4. The Interactive Option of the MULTISAFE Language

Many of the commands are at least partly interactive in their basic forms, as shown above. To simplify the use of the command language, a MULTISAFE user can also enter any part of a command followed by a '?'. When the system detects a '?' at the end of an input line, it changes to an interactive prompting mode in which the system displays a list of appropriate keywords and/or other items from which the user makes a selection.

The following examples of the use of '?' to change to the interactive mode at various places within a command are all based on this RETRIEVE command:

```
RETRIEVE NAME AND SALARY FROM EMP WHERE DEPT = 'D3'.
```

The first example uses '?' at the very beginning.

```
U: ?
S: SELECT ONE OF THE FOLLOWING COMMANDS.
   INSERT, DELETE, RETRIEVE, UPDATE, LOAD, STORE, TEST
U: RETRIEVE
S: SELECT ONE OF THE FOLLOWING RELATION TYPES.
```

```
   SCHEMAS, USERS, AUTHS, DATA
U: DATA
S: ENTER THE LIST OF ATTRIBUTES YOU WISH TO SEE.
U: NAME AND SALARY
S: ENTER THE NAME OF THE RELATION.
U: EMP
S: ENTER THE SELECTION PREDICATE
U: DEPT = 'D3'
S: THE COMPLETE COMMAND IS:
   RETRIEVE NAME AND SALARY FROM EMP WHERE DEPT = 'D3'


U: RETRIEVE ?
S: SELECT ONE OF THE FOLLOWING OBJECT TYPES.
   SCHEMAS, USERS, AUTHS, DATA
U: DATA
S: ENTER THE LIST OF ATTRIBUTES YOU WISH TO SEE.
U: NAME AND SALARY
S: ENTER THE NAME OF THE RELATION.
U: EMP
S: ENTER THE SELECTION PREDICATE
U: DEPT = 'D3'
S: THE COMPLETE COMMAND IS:
   RETRIEVE NAME AND SALARY FROM EMP WHERE DEPT = 'D3'


U: RETRIEVE NAME AND SALARY ?
S: ENTER THE NAME OF THE RELATION.
U: EMP
S: ENTER THE SELECTION PREDICATE
U: DEPT = 'D3'
S: THE COMPLETE COMMAND IS:
   RETRIEVE NAME AND SALARY FROM EMP WHERE DEPT = 'D3'


U: RETRIEVE NAME AND SALARY FROM EMP ?
S: ENTER THE SELECTION PREDICATE
U: DEPT = 'D3'
S: THE COMPLETE COMMAND IS:
   RETRIEVE NAME AND SALARY FROM EMP WHERE DEPT = 'D3'
```

5.5. Processing and Enforcement for Commands

5.5.1. Example Policies

This section lists some (more or less arbitrary) choices for protection policy upon which later examples are based. These policies will be referred to by number. Some of these policies have already been mentioned, but are repeated here to emphasize that they are, in fact, policy decisions and not part of the basic mechanisms. Because many of these policies are represented by tuples in AUTHS, they can be changed without redesigning the protection mechanisms.

1) Discretionary authorization is decentralized (implying ownership of data by users), but a SYSADMIN controls certain system resources.

2) Any user can create new data relations. (This policy is included mostly for purposes of illustration and would not be appropriate for all database systems.)

3) Anyone who creates a new data relation is the "owner" of the data contained in that relation, and can share access (and SUBOWNWNership) to it. An owner can grant subownership to another user. A subowner can issue authorizations governing the use of the data, but cannot grant ownership or subownership to others.

4) Only the SYSADMIN can define new users (new accounts).

5) The SYSADMIN will share the right to create user groups with all users. (Again, this is not necessarily always appropriate.)

6) All users may look at all user information, except passwords.

7) Each user may look at all authorization information that governs his/her access. This is a "full disclosure" policy [HARTH81b]. Each authorizer may also see all authorizations which he/she has entered into AUTHS.

8) A partial enforcement policy [HARTH77] will be used. Under such a policy, access is allowed to that subset of the query response which passes all security checking. (In contrast, under a full enforcement

policy, if any part of the query response fails the security checking, no access is allowed to any part of the response.)

9) Revocation (deletion) and modification of an authorization will be limited to the person (AUTHORIZER) who entered the corresponding tuple into AUTHS.

5.5.2. Initial State of System Relations

Since authorization can be highly decentralized in MULTISAFE (and since policies 1 and 3 require it), each user is an authorizer with at least the authority to share access to his/her own data. Although authorization is decentralized, a system administrator, SYSADMIN, can be (and is, per policy 1) included. The SYSADMIN is the owner of the system relations USERS, AUTHS, and SCHEMAS. AUTHS initially contains the rights of the SYSADMIN to INSERT and DELETE relations, users and user groups, and the rights to grant these INSERT and DELETE rights to other authorizers. SYSADMIN has no initial rights to data, but can obtain them as any authorizer can. That is, by creating a relation, a user (possibly the SYSADMIN) becomes its owner and receives full access and granting rights (per policy 3). In addition, a user can receive access and/or granting rights by being granted them from another (owning) authorizer.

Initially the system has only one user, the system administrator (SYSADMIN) and one user group, GENERAL. The GENERAL user group, by definition, contains all users and serves as an efficient repository for that minimal set of rights common to all users.

The following tables show the relations USERS and AUTHS, and the initial tuples in the relations. In all relations an attribute of '*' indicates a 'don't care' attribute; that is this attribute will match any value. Since the GENERAL group below has a '*' value for each of its attributes, it will match every user.

USERS:

	GROUP_NAME	USER_ID	ACCT_NO	TERM_NO	PROJ_NAME	PASS WORD
1	SYSADMIN	SYSADMIN	0	*	SYS	ADMIN
2	GENERAL	*	*	*	*	*

Initially the relation AUTHS contains only three tuples. These tuples give all rights to access the three system relations (USERS,

AUTHS:

	AUTHO-RIZER	GROUP_NAME	OPERATIONS	RELATION	ATTRI-BUTES	ACCESS_CONDITION
1	--	SYSADMIN	OWN,INSERT,...	AUTHS	111111	T
2	--	SYSADMIN	OWN,INSERT,...	USERS	111111	T
3	--	SYSADMIN	OWN,INSERT,...	SCHEMAS	111...	T
4	SYSADMIN	GENERAL	INSERT	USERS	111111	C1
5	SYSADMIN	GENERAL	RETRIEVE	USERS	111110	T
6	SYSADMIN	GENERAL	INSERT,RETRIEVE,...	SCHEMAS	111...	T
7	SYSADMIN	GENERAL	RETRIEVE	AUTHS	111111	C2
8	SYSADMIN	GENERAL	UPDATE,DELETE	AUTHS	111111	C3

CONDITIONS:

COND_NAME	PREDICATE
C1	NEW(GROUP_NAME) > NEW(USER_ID)
C2	(u(q) ∈ AUTHS.GROUP_NAME) OR (u(q) = AUTHS.AUTHORIZER)
C3	u(q) = AUTHS.AUTHORIZER

AUTHS, and SCHEMAS) to the system administrator, SYSADMIN. When the system is first run, the system administrator grants the GENERAL user group some access to the system relations, resulting in the tuples shown below. Tuples 4 and 5 are in discussed in the next section. Policy 2 is implemented by the sixth tuple. Tuple 7 and its access condition implement policy 7. Tuple 8 and condition C3 represent policy 9. The ownership requirement (of policy 3) to INSERT, DELETE, or UPDATE authorizations is not shown in the AUTHS table, but is built into the authorization enforcement mechanism, discussed in section .

5.5.3. User and User Group Definition and Enforcement

Both individual users and user group definitions are added to the relation USERS with the command INSERT INTO USERS. Note that all new users (being members of GENERAL) will automatically receive the GENERAL rights to access the system relations as described in the previous section. For example, tuple 4 of AUTHS, and its access condition, implement policy 5. The access condition predicate, C1, restricts insertions in USERS to groups, as individual user definitions are denoted by having the USER_ID appear also as the value in the GROUP_NAME field. Tuple 5 implements policy 6 by allowing all users to read all user information, except for passwords--this exception enforced by the zero in the final bit position of the ATTRIBUTES column.

To continue the running example, the system administrator next adds three new users to the system.

```
SYSADMIN: INSERT INTO USERS
SYSTEM  : ENTER THE VALUES AS THEY ARE REQUESTED.
```

```
          GROUP_NAME = ? (USER_ID, IF ADDING A NEW USER TO THE SYSTEM)
SYSADMIN: FIKE
SYSTEM  : USER_ID = ?  (ENTER NULL LINE TO END THE LIST)
SYSADMIN: FIKE
SYSTEM  : ACCT_NO = ?
SYSADMIN: 12001
SYSTEM  : TERM_NO = ?
SYSADMIN:
SYSTEM  : PROJ_NAME = ?
SYSADMIN: DESIGN
SYSTEM  : PASSWORD = ?
SYSADMIN: CHET
SYSTEM  : PASSWORD, AGAIN = ?
SYSADMIN: CHET
```

For brevity, the reader may assume that two other similar commands have been issued, creating users Talbott and Lundin. The ability of SYSADMIN to INSERT these new user definitions into USERS is represented by the first tuple in AUTHS (policy 4). Enforcement for this operation is the same as enforcement for access to any data relation--detailed in section .

The new USERS relation is as follows.

USERS:

	GROUP_ NAME	USER_ ID	ACCT_ NO	TERM_ NO	PROJ_ NAME	PASS WORD
1	SYSADMIN	SYSADMIN	0	*	SYS	ADMIN
2	GENERAL	*	*	*	*	*
3	FIKE	FIKE	12001	*	DESIGN	CHET
4	TALBOTT	TALBOTT	12004	*	IMPL	TOM
5	LUNDIN	LUNDIN	12003	42	IMPL	ROB

The contents of USERS are interpreted as predicates in the following way. Each tuple is an AND of simple predicates of the form "attribute-value". If there is more than one tuple with the same GROUP_NAME, the ANDs of those tuples are ORed to make up the predicate that defines that group. A list of USER_IDs that defines a group, then, is actually an OR of several simple predicates of the form "USER_ID = value". In the above example table, the TERM_NO value of 42 for user Lundin means that this individual user can use only one specific terminal to access the system. No other terminal number will match the AND of this tuple and allow a successful log-in (see log-in enforcement in section).

Next, user Fike creates two user groups, one by listing the users which are to be members of the group and the other by stating a group defining predicate. Before the tuples are added to the relation USERS, the relation AUTHS is checked to see if user Fike has the right to INSERT tuples into USERS. Indeed, tuple 4 of AUTHS (per policies 4 and 5) gives all users the right to INSERT tuples into USERS if the GROUP_NAME and USER_ID are not the same.

```
FIKE      : INSERT INTO USERS
SYSTEM    : ENTER VALUES AS THEY ARE REQUESTED.
            GROUP_NAME = ? (USER_ID IF ADDING A NEW USER TO THE SYSTEM)
FIKE      : GROUP1
SYSTEM    : USER_ID = ?   (ENTER NULL LINE TO END THE LIST)
FIKE      : TALBOTT
SYSTEM    : USER_ID = ?   (ENTER NULL LINE TO END THE LIST)
FIKE      : LUNDIN
SYSTEM    : USER_ID = ?   (ENTER NULL LINE TO END THE LIST)
FIKE      :
SYSTEM    : ACCT_NO = ?
FIKE      :
SYSTEM    : TERM_NO = ?
FIKE      :
SYSTEM    : PROJ_NAME = ?
FIKE      :
SYSTEM    : PASSWORD = ?
FIKE      : PASS1
SYSTEM    : PASSWORD, AGAIN = ?
FIKE      : PASS1

FIKE      : INSERT INTO USERS
SYSTEM    : ENTER VALUES AS THEY ARE REQUESTED.
            GROUP_NAME = ? (USER_ID IF ADDING A NEW USER TO THE SYSTEM)
FIKE      : GROUP2
```

```
SYSTEM    : USER_ID = ?    (ENTER NULL LINE TO END THE LIST)
FIKE      :
SYSTEM    : ACCT_NO = ?
FIKE      :
SYSTEM    : TERM_NO = ?
FIKE      :
SYSTEM    : PROJ_NAME = ?
FIKE      : IMPL
SYSTEM    : PASSWORD = ?
FIKE      : PASS2
SYSTEM    : PASSWORD, AGAIN = ?
FIKE      : PASS2
```

The two commands above entered three new tuples into the relation USERS (see below). Since GROUP1 was defined with a list (an OR) of two userids, the first command entered two tuples, 6 and 7, into the relation. The second command entered only one tuple, number 8.

USERS:

	GROUP_ NAME	USER_ ID	ACCT_ NO	TERM_ NO	PROJ_ NAME	PASS WORD
1	SYSADMIN	SYSADMIN	0	*	SYS	ADMIN
2	GENERAL	*	*	*	*	*
3	FIKE	FIKE	120001	*	DESIGN	CHET
4	TALBOTT	TALBOTT	120004	*	IMPL	TOM
5	LUNDIN	LUNDIN	120003	42	IMPL	ROB
6	GROUP1	TALBOTT	*	*	*	PASS1
7	GROUP1	LUNDIN	*	*	*	PASS1
8	GROUP2	*	*	*	IMPL	PASS2

5.5.4. Data Definition

The creation of new relations is the creation of object types (new tuple types), and is accomplished by the insertion of a tuple into SCHE-

MAS. On the other hand, the insertion of a new data tuple in the new relation is the creation of an instance of that new object type. The right to create either the object type or the instance must be explicitly represented in AUTHS. Tuple 6 allows all users the right to create new relations, according to policy 6.

It doesn't make sense to allow operations on some attributes of SCHEMAS, but not on others. For example, one would not want a given user to be able to define a new relation without being able to name its attributes. Therefore, a simple convention was built into the system which forces the bit code of the ATTRIBUTES column of AUTHS to contain all ones, when the RELATION value is SCHEMAS. Similarly, for <u>all</u> relations, it will be a convention that ATTRIBUTES will be all ones for INSERT and DELETE operations. It makes no sense to INSERT or DELETE only part of a tuple.

Next, suppose user Talbott creates a relation, EMP, with 4 attributes, NAME, MGR, SALARY, and DEPT. Enforcement for insertions into SCHEMAS is the same as enforcement for access to any data relation--detailed in section .

```
TALBOTT : INSERT INTO SCHEMAS
SYSTEM  : ENTER VALUES AS THEY ARE REQUESTED.
          RELATION = ?
TALBOTT : EMP
SYSTEM  : NBR_ATTRS = ?    (NUMBER OF COLUMNS IN THE RELATION)
TALBOTT : 4
SYSTEM  : NAME_OF_ATTR_NO_1 = ?
TALBOTT : NAME
SYSTEM  : TYPE_OF_ATTR_NO_1 = ?  (C OR I)
TALBOTT : C
SYSTEM  : ATTR_NO_1_INDEXED = ?  (Y OR N)
TALBOTT : Y
SYSTEM  : NAME_OF_ATTR_NO_2 = ?
TALBOTT : MGR
```

```
SYSTEM  : TYPE_OF_ATTR_NO_2 = ?  (C OR I)
TALBOTT : C
SYSTEM  : ATTR_NO_2_INDEXED = ?  (Y OR N)
TALBOTT : N
SYSTEM  : NAME_OF_ATTR_NO_3 = ?
TALBOTT : SALARY
SYSTEM  : TYPE_OF_ATTR_NO_3 = ?  (C OR I)
TALBOTT : I
SYSTEM  : ATTR_NO_3_INDEXED = ?  (Y OR N)
TALBOTT : N
SYSTEM  : NAME_OF_ATTR_NO_4 = ?
TALBOTT : DEPT
SYSTEM  : TYPE_OF_ATTR_NO_4 = ?  (C OR I)
TALBOTT : C
SYSTEM  : ATTR_NO_4_INDEXED = ?  (Y OR N)
TALBOTT : Y
```

After creating the relation EMP, user TALBOTT inserts 4 tuples into it.

```
TALBOTT : INSERT INTO EMP
SYSTEM  : ENTER VALUES AS THEY ARE REQUESTED.
          NAME = ?
TALBOTT : SMITH,J
SYSTEM  : MGR = ?
TALBOTT :
SYSTEM  : SALARY = ?
TALBOTT : 40000
SYSTEM  : DEPT = ?
TALBOTT : D1
```

For brevity, the commands for the other three tuples are omitted. The data relation EMP becomes as shown below. A '-' in an attribute indi-

EMP:

	NAME	MGR	SALARY	DEPT
1	SMITH,J	-	40000	D1
2	JONES,J	SMITH,J	20000	D1
3	SMITH,S	SMITH,J	20000	D1
4	JONES,S	-	45000	D2

cates that the user did not supply a value for that attribute; ie, it is a 'null' value. The reader should note that the creation of EMP not only inserts a tuple into SCHEMAS, but also adds a tuple (tuple 8 below) into AUTHS, automatically establishing ownership and all access rights to EMP by Talbott (per policy 3).

AUTHS:

	AUTHO- RIZER	GROUP_ NAME	OPERATIONS	RELATION	ATTRI- BUTES	ACCESS CONDITION
1	--	SYSADMIN	OWN,INSERT,...	AUTHS	111111	T
2	--	SYSADMIN	OWN,INSERT,...	USERS	111111	T
3	--	SYSADMIN	OWN,INSERT,...	SCHEMAS	111...	T
4	SYSADMIN	GENERAL	INSERT	USERS	111111	C1
5	SYSADMIN	GENERAL	RETRIEVE	USERS	111110	T
6	SYSADMIN	GENERAL	INSERT,RETRIEVE,...	SCHEMAS	111...	T
7	SYSADMIN	GENERAL	RETRIEVE	AUTHS	111111	C2
8	SYSADMIN	GENERAL	UPDATE,DELETE	AUTHS	111111	C3
9	TALBOTT	TALBOTT	OWN,RETRIEVE,...	EMP	1111	T

5.5.5. Authorization

So far, user groups GROUP1 and GROUP2 have been created, but members of these groups have not been allowed to access any data relations. For the members of a user group to be allowed to access a data relation, the owner must grant rights to the user group. In the following example the user Talbott grants to the members of GROUP1 the right to UPDATE the attributes NAME and SALARY of all tuples of the relation EMP which have the value D1 for the attribute DEPT.

```
TALBOTT : INSERT INTO AUTHS
SYSTEM  : ENTER VALUES AS THEY ARE REQUESTED.
          GROUP_NAME = ?
TALBOTT : GROUP1
SYSTEM  : OPERATIONS = ?
TALBOTT : UPDATE
```

```
SYSTEM  : RELATION = ?
TALBOTT : EMP
SYSTEM  : ATTRIBUTES = ?
TALBOTT : NAME, SALARY
SYSTEM  : ACCESS CONDITION = ?
TALBOTT : DEPT = 'D1'
```

Next, GROUP2 is granted the right to RETRIEVE the names and departments of employees which are in one of the three departments D1, D2, or D3.

```
TALBOTT : INSERT INTO AUTHS
SYSTEM  : ENTER VALUES AS THEY ARE REQUESTED.
          GROUP_NAME = ?
TALBOTT : GROUP2
SYSTEM  : OPERATIONS = ?
TALBOTT : RETRIEVE
SYSTEM  : RELATION = ?
TALBOTT : EMP
SYSTEM  : ATTRIBUTES = ?
TALBOTT : NAME, DEPT
SYSTEM  : ACCESS CONDITION = ?
TALBOTT : DEPT IN ('D1', 'D2', 'D3')
```

The user Lundin is given the right to UPDATE or DELETE the NAMEs of EMPloyees who earn less than 25,000.

```
TALBOTT : INSERT INTO AUTHS
SYSTEM  : ENTER VALUES AS THEY ARE REQUESTED.
          GROUP_NAME = ?
TALBOTT : LUNDIN
SYSTEM  : OPERATIONS = ?
TALBOTT : UPDATE, DELETE
SYSTEM  : RELATION = ?
TALBOTT : EMP
SYSTEM  : ATTRIBUTES = ?
TALBOTT : NAME
SYSTEM  : ACCESS CONDITION = ?
TALBOTT : SALARY < 25000
```

The above authorizations add three tuples to AUTHS, producing the table shown below. All three insertions are allowed because tuple 9 of AUTHS confirms that Talbott is an OWNer of the EMP relation. The details of enforcement of authorization are given in section .

AUTHS:

	AUTHO-RIZER	GROUP_NAME	OPERATIONS	RELATION	ATTRI-BUTES	ACCESS CONDITION
1	--	SYSADMIN	OWN,INSERT,...	AUTHS	111111	T
2	--	SYSADMIN	OWN,INSERT,...	USERS	111111	T
3	--	SYSADMIN	OWN,INSERT,...	SCHEMAS	111...	T
4	SYSADMIN	GENERAL	INSERT	USERS	111111	C1
5	SYSADMIN	GENERAL	RETRIEVE	USERS	111110	T
6	SYSADMIN	GENERAL	INSERT,RETRIEVE,...	SCHEMAS	111...	T
7	SYSADMIN	GENERAL	RETRIEVE	AUTHS	111111	C2
8	SYSADMIN	GENERAL	UPDATE,DELETE	AUTHS	111111	C3
9	TALBOTT	TALBOTT	OWN,RETRIEVE,...	EMP	1111	T
10	TALBOTT	GROUP1	UPDATE	EMP	1010	C4
11	TALBOTT	GROUP2	RETRIEVE	EMP	1001	C5
12	TALBOTT	LUNDIN	UPDATE,DELETE	EMP	1000	C6

CONDITIONS:

COND_NAME	PREDICATE
C1	NEW(GROUP_NAME) > NEW(USER_ID)
C2	(u(q) ∈ AUTHS.GROUP_NAME) OR (u(q) = AUTHS.AUTHORIZER)
C3	u(q) = AUTHS.AUTHORIZER
C4	DEPT = 'D1'
C5	DEPT IN ('D1','D2','D3')
C6	SALARY < 25000

5.5.6. Log-in Enforcement

In order for defined users to make use of the database, they must log into the system. When a user logs in, s/he provides certain information such as userid and password, and possibly account number and project name. The terminal number is determined by system supplied information. These log-in attributes are stored in a temporary relation, L, as shown in the following example:

L:

USER-ID	PASSWORD	TERMINAL
LUNDIN	ROB	17

The enforcement process must decide if the log-in is to be allowed. It does so by formulating this query*:

```
INTO LOG
RETRIEVE ALL (attributes)
FROM USERS
WHERE USERS.USER_ID = L.USER_ID AND
      USERS.GROUP_NAME = L.USER_ID AND
      USERS.TERMINAL = L.TERMINAL AND
      USERS.PASSWORD = L.PASSWORD.
```

Such a tuple (or tuples) corresponds to the user and his/her password. If found, the log-in is allowed. Tuple 5 of USERS is found in this example.

The enforcement process next determines those user groups of which this user is a member, by the following query. The computation of this set of user groups corresponds to step 1 of the enforcement algorithm given in section 2.3.

---

* All names of temporary relations used by the enforcement process are actually qualified by the USER_ID, in order to keep them separate in a multiuser environment.

```
INTO UG
RETRIEVE GROUP_NAME, PASSWORD
FROM USERS
WHERE USERS.USER_ID = LOG.USER_ID AND
      USERS.ACCT_NO = LOG.ACCT_NO AND
      USERS.TERM_NO = LOG.TERM_NO AND
      USERS.PROJ_NAME = LOG.PROJ_NAME.
```

For the example USERS relation of the previous section, user Lundin will match the following group names with the above query: LUNDIN, GROUP1, GROUP2, and GENERAL. (As mentioned earlier, a value of "*" in the relation will match any value in the query predicate.) The user must provide the passwords for each of these groups, except for GENERAL, in order to use the rights of each one. A user can exclude himself/herself from a group for a terminal session by withholding its password.

After log-in, the authorization information pertinent to this user must be gathered together (into the user's "franchise") in order to make access decisions in response to access requests by the user. The creation of the user's franchise is discussed next.

5.5.7. Determining the User Franchise

After log-in, in anticipation of having to make access decisions on user requests, the enforcement process creates a temporary relation containing that authorization information from AUTHS which is pertinent to just this user and his/her user groups. This set of authorization tuples is called the user's <u>franchise</u> and its computation corresponds to step 2 of the enforcement algorithm given in section 2.3.

```
INTO FRANCHISE
RETRIEVE ALL
FROM AUTHS
WHERE AUTHS.GROUP_NAME IS IN UG.GROUP_NAME
```

Given that the user's franchise is non-empty, the requesting user is now legally occupying the system and has access rights to some parts of the database. FRANCHISE defines this user's access rights for this terminal session until logout, or until some change in authorization requires FRANCHISE to be computed, dynamically, again. Lundin's franchise contains tuples 4 through 8 and 10 through 12 of AUTHS, by virtue of his membership in GENERAL, GROUP1, and GROUP2.

5.6. Making an Access Decision

The enforcement process will be described in terms of single variable (single relation) queries. It has been shown [WONGE76] that all queries can be decomposed into a set of single variable queries. In SEQUEL, a nested query is an illustration of a multi-variable query. Assume the existence of a second data relation,

DEPART (DNO, MGR, FLOOR).

DEPART gives, for each department number, the name of its manager and the floor (of the building) on which it is located. A query which requests the names of all people who work on the seventh floor is:

```
SELECT NAME
FROM EMP
WHERE DEPT IS IN
    SELECT DNO
    FROM DEPART
    WHERE FLOOR = 7
```

The inner and outer parts of this nested query can each be treated as a single variable query, even though such a view may not reflect the way in which retrieval is implemented. Under a partial enforcement policy, each part would retrieve only those tuples passing the data dependent security checks for the corresponding relation.

In order to illustrate the enforcement process, suppose that Lundin enters this query into the UAM:

```
RETRIEVE NAME, SALARY
FROM EMP
WHERE SALARY < 13000.
```

Initial parsing determines and stores the query attributes in a temporary relation, Q, as shown here:

<u>Q</u>:	<u>OPERATION</u>	<u>RELATION</u>	<u>ATTRIBUTES</u>	<u>WHERE COND</u>
	RETRIEVE	EMP	1010	SALARY < 13000

The ATTRIBUTES value in Q denotes all attributes of EMP that appear anywhere in the query. In general, this refers to all attributes in RETRIEVE, UPDATE, or WHERE clauses. INSERT and DELETE commands, which affect whole tuples, imply that all attributes of the relation are involved.

Previous sections (5.5.6 and 5.5.7, respectively) have detailed steps 1 and 2 of the enforcement process (section 2.3). Next to be considered are the data subsets described in steps 3 and 4 of section 2.3. The use of a relational data model and the bit-coded ATTRIBUTES attribute simplifies these concepts. The data subsets of step 3 in the enforcement algorithm are those data relation projections defined in RELATION and ATTRIBUTES of FRANCHISE which overlap D(q), the requested data subset. D(q) is composed of Q.RELATION and Q.ATTRIBUTES, taken together. In section 2.3, overlap of the requested data, D(q), and the data defined in an authorization, D(p), was determined by: D(q) <u>XN</u> D(p)

$\neq \{\}$. In the implementation model that overlap is determined by this predicate:

FRANCHISE.RELATION = Q.RELATION AND FRANCHISE.ATTRIBUTES $\oplus$ Q.ATTRIBUTES

where $\oplus$ is the logical OR over the bits of a bit-by-bit AND over the bits of the two ATTRIBUTES values being compared. Specifically, if A and B are bit coded as (a1 a2 ... an) and (b1 b2 ... bn), then A $\oplus$ B = (a1 & b1) v (a2 & b2) v ... v (an & bn). Practically, A$\oplus$ B has a truth value of one ("true") iff A and B have a value of one in common at any of the bit positions (i.e., if A and B have any overlap).

The computation of the <u>franchise</u> (of the user) <u>for the query</u>, F(q), in step (6) of section 2.3, combines the results of steps (2), (4), and (5). In the implementation model, F(q) is computed as follows:

```
INTO FQ
RETRIEVE ALL
FROM FRANCHISE
WHERE Q.OPERATION IS IN FRANCHISE.OPERATION
    AND Q.RELATION = FRANCHISE.RELATION
    AND Q.ATTRIBUTES ⊕ FRANCHISE.ATTRIBUTES
```

The important attributes of FQ are shown below, for the example:

RELATION	ATTRI-BUTES	ACCESS CONDITION
EMP	1010	C4
EMP	1001	C5
EMP	1000	C6

In the relational database implementation, again, this is a very simple operation. In the example, Lundin's request results in having FQ contain tuples 10, 11, and 12 of AUTHS (which, by this point, were in FRANCHISE per section 5.5.7). Formally, in step 6 of the enforcement algorithm, this computation is:

$$F(q) = \{p \in \underline{P} \mid u(q) \in U(p) \ \& \ D(q) \ \underline{XN} \ D(p) \neq \{\} \ \& \ o(q) \in O(p)\}$$

The characteristic function defining this set is a boolean predicate of three ANDed factors. The first, $u(q) \in U(p)$, was taken into account when FREANCHISE was computed from AUTHS (see section 5.5.7). This is computed at log-in rather than at query processing time, because of performance considerations. The second and third factors of $F(q)$ are represented by the WHERE clause in the above query.

$D^*(q)$, the "data reference", and its covering of $D(q)$, the requested data subset, are next to be considered. In the implementation model, $D^*(q)$ is simply the projection of FQ over RELATION and ATTRIBUTES. Since a partial enforcement policy [HARTH77] will be implemented (policy 8), the question of covering can be solved simply by setting

$$D(q) = D(q) \ \underline{XN} \ D^*(q)$$

as in step 7 of the enforcement algorithm (see section 2.3). Within the implementation described here, both $D^*(q)$--all of its tuples--and $D(q)$ refer to the same relation, namely Q.RELATION (which is EMP in the example). Therefore, the above equation is satisfied by projecting the user query response tuples over attributes allowed by $D^*(q)$. The allowed attributes are indicated by the presence of a "one" in the ATTRIBUTES of a $D^*(q)$ tuple. In other words, the set of allowed attributes is denoted by "ones" in an OR taken down each column of bits in the ATTRIBUTES values of FQ. In the example, the ATTRIBUTES values of the $D^*(q)$ tuples are 1010, 1001, and 1000. The bit-wise OR over these values yields 1011, meaning that only NAME, SALARY, and DEPT can be returned to Lundin for this query. Thus, the request, involving only attributes NAME and SALARY, is indeed covered by $D^*(q)$. This step enforces all query dependent conditions (of section 3).

In step (8) of the enforcement algorithm, F(q) is partitioned into equivalence classes based on elements of D*(q). These partitions are groups of tuples in F(q) having the same value for ATTRIBUTES. Each such group has its ACCESS_CONDITIONs ORed together. In this example, each of the three tuples of F(q) is a class by itself.

The resulting predicates for classes are then ANDed together to form the Effective Access Condition (EAC) of step (9), which represents all system dependent and data dependent conditions (of section 3). In the example, EAC = C4 & C5 & C6. The evaluation of the EAC is discussed in section .

5.7. Enforcement of Authorization

Enforcement of all access operations on USERS and SCHEMAS is via authorizations stored in AUTHS, exactly the same as for operations on data relations. In controlling access to itself, AUTHS is used in a slightly different way, involving OWNership. In particular, enforcement for authorization insertion (and update) depends not on Q (the request to make an authorization), but on the <u>contents</u> of the tuple (the authorization itself) to be inserted (or updated). As an example, consider (from section 5.5.5) the request to insert a tuple granting GROUP1 the right to update NAME and SALARY in EMP for records having DEPT = 'D1'. This request appears in Q as follows:

Q:	OPERATION	RELATION	ATTRIBUTES	WHERE COND.
	INSERT	AUTHS	*	*

The important dependency in computing FA, the franchise for an authorization, is on the new (incoming) tuple for AUTHS:

```
INTO FA
RETRIEVE ALL
FROM FRANCHISE
WHERE ('OWN' IS IN FRANCHISE.OPERATION
       OR 'SUBOWN' IS IN FRANCHISE.OPERATION)
       AND NEW(AUTHS.RELATION) = FRANCHISE.RELATION
```

The requesting authorizer can have OWN as an OPERATION in his/her FRANCHISE only by being the creator of the relation in question. He/she can have SUBOWN as an operation by being a subowner (see policy 3). A subowner is one to whom ownership is granted by the creating onwer. In general, the SUBOWN operation for a subowner can have an access condition predicate attached, making subownership a dynamic attribute. Requests to RETRIEVE from AUTHS are dependent on the tuples affected and are primarily governed by tuple 7 of AUTHS (policy 7). DELETE (and UPDATE) operations on AUTHS involve some potentially complex questions about revocation policy [GRIFP76]. The information in the AUTHORIZER column is sufficient to allow condition C3 of the eighth tuple of AUTHS to implement policy 9, a rather simple revocation policy.

Similarly, USERS and SCHEMAS could have a "DEFINER" column for the USER_ID of the person who entered each tuple. This would be useful for supporting policies that allow deletion or modification only by the person who originally made the definitions. The database approach to protection allows such flexibility in responding to a broad range of policies by changing the PDB structure and contents (and sometimes the queries posed against the PDB by the enforcement process), but not changing the basic mechanisms that operate on the PDB.

5.8. The Treatment of Access Conditions as Boolean Functions

Each access condition will actually create up to three tuples in the CONDITIONS relation. One tuple will contain the access condition, as typed in by the authorizer. The second tuple will contain the part of the condition which can be evaluated before data retrieval and the third tuple contains the part which is data dependent. Simple predicates are of the form: <attribute_name> <arithmetic_comparison_operator> <value>. Predicates, simple predicates connected with logical operators, are parsed and stored as coded strings. Attribute names are given numerical codes that refer to the ordinal positions of the attributes (and their relations) in SCHEMAS. Certain system variables (e.g., the time clock) are given special attribute numbers. Arithmetic comparison operators ($=,\neq,<,>,\leq,\geq$) and logical operators (AND, OR) are given numerical codes, too. A value is stored as a variable length item with its length coded as part of the value. When needed, a predicate can be rapidly expanded into a very simple logic tree and evaluated.

6. SPECIAL IMPLEMENTATION PROBLEMS

6.1. VAX Architectural Constraints

The system described in this paper was implemented on a VAX 11/780. A few implementation problems are worthy of discussion here. First of all, present day computer systems do not satisfy the architectural needs of MULTISAFE. The nearest facsimile was attempted within the constraints of the VAX. VMS was the only available operating system at the time; UNIX might have provided a more suitable intermodule communication facility. A future version under UNIX is being considered.

Each MULTISAFE module (UAM, SRM, PSM) is implemented as a separate subprocess and VAX mailboxes, a VMS interprocess communication mechanism, are used for carrying messages among the modules of MULTISAFE. A mailbox is an area of main memory which is accessed as if it were a sequential file. Since some messages can carry large amounts of data to and from the database, the operating system mailbox facility can be overloaded. Therefore, the messages are divided into two parts: a short fixed length message descriptor and a variable length message text. The message descriptor, which is twenty bytes long, contains a message type code, a user identifier (on behalf of whom the message is sent), and a pointer to the corresponding message text. Message descriptors flow from module to module via mailboxes, while message texts are passed in shared files. (In a more suitable system architecture

[TRUER80] message texts can be passed at high speed by direct memory access within a multiprocessor configuration, or only logically passed merely by switching addressing spaces in shared main memory.) Several processes and subprocesses in the VAX can simultaneously access the same shared file. VMS automatically interlocks shared files at the individual record level. Two shared files are needed--one between the UAM and the PSM and one between the SRM and the PSM. A MULTISAFE module sends a message by reading the first record of the shared file, the directory of used records. Text is placed into an available record in the file, and the directory is updated. If the text is longer than 510 bytes, the last two bytes of the record (page) are set as a pointer to the next record of text. When the text has all been placed in the file, a message descriptor is written into the mailbox. Four mailboxes are used, two between the UAM and the PSM, and two between the SRM and the PSM. They are used in pairs because VMS allows a subprocess to read from a mailbox into which it can write. A second mailbox prevents a module from mistaking messages it has sent, for incoming messages. The two incoming mailboxes at the PSM cannot be combined, because a combined box might allow the UAM to read messages sent to the PSM by the SRM.

An important question which occurred during the system design phase was that of how to divide the software and data structures among the three modules. All three modules need a copy of SCHEMAS, the database directory--the UAM for parsing queries, the SRM for storing and retrieving data, and the PSM for doing security checking. All database operations that modify SCHEMAS in the SRM cause the copies in the UAM and PSM to be updated as well.

Whole tuples are returned to the PSM from the SRM so that it can make security checks dependent on non-retrieved parts of retrieved tuples (see section 3). The SRM builds a file of results and passes it to the PSM, where data dependent checking is done on a per-tuple basis.

Throughout this paper, the term "log-in" refers to a command that runs the MULTISAFE system, not a log-in to the VAX computer. At the operating system level, protection privileges are set so that a VAX user not running the MULTISAFE system cannot access any of its files. This eliminates a rather obvious "back door" path that exists in many prototype database systems.

6.2. Use of the MDB in the PSM

Since the PDB is a relational database, the MDB system (which the SRM uses for data storage and retrieval) was adapted for the PSM to access the PDB. As a consequence, structured FORTRAN (FORTRAN 77) was used as a programming language, in order to be compatible with the existing MDB. Queries, generated by the PSM, to the PDB are internal and have no requirement to be in human-readable form. Further, the PDB queries are always the same. Thus, the PDB queries do not actually have the SEQUEL-like form used for expository purposes in this paper. Instead, they are "built-in" to the PSM software.

A few other changes to MDB proved useful for the PDB. Because all attributes of the USERS relation are indexed, it is not necessary for the log-in PDB query of section 5.5.6 to actually retrieve any tuples.

It is enough just to search the B-tree indexes to determine the existence of any tuples that satisfy the query. Also, the "⊕" operation in the PDB query to obtain FQ, the "franchise for the query" (section 5.6), is not provided as part of the MDB operations. Therefore, the "⊕" operation is separately applied to the tuples retrieved from the rest of the query.

Some features of the MDB itself had to be taken into account for use within a secure environment. As a user convenience, the MDB displays the present values of a tuple before that tuple is updated. In cases (probably rare) where a user has UPDATE, but not RETRIEVE, access rights to a relation, updates will have to be made "blind," probably without even an acknowledgement of whether or not any tuples satisfied the WHERE clause of the UPDATE command. An alternative is to use a policy that says the UPDATE privilege subsumes the right to RETRIEVE. For UPDATE, INSERT, and DELETE all enforcement must be done before physical access occurs. Enforcement for RETRIEVE must be done after physical access, but before results are passed to the user.

6.3. Future

Certain features less central to the present emphasis in MULTISAFE are left as possible future enhancements. One example is a multi-user environment, which present new problems in synchronization, locking, backup and recovery. Another example, is system occupancy checking (section 4), at log-in and other times. A system occupancy condition, a predicate that must be satisfied in order for a user to be using the system at all, can be kept in a new column added to the USERS relation. History keeping and auxiliary program invocation [HARTH76b] are also left to the future. As a very interesting near-term modification, the entire user language will be formally defined in BNF and LLPARSE, a compiler-compiler now available on the VAX, will be used to generate parsers for queries, data definitions, and authorizations. This approach will allow a great deal of flexibility in the still-experimental language syntax.

ACKNOWLEDGEMENTS

The section introducing MULTISAFE is adapted from [HARTH81a], and parts of the section on the predicate-based protection model are adapted from [HARTH76a]. The description of this model here simplifies the development in [HARTH76a] somewhat, but reaches basically the same results. Many thanks are due to Earl Balliet for his contributions to the material in this paper. Thanks are also due to Mason C. Deaver, Jr. for his contribution to the example of MULTISAFE protection. Appreciation is expressed to Tom Talbott, who is programming the protection mechanisms on the VAX 11/780 computer, and for much of the material about the authorization commands and the special implementation problems. It is a pleasure to acknowledge the extensive programming work that Rob Lundin did on the Mini Data Base system.

REFERENCES

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, Maxwell, William, and Morgan, Howard, "Selective Capabilites in ASAP--A File Management System," Proc. of the SJCC (1972), 1181-1185.

DEAVM81 Deaver, Mason C., Jr., "Performance Analysis of MULTISAFE Protection Enforcement Processes," Department of Computer Science, Virginia Polytechnic Institute and State University (expected 1981).

GRIFP76 Griffiths, Patricia P., and Wade, Bradford W., "An Authorization Mechanism for a Relational Database System," ACM Trans. on Database Systems 1, 3 (September 1976), 242-255.

HARTH75 Hartson, H. Rex, "Languages for Specifying Protection Requirements in Data Base Systems--A Semantic Model," Ph.D. Dissertation, Dept. of Computer and Information Science, The Ohio State University (August 1975), Research report: OSU-CISRC-TR-75-6.

HARTH76a 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).

HARTH76b Hartson, H. Rex, and Hsiao, David K., "Full Protection Specifications in the Semantic Model for Database Protection Languages," Proc. of the Annual Conf. of the ACM Houston (October, 1976), pp.90-95.

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.

HARTH81a Hartson, H. Rex, "Database Security--System Architectures," to appear in Information Systems.

HARTH81b Hartson, H. Rex, and Earl J. Balliet, "Modeling of MULTISAFE Protection Enforcement Processes with Extended Petri Nets," submitted for publication. Also available as Technical Report CS91005-R, Department of Computer Science, VPI & SU, Blacksburg, VA 24061.

LAMPB71 Lampson, Butler W., "Protection," Proc. Fifth Princeton Symp. on Information Sciences and Systems, Princeton University (March 1971), 437-443; reprinted in ACM SIGOPS Operating Systems Review 8, 1 (January 1974), 18-24.

REISP81 Reisner, Phyllis, "Formal Grammar and Human Factors Design of an Interactive Graphics System," IEEE Transactions on Software Engineering, SE-7, 2 (March 1981), 229-240.

TALBT81 Talbott, Thomas, "Implementation of MULTISAFE in a Relational Database Environment," M.S. Project Report, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 (expected 1981).

TRUER80 Trueblood, Robert P., H. Rex Hartson, and Johannes J. Martin, "MULTISAFE--A Modular Multiprocessing Approach to Secure Database Management," Technical Report CS80008R, Department of Computer Science, VPI & SU, Blacksburg, VA 24061, also submitted for publication.

WONGE76 Wong, Eugene, and Karel Youssefi, "Decomposition--A Strategy for Query Processing," ACM Trans. on Database Systems, 1, 3 (September 1976), 223-241.