Technical Report CS78007-R

A WORKING PAPER ON THE DEVELOPMENT OF MULTIPROCESSOR ARCHITECTURES FOR SUPPORTING SECURE DATABASE MANAGEMENT

by

Robert P. Trueblood and H. Rex Hartson

September 1, 1978

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

ABSTRACT

The MULTISAFE secure database system architecture was introduced in an earlier technical report [TRUER77]. That architecture is summarized briefly here in section 2. This present report is an interim (not complete, some parts changing rapidly) report which indicates progress since the first report, primarily in the area of message communication within the constrained dataflow network. A formal description of inter-module communication messages and message sequences is presented. In section 4 current directions of the work are indicated. The use of Petri nets for modelling and verifying security is introduced, as is the concept of distributed protection for distributed data.

Keywords: Database Management Systems, Computer Architecture, Multiprocessor Systems, Security, Data Protection, Concurrent Processing, Access Control, Authorization, Enforcement, Inter-module Communication, Data Flow Systems, Message Classification, Message Structure, Message Descriptor, Nested Messages, Distributed Systems, Distributed Database, Distributed Protection, Distributed Authorization

CR Categories: 6.22, 4.33, 3.7, 4.32

TABLE OF CONTENTS

1. INTRODUCTION

Two serious problems with database security are:

1. limited ability of mechanisms to respond precisely to a broad range of system conditions has caused protection policies to be highly constrained in range and variety,
2. high performance cost in processing time, storage, and Input/Output (I/O) operations for removing these constraints.

The first problem, the question of precision in adherance to policies, has only recently received attention in database protection literature. In [HARTH75] and [FERNE75] are some of the first attempts to provide mechanisms sensitive to the general system state. To further improve precision, some of this work has been extended to take into consideration previous access events [HARTH76].

The second problem, performance overhead, has also received limited attention in the literature on database protection. However, from the beginning, its importance has been acknowledged.

The seriousness and importance of these problems are amplified by the requirements of increasingly stringent legislation to protect privacy and individual rights [HOFFL77, TURNR76]. Many of these legislated requirements cannot be satisfied without very responsive protection mechanisms [FONGE77]. Furthermore, responsiveness to these requirements may soon be mandatory, regardless of the added cost. A key objective of this work is to introduce an alternative approach that can meet security requirements more precisely and without unacceptable performance penalties. It is a

further objective that, in terms of security, the behavior of the system described in this approach, namely the data-flow message sequences, can be structured well enough to be verified as secure.

2. AN ALTERNATIVE SYSTEM ARCHITECTURE

In a conventional uniprocessor computer system, an increase in complexity of security mechanisms for greater precision and resolution (granularity) can degrade the performance of the system such that it is no longer cost-effective. In that type of system, processing is sequential and the user and system software share a common main storage. The database management system (DBMS) is implemented as a complex application "on top of" the operating system (OS), and security checking is performed by the DBMS and/or the OS. There are several problems with this type of architecture. For example, because of the common main storage, system software, security procedures, and other user data buffers can be read or altered by other software procedures. The database access mechanisms are usually more complex because they are processed through the operating system. The sequential nature of the whole system offers little aid in improving system performance.

A new MULTIprocessor system for supporting Secure Authorization with Full Enforcement (MULTISAFE) for database management is now being developed [TRUE77]. This new architecture provides the next natural step in resolving some of these problems. By combining the concepts of multiprocessing, pipelining, and parallelism into the new system

organization, improvements in system performance and security are possible.

Many of the current architectural approaches to data management have their early roots in the associative memory work by Berra et al. [DEFIC73, BERRB74] and the back-end computer work by Canaday etal. [CANAR74]. This latter work showed that it was economically feasible to divide a DBMS into two components: a host user/applications processor and a back-end DBMS computer. The most significant and well established work in architectural approaches to secure data management has been done in the Data Base Computer project directed by Hsiao at Ohio State University [BAUMR75, HSIAD76]. Other work has been done on database machines--for example, back-end database management systems have been implemented on minicomputers at Kansas State University [MARYF77]--but without significant emphasis on security. Lang, Fernandez, and Summers [LANGT76] have proposed a partitioning of applications object module software into three parts: application module, data interaction module, and data control (protection) module. Downs and Popek [DOWND77] describe a software data management kernel. The following multiprocessor architecture pulls many of these concepts together in a single systems concept.

The new system configuration (described in more detail in [TRUER77]) is based on functionally dividing a DBMS 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 basic idea is to implement each module on one or more processors forming the multiprocessor system called MULTISAFE. These processors can be as small as microprocessors, or they can be as large as maxiprocessors (full sized main

frame processors). In a conventional uniprocessor environment these three modules function sequentially in an interleaved fashion. In MULTISAFE all three modules function in a concurrent fashion. That is, 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.

The basic (or minimal) multiprocessor architecture for MULTISAFE is composed of three separate processors which are connected to three separate primary random access memory blocks. Thus, each functional module has its own processor and primary memory. The system organization follows the multiport-memory organization with private memories [ENSLP77]. A memory is made "private" by connecting only certain processors to it, thereby providing physical separation between the user's memory and the PSM and SRM memories, for example. This separation (or isolation) can significantly improve security because it is physically impossible for a user to access the PSM or the SRM memories. System performance is also enhanced by the concurrent processing.

With this new MULTISAFE approach there are some expected advantages as well as disadvantages. The advantages of the architecture are as follows:

1. better performance--concurrent processing
2. improved security--flexibility of mechanisms
3. improved verifiability--isolation of mechanisms
4. modularity--changeability of software

Performance gains are achieved by overlapping the processing time of the PSM with the UAM and SRM. Greater resolution of enforcement can therefore be had without substantially degrading the performance of the overall system. Security is improved because a separate processor has been dedicated for administering the protection policies. The processing paths through the system are controlled by this

processor. The protection processor operates concurrently with other processors with the capability of interrupting their processing at any point in time. This capability provides for more flexibility in the security mechanisms--particularly, with regard to the time and places where access decisions can be made. In summary the new system architecture:

1. reduces the possibility of "back doors" or sneak paths, by isolating the protection mechanisms and by forcing all accesses of data to occur through a single path;
2. offers safer failure modes, by showing that failures or penetrations of hardware or software outside of the PSM cannot compromise security; and
3. features a relationship between the operating system and the database system which can protect the data against much of the system's own software.

The verifiability of protection mechanisms is also enhanced by the proposed architecture. This enhancement is primarily due to the logical and physical isolation of protection mechanisms from other mechanisms in the system. Not only are protection mechanisms isolated, but the database access mechanisms are isolated from the user. It is easier to verify (prove correctness of) the protection mechanisms because they are not embedded within other mechanisms. In most conventional systems protection is often embedded within the OS and/or within the DBMS. Verifying the correctness of protection in the conventional system often necessitates the proof or verification of the OS or DBMS in addition to the protection mechanisms. In the proposed architecture the protection mechanisms are grouped together as a whole and are separated from other mechanisms. This eliminates the need of verifying or proving the correctness of the other mechanisms not associated with protection.

The idea of modularity is based on the concept of the "top-down approach" used in system design. Not only does this approach provide a technique in design development and implementation, it also aids in maintenance or reliability of the system. The proposed architecture supports high-level modularity. There are two advantages gained by using this modular approach. One advantage is that the system can function with or without the PSM. That is, it is possible to construct a system with only a UAM and SRM for those application not requiring security. A PSM could be added later, if desired. The other advantage is that changes in one module's software have minimal effect on the other two modules. For example, changing a query language has little or no effect on the PSM or the SRM. The modular architecture of MULTISAFE is also ideally suited for a distributed database environment (see section 4.2).

Some disadvantages of the proposed architecture are the additional communication connections and the overhead for process interruption and synchronization. However, with improved security and concurrent processing, the cost of the additional communication and processing overhead seems justifiable. Another disadvantage is that failure in one processor can halt the whole system. However the failure problem is also very evident for uniprocessor systems. Further, improvements in hardware reliability have decreased the probability of failure.

3. INTER-MODULE COMMUNICATION

From the step by step processing flow, in [TRUER77], it has been shown that the modules (or processors) require

direct communication among themselves and that the modules also need a communication path to the system users. MULTISAFE is an event-driven dataflow system. Thus, all processing is initiated and controlled by events occurring within the message flow, including such events as the transmission of data to and from the database. The flow of messages in MULTISAFE is a critical factor for two reasons. First the message flow must be secure. Secondly, it must be efficient and flexible enough to satisfy all interprocessor needs. To gain a better understanding of these communication needs, it is necessary to examine the characteristics of the messages. Three approaches are used to study the characteristics of the messages. These approaches are message structure, message classification, and message sequences. These approaches are discussed in the following sections.

3.1 Message Structure

Communication between the PSM and the SRM is important, especially for data-dependent security checks where communication is required on each data record (or block of tuples) processed. If this communication process is too slow due to operations such as passing and processing lengthy messages, then the performance of both the PSM and the SRM can be degraded. On the other hand, very short messages, such as single bit flags (or indicators), provide limited capabilities. Thus, to serve both system efficiency and protection needs, the messages are divided into two parts: a short, fixed length descriptor and a variable length text.

The message descriptor is composed of three parts:

1. a message classification code
2. a message ID
3. a message text address

The "classification code" identifies the kind of message being processed. This code is discussed in the next section on message classification. The "message ID" contains unforgeable message identification markings which associate the message with a specific user ID, job name, and/or job number. It may also contain a time stamp indicating when the message was initiated. The "text address" identifies the memory address where the message text resides. The message text contains further specific information which can vary with every message. See Figure 1 for an illustration of a message descriptor and text.

The security of the descriptor is ensured by putting it in a "locked box." The descriptor is set up as part of an abstract data type. Its contents are set and checked by protected procedures which are invoked parametrically. No user or user process can directly access message descriptors. The security of the message text is ensured either by PUSHing (depositing the text in the receiver's memory) or by PULLing (retrieving the text from the sender's memory). For example, all texts for the UAM are deposited (PUSHed) in the UAM's memory, because the UAM is not allowed to access the memory of any other modules. On the other hand, all texts for the PSM are retrieved (PULLed) from the sender's memory, because no other modules can write into the PSM's memory. The dataflow of message descriptors, message texts, and the PUSH/PULL concept are illustrated in Figure 2 and Figure 3. Figure 2 shows the dataflow paths for the storage and retrieval of information to/from the database, and Figure 3 shows the dataflow paths for the display and/or change of authorizations by the authorizer. The PUSH/PULL security mechanism is implemented directly in hardware by the private memory structure of the system and is a key mechanism supporting secure inter-module communication. The relationship between the PUSH/PULL mechanism and access decision binding

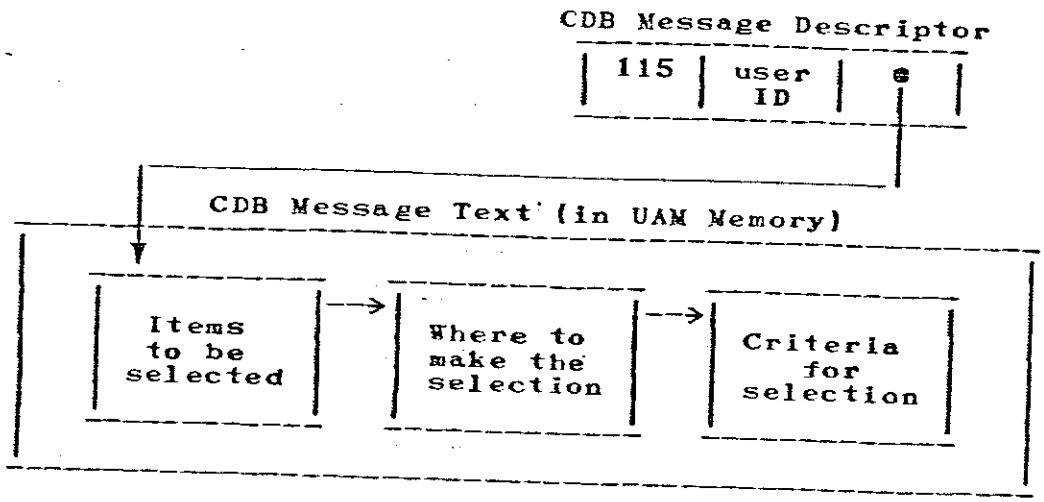
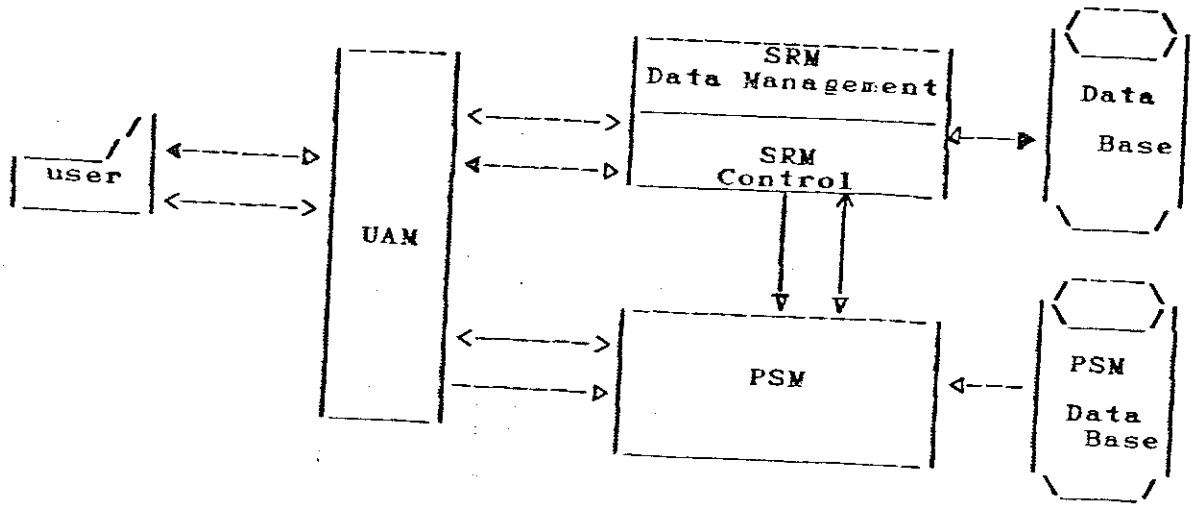
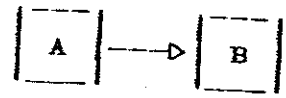


Figure 1. Illustration of the CDB Message.

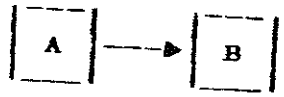


Message text:
---▶ push
---▷ pull

Message descriptor:
---> push



B "pulls" data from A.



A "pushes" data to B.

Figure 2. Dataflow for Storage and Retrieval Operations on the Database.

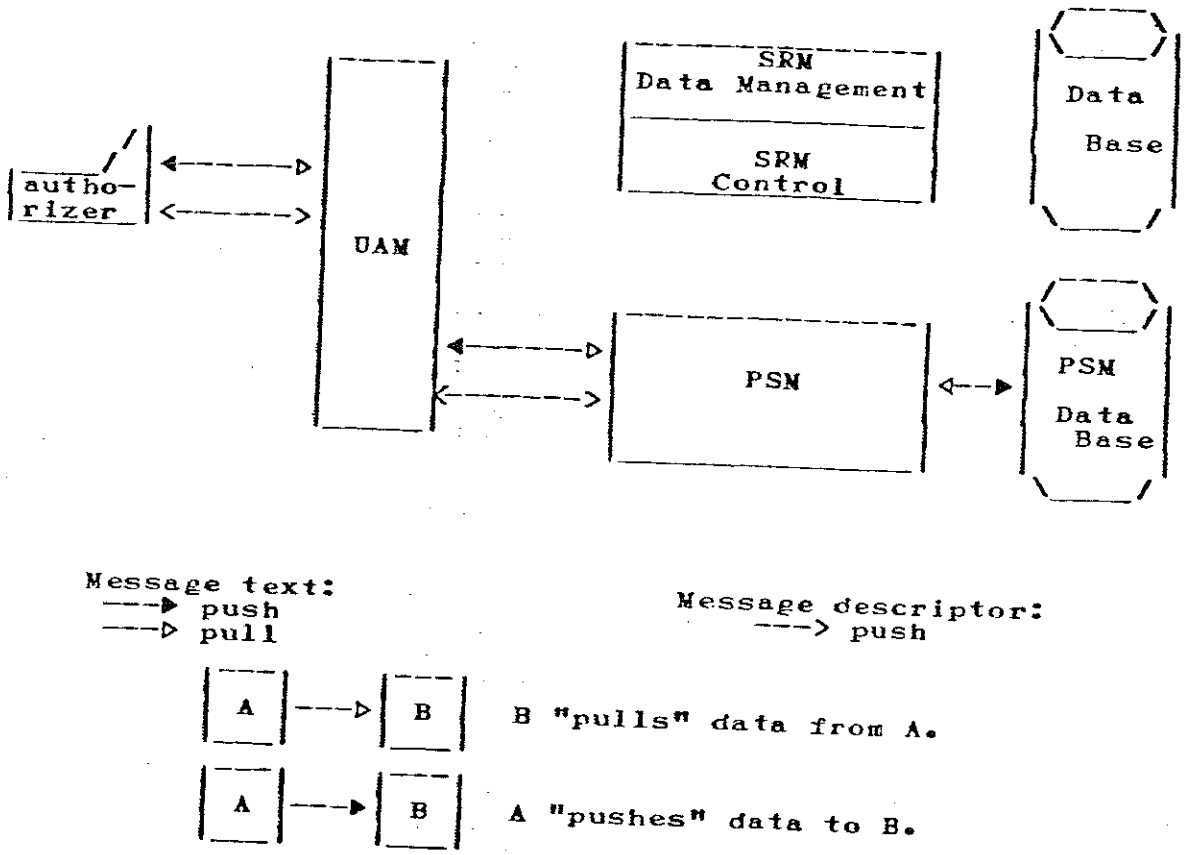


Figure 3. Dataflow for Processing Authorizations.

times is critical. For example, in the UAM, not verified and also more open to inputs from other places in the system, it is possible that the text of a message to the PSM could be altered. However, no such alteration can occur after it has been PULLed into the PSM, since the PSM will be verified and no other module can modify its memory contents. Thus, delaying access checking until the message is safely in the PSM ensures that the checked form of the request is the form in which it finally will be processed by the system, and the door to further tampering is closed.

In a mathematical view each message part is a finite set. These sets are as follows:

MC--the set of message classifications,
ID--the set of valid message ID's,
TEXT--the set of message texts.

The set of all possible messages, M, is represented by the cartesian product of the above three sets:

$$M = MC \times ID \times TEXT.$$

A specific message, m, is represented by the triple:

$$m = (mc, id, text)$$

where, $m \in M$, $mc \in MC$, $id \in ID$, and $text \in TEXT$. Note that each element of M contains its own classification.

3.2 Message Classification and Hierarchy

Messages are the means to communicate between two modules or between users and the system. In order to carry out this communication in a controlled manner, it is necessary to be able to identify every message in the system. The objective of this section is to present a means whereby messages can be classified.

A message is characterized by five attributes. These attributes are:

1. class
2. source
3. target
4. type
5. subtype

Messages are grouped into three classes--request, response, and status. For each message in a class there is a message source, a message target, a message type, and a message subtype. The message source is the entity which generates the message. The message target is the entity which receives the message. Database users, authorizers, UAM, SRM, and PSM can all be source and/or target entities (depending on the other attributes). The message type and subtype identify the functional context of the message. Some of these types are access, authorization, and enforcement. The access type messages are those messages which are related to system accesses such as user login and database accesses. Authorization type messages are those messages which inquire about or modify the authorization information in the PSM. The enforcement type messages are those messages pertaining to enforcement decisions.

The range of each attribute is a finite set of values. These sets are as follows:

CLASS = <request, response, status>

SOURCE = <database user, authorizer, UAM, PSM, SRM>

TARGET = <database user, authorizer, UAM, PSM, SRM>

TYPE = <access, authorization, enforcement, additional-text, dataflow, termination, end-of-data>

SUBTYPE = <login, data, display, change, information, login-decision, data-decision, display-decision, change-decision, decision, CDB, buffer, overall-decision, block-decision>

All possible message classifications (MC) are represented by the cartesian product of the above five sets:

MC = CLASS x SOURCE x TARGET x TYPE x SUBTYPE.

A specific classification is represented by the five-tuple:

$$mc = (c, s, t, p, b)$$

where $c \in$ CLASS, $s \in$ SOURCE, $t \in$ TARGET, $p \in$ TYPE, and $b \in$ SUBTYPE. The collection of all such five-tuples is the set MC.

Not every classification in MC is allowable in MULTISAFE; the secure architecture places constraints on the elements of the set MC. For example, a database user as a source is not allowed to send a message directly to the PSM as a target. There is a non-empty proper subset of MC which will be referred to as the secure message classification (SMC) set.

The five attributes are used to establish a hierarchy of secure message classifications. At the root of the structure is the collection of all secure message classifications. The next five levels are used to represent the five attributes--class, source, target, type, and subtype--respectively. The hierarchical relationship of the attributes is illustrated in Figure 4.

The class attribute partitions the set of message classifications, MC, into three equivalent classes (subtrees). These classes are status, request, and response. The hierarchical structure in Figure 5 illustrates the partitioning.

3.2.1 Status Class

The messages in the status class are basically announcements which require no response from the receiver. The hierarchical structure for the status class is given in Figure 6. This tree structure represents the following three five-tuples:

(status, UAM, PSM, termination, Ø),

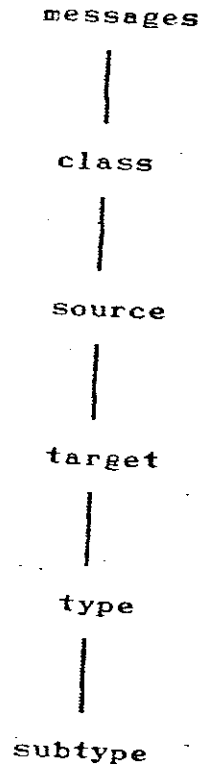


Figure 4. The Hierarchical Relationship of the Message Classification Attributes.

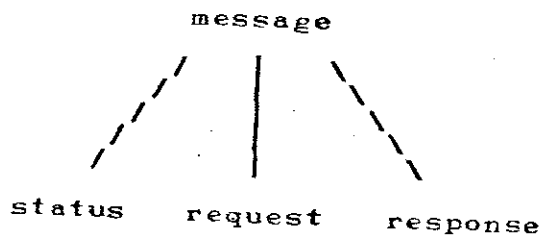


Figure 5. Hierarchical Structure for Partitioning Messages into Classes.

(status, UAM, user, announcement, Ø), and
(status, SRM, PSM, end-of-data, Ø)

where Ø represents "no subtype." The three digit numbers at the leaves of the tree are the classification codes for the messages in the status class. The first digit, 0, represents the status class. The remaining two digits in each code are used to identify each status class message. As an example, the first tuple (code 001) might be used when request processing has been terminated due to user error. An example of the second tuple is an announcement by the UAM to the user about the system's status (such as a hardware or system error). The third tuple might be used by the SRM to inform the PSM that all the tuples for a given request have been retrieved.

3.2.2 Request Class

The request class contains those messages which are basically calls for data or information. These messages require a response from their receivers. The tree structure for classifying messages in the request class is given in Figure 7. This tree structure represents the following twenty-one five-tuples:

(request, user, UAM, access, login),
(request, user, UAM, access, data),
(request, authorizer, UAM, authorization, display),
(request, authorizer, UAM, authorization, change),
(request, UAM, user, enforcement, information),
(request, UAM, user, additional-text, Ø),
(request, UAM, authorizer, enforcement, information),
(request, UAM, authorizer, additional-text, Ø),
(request, UAM, PSM, enforcement, login-decision),
(request, UAM, PSM, enforcement, data-decision),
(request, UAM, PSM, enforcement, display-decision),
(request, UAM, PSM, enforcement, change-decision),
(request, UAM, PSM, authorization, display),
(request, UAM, SRM, access, CDB),
(request, PSM, UAM, enforcement, information),
(request, PSM, SRM, enforcement, information),
(request, SRM, PSM, enforcement, overall-decision),
(request, SRM, PSM, enforcement, block-decision),
(request, SRM, UAM, dataflow, buffer), and
(request, SRM, UAM, dataflow, data)

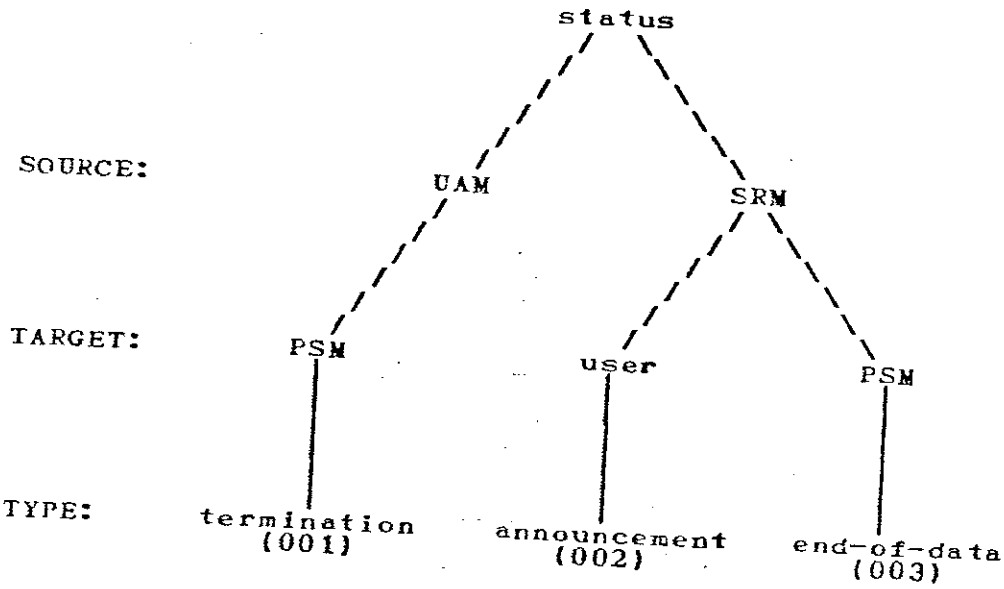


Figure 6. Tree Structure for Messages in the Status Class.

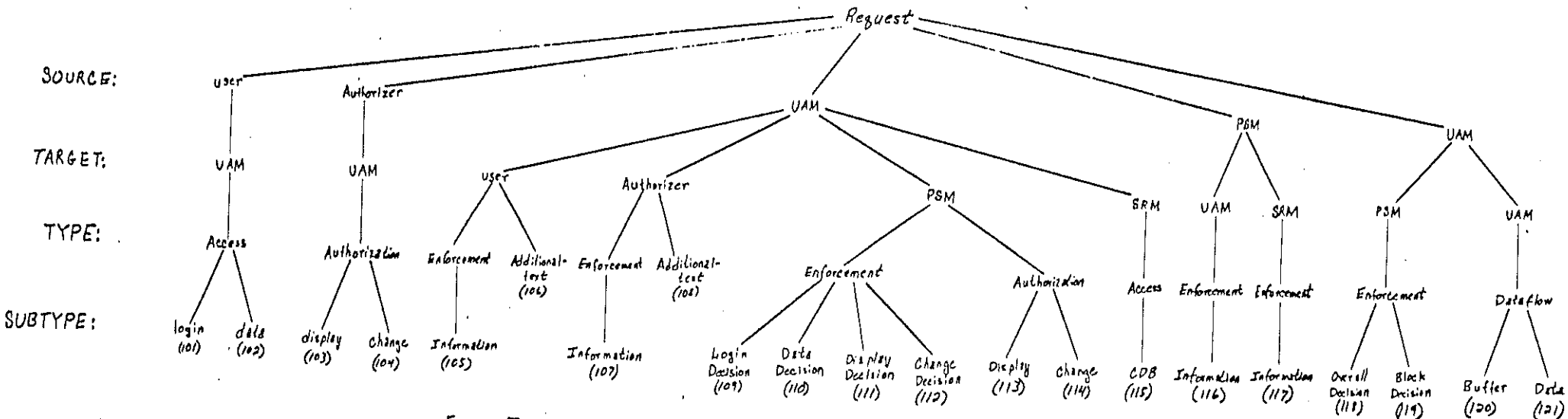


Figure 7. Tree Structure for Classifying Messages in the Request Class.

where Ø represents "no subtype." The meaning for each of the above five-tuples can be found in Table I. The first column of the table gives the SMC number which is used in each message as the "classification code" (see Figure 1). The first digit of the SMC number, which is the digit 1, identifies these messages as belonging to the request class. The last two digits are sequence numbers used to identify each specific request. The SMC number is also given in parentheses at the leaves of the request tree in Figure 7.

3.2.3 Response Class

Those messages in the response class are basically answers to the messages in the request class. The tree structure for classifying messages in the response class is given in Figure 8. This tree structure represents the following twenty-one five-tuples:

(response, user, UAM, enforcement, information),
(response, user, UAM, additional-text, Ø),
(response, authorizer, UAM, enforcement, information),
(response, authorizer, UAM, additional-text, Ø),
(response, UAM, user, access, login),
(response, UAM, user, access, data),
(response, UAM, authorizer, authorization, diplay),
(response, UAM, authorizer, authorization, change),
(response, UAM, PSM, enforcement, information),
(response, UAM, SRM, dataflow, buffer),
(response, PSM, SRM, dataflow, data),
(response, PSM, SRM, enforcement, overall-decision),
(response, PSM, UAM, enforcement, block-decision),
(response, PSM, UAM, authorization, diplay),
(response, PSM, UAM, authorization, change),
(response, PSM, UAM, enforcement, login-decision),
(response, PSM, UAM, enforcement, data-decision),
(response, PSM, UAM, enforcement, display-decision),
(response, PSM, UAM, enforcement, change-decision),
(response, SRM, UAM, access, CDB), and
(response, SRM, PSM, enforcement, information)

where Ø represents "no subtype." The meaning for each of the above five-tuples can be found in Table II. In the first column of the table is the SMC number which is used in each message as the classification code. The first digit of the SMC number, which is a 2, identifies these messages as belonging to the response class. The remaining two digits in the SMC number are sequence numbers used to identify each

TABLE I

REQUEST MESSAGES

SMC NO.	Source	Target	Type	Subtype	Meanings
101	user	UAM	acc	login	user wants to login
102	user	UAM	acc	data	data processing request
103	Aut	UAM	auth	display	display an authorization
104	Aut	UAM	auth	change	change an authorization
105	UAM	user	enf	info	need additional information for enforcement decision
106	UAM	user	Add-text		need more text to complete an earlier request
107	UAM	Aut	enf	info	need information for enforcement decision
108	UAM	Aut	Add-text		need more text to complete an earlier request
109	UAM	PSM	enf	login decis.	overall login check
110	UAM	PSM	enf	data decis.	overall query processing check
111	UAM	PSM	enf	display decis.	overall display check
112	UAM	PSM	enf	change decis.	overall change check
113	UAM	PSM	auth	display	display authorization
114	UAM	PSM	auth	change	change authorization
115	UAM	SRM	acc	CDB	CDB call
116	PSM	UAM	enf	info	need additional information for enforcement decision
117	PSM	SRM	enf	info	need external information for data-dependent checks
118	SRM	PSM	enf	overall decis.	need overall CDB check
119	SRM	PSM	enf	block decis.	need block check
120	SRM	UAM	data flow	buffer	request for an empty buffer
121	SRM	UAM	data flow	data	please receive this buffer full of data

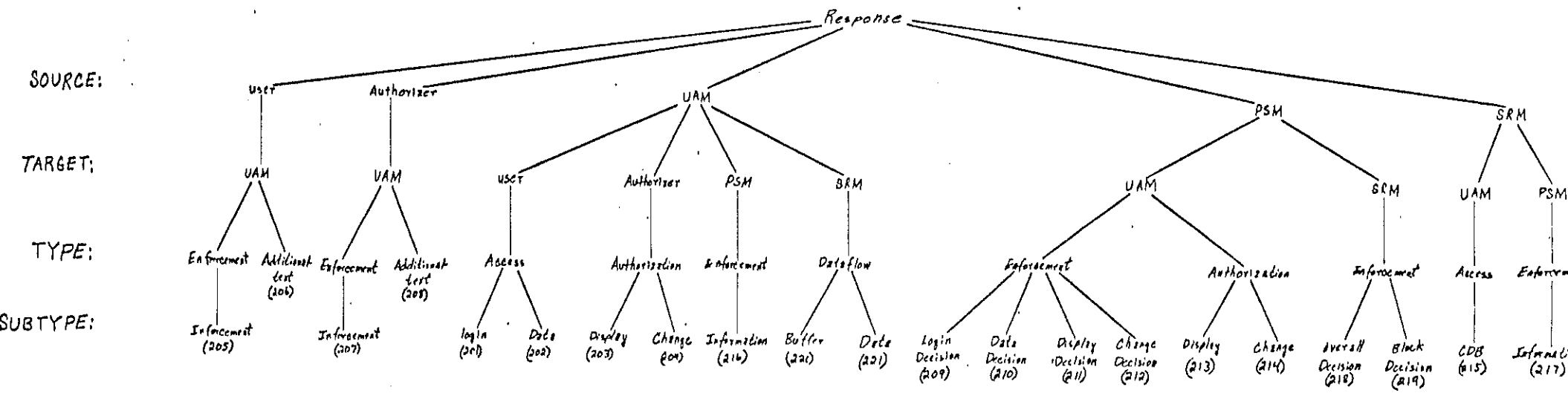


Figure 8. Tree Structure for Classifying Messages in the Response Class.

TABLE II

RESPONSE MESSAGES

SMC NO.	Source	Target	Type	Subtype	Meanings
201	UAM	user	acc	login	login decision
202	UAM	user	acc	data	data processing results
203	UAM	Aut	auth	display	return authorization
204	UAM	aut	auth	change	authorization is changed
205	user	UAM	enf	info	return additional information for enforcement decision
206	user	UAM	add-text		return additional information to complete earlier request
207	Aut	UAM	enf	info	return needed enforcement information
208	Aut	UAM	add-text		return additional information to complete earlier request
209	PSM	UAM	enf	login decis.	security decision on login request
210	PSM	UAM	enf	data decis.	security decision on data access request
211	PSM	UAM	enf	display decis.	security decision on display authorization request
212	PSM	UAM	enf	change decis.	security decision on change authorization request
213	PSM	UAM	auth	display	return authorization
214	PSM	UAM	auth	change	authorization is changed
215	SRM	UAM	acc	CDB	end-of-CDB
216	UAM	PSM	enf	info	return additional information for enforcement decision
217	SRM	PSM	enf	info	return external information for data-dependent checks
218	PSM	SRM	enf	overall decis.	security decision on CDB
219	PSM	SRM	enf	block decis.	security decision on a block
220	UAM	SRM	data flow	buffer	empty buffer is ready
221	UAM	SRM	data flow	data	received full buffer

specific response. The SMC number is also given in parentheses at the leaves of the response tree in Figure 8.

The request and response trees are very similar. The main difference between the two trees is that, in most cases, the target of a given request has become the source of the corresponding response and vice versa. This relationship implies that for every request in the SMC set there is a response in the SMC set. The last two digits of a response SMC number match those of the corresponding request SMC number. The relationship between a request and its response is discussed further in the section on message sequences.

There are some other characteristics about the tree structure. Not all source/target pairs are represented by this structure, and likewise these pairs do not appear in the set SMC. For example, users and authorizers cannot communicate directly with the PSM or SRM, but they can with the UAM. Also, messages of the access type cannot go to or from the PSM.

3.2.4 Classification Language

A message classification language and the grammar for the language are developed in this section. The grammar for generating a single message classification is discussed. The main emphasis of this section is on the grammar rules. These grammar rules must follow the tree structure used for classifying messages, and the grammar rules must also follow the constraints of the MULTISAFE architecture. In short, the grammar must generate only legal, secure message classifications; that is, elements of the set SMC.

The syntax rules (given in Appendix A) for generating a single message classification form a regular grammar. These rules generate the five-tuples which belong to the set SMC. To illustrate the generation process consider the following five-tuple:

(request, PSM, UAM, enforcement, information)

This five-tuple is generated by applying syntax rules 1, 3, 10, 26, and 47 of Appendix A. A derivation tree is given in Figure 9.

The grammar rules also preserve the structure of the message classification trees (see Figures 5, 6, 7, and 8). The preservation of the classification tree is accomplished by representing a node's value and its siblings in each grammar rule. Multiple siblings are represented by alternatives (the "exclusive or" of choices). As an illustration of how the grammar preserves the tree structure, consider the fifth syntax rule of Appendix A:

<UAM-attribute> ::= PSM <UAM-PSM-type>
| user <UAM-user-type>

The rule identifies two subtrees--"PSM <UAM-PSM-type>" and "user <UAM-user-type>." The roots of the subtrees are "PSM" and "user," respectively. The siblings for these two subtrees are "<UAM-PSM-type>" and "<UAM-user-type>."

3.4 Message Sequences

MULTISAFE is a modular dataflow system; communication paths must exist to carry information (messages) from one module to another. To control the flow of messages it is

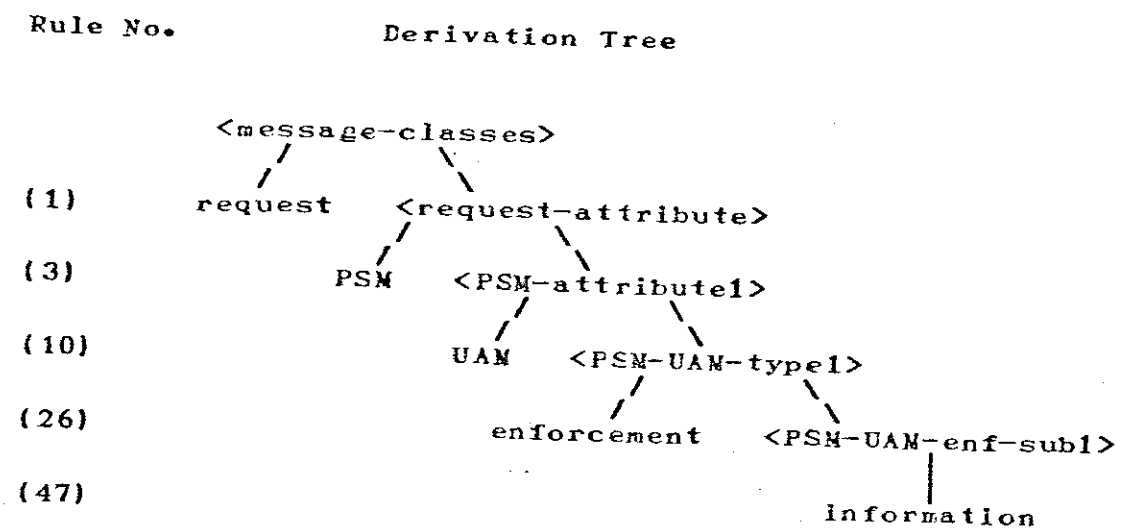


Figure 9. Derivation Tree for the Five-Tuple (request, PSM, UAM, enforcement, information).

necessary to identify the message flow patterns. These patterns, or sequences, may be as simple as a request followed by its respective response, or the patterns may be a complex nesting or embedding of request/response (and other) combinations. A related combination of messages is referred to a message sequence. It is the identification of secure sequences which is of importance in this section. This is done by examining the components of each message and the relationships among various messages within a sequence.

3.4.1 Definition and Identification

Let MC(m), ID(m), and TEXT(m) be defined as selection operators that extract the MC, ID, and TEXT portions, respectively, from a given message, m. Further, let C(mc), S(mc), T(mc), P(mc), and B(mc) be selection operators that extract the CLASS, SOURCE, TARGET, TYPE, and SUBTYPE portions, respectively, from a given message classification, mc.

A linear message sequence, lms, is a sequence of messages, m[1], m[2], ..., m[n], defined by the following axioms:

1. $m[i] \in M$, for all $i = 1, 2, \ldots, n$; and
2. $T(MC(m[i])) = S(MC(m[i+1]))$, for all $i = 1, 2, \ldots, n-1$.

A parallel message sequence, pms, is formed when two linear message sequences, m1[1], m1[2], ..., m1[m] and m2[1], m2[2], ..., m2[n] are combined by the following axioms:

1. $m1[1] = m2[1]$ and $m1[m] = m2[n]$; and
2. $m1[i] \neq m2[j]$, for all $i = 2, 3, \ldots, m-1$ and $j = 2, 3, \ldots, n-1$.

The above definitions allow for the possibility of an lms containing another lms. An lms m1[1], m1[2], ..., m1[n] will be said to contain another lms m2[1], m2[2], ..., m2[j] if:

1. $j \leq n$
2. m2[1] = m1[i], m2[2] = m1[i+1], ..., m2[j] = m1[i+j-1], for some i, $1 \leq i \leq n-j+1$.

A contained lms is sometimes referred to as a subsequence of the containing sequence. The first lms is said to properly contain the second lms if the first condition above is: $j<n$. Most of the time the term "containment" will be used to refer to either of these cases. An lms is said to contain a pms if it contains an lms which is part of a pms. In general, a message sequence, ms, is a linear message sequence which may or may not contain parallel message sequences. The collection of all message sequences forms the set MS.

A secure linear message sequence, slms, is a linear message sequence, m[1], m[2], ..., m[n], defined by the following axioms:

1. m[i] ϵ SMC for all i = 1, 2, ..., n
2. ID(m[i]) = ID(m[i+1]) for all i = 1, 2, ..., n-1

A secure parallel message sequence, spms, is a parallel message sequence, m1[1], m2[2], ..., m1[m] and m2[1], m2[2], ..., m2[n], defined by the following axioms:

1. m1[i] ϵ SMC for all i = 1, 2, ..., m
2. m2[j] ϵ SMC for all j = 1, 2, ..., n
3. ID(m1[i]) = ID(m1[i+1]) for all i = 1, 2, ..., m-1
4. ID(m2[j]) = ID(m2[j+1]) for all j = 1, 2, ..., n-1

A secure message sequence, smc, is a secure linear message sequence which may or may not contain secure paral-

lel message sequences. The collection of all secure message sequences form the set SMS.

The elements of SMS can be partitioned into two equivalence classes. The partition is based on the extrinsic events that initiate the generation of a secure message sequence. Figure 7 illustrates that there are only two extrinsic sources that can initiate a message sequence--the user and the authorizer. Thus, there are two groups of message sequences--those message sequences initiated by a user and those message sequences initiated by an authorizer. Figure 7 goes further to show that a user has two subtypes of access request messages: login and data requests. Likewise, an authorizer has two subtypes of authorization request messages: display and change requests. This means that the user initiated message sequences can be partitioned into two groups--the login subtype and the data subtype. Also, the authorizer initiated message sequences can be partitioned into two groups--the display subtype and the change subtype. In summary, the set SMS is partitioned into four groups for identification--login, data, display, and change subtype.

Messages of each subtype from either source are subject to two kinds of security checking:

1. checking specific to the request
2. system occupancy checking

The specific check for a login request involves user identification, possibly augmented by password, authentication dialogues, etc. The specific check for a data request employs data-independent and data-dependent access conditions. 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 provides an (optional) additional binding time for these conditions.

Within a secure message sequence, regardless of subtype, every request message has a response. A request and its response form a request/response pair (or pair). Formally, a pair of messages, (m[i], m[j]), i≠j, form a request/response pair if and only if

1. C(MC(m[i])) = request and
   C(MC(m[j])) = response,
2. S(MC(m[i])) = T(MC(m[j])),
3. T(MC(m[i])) = S(MC(m[j])),
4. P(MC(m[i])) = P(MC(m[j])),
5. B(MC(m[i])) = B(MC(m[j])), and
6. ID(m[i]) = ID(m[j]).

It is possible for request/response pairs to be nested within other pairs. For example, a user's request to login may contain another pair of messages such as the request/response for a password before a login response is given. Mathematically, the message pair (m[k],m[l]) is nested in the pair (m[i],m[j]) if and only if m[i], m[k], m[l], and m[j] is a secure message sequence. In terms of the source and target attributes, this means that

T(MC(m[i])) = S(MC(m[k])),
T(MC(m[k])) = S(MC(m[l])),
T(MC(m[l])) = S(MC(m[j])),
T(MC(m[j])) = S(MC(m[i])), and
T(MC(m[l])) = S(MC(m[k])).

Even though nesting of message pairs is a key building block used in the construction of secure message sequences, not every message sequence is a perfect nesting of pairs. That is, message sequences can have non-nested adjacent pairs. Also, the slms's of a spms which represent the concurrency of multiprocessing are not nested. (More examples and explanation are given in the next section.)

The request/response pairs for the secure message sequences in the login subtype are given in Figure 10, and Figure 11 illustrates the request/response pairs for message sequences in the data access subtype. Figures 12 and 13 illustrate the request/response pairs for message sequences within the display subtype and change subtype. The arcs in the figures represent the messages. The direction of the arcs illustrate the flow of messages from source to target. The numbers on each arc correspond to the SMC number as given in Tables I and II. Request/response pairs are identified by the commonality of the last two digits of their SMC numbers. The nesting of pairs is illustrated by the nested arcs. For example, in Figure 10 the pair (105, 205) is nested within the pair (101, 201). Such a nesting represents a request to login which require a request/response for a password before the login response is made.

3.4.2 Examples of Secure Message Sequences

Presented below are some examples that illustrate the flow of messages through MULTISAFE. Accompanying these examples are figures that show the message direction, message type, and nested messages. Message direction is depicted by an arrow with its tail at the source and its head at the target. Each arrow is identified by the message SMC number (see Tables I and II). Message nesting is depicted by loops. Some of these loops are formed with a dashed line that connects the request message with its corresponding response message when other pairs are embedded. First, examples of message sequences initiated by having the user as a source are presented. It is then followed by some examples which have the authorizer as the initiating source.

The first example in Figure 14 illustrates a login message sequence. Using the SMC numbers, the order of the messages in this message sequence is as follows:

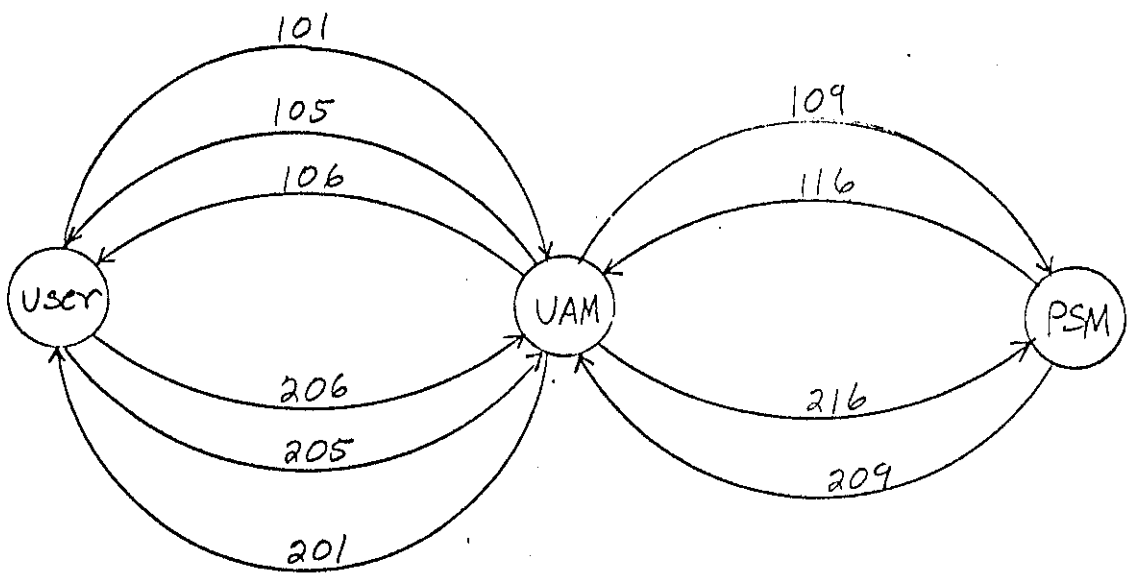


Figure 10. Nesting of Request/Response Pairs within the Login Subtype.

101, 109, 116, 105, 205, 216, 116, 105, 205, 106, 206, 216, 209, 201

Such a message sequence represents an overall login check. This overall login check has two nested subsequences. The first subsequence is a request and a response for a password. In this example, the first password was incorrect and the PSM had to make a second request for the password. However, this time the user did not supply the UAM with sufficient message text; thus, causing the additional text request/response pair (106, 206) to occur. Once the PSM has the correct password, the overall login check is completed (at the point on the PSM line between the adjacent 216 and 209 messages) and the user is allowed on the system. The system occupancy check is also made at this time (between 216 and 209).

If, however, the second password was also incorrect and the user was not allowed on the system, the message sequence in Figure 14 would not have changed. Only the message text for response 209 and 201 and the user status in the PSM would be changed. (The number of times a user can attempt to enter the correct password is within the policies of the authorizer and is not limited by MULTISAFE, per se.)

Figure 15 illustrates the message sequence for query processing in which three blocks of data are retrieved and passed to the user. The message sequence begins with a user entering his query (message 102). Upon receiving the query the UAM notifies the PSM (110) so that overall query processing checking can begin. After the query has been analyzed syntactically, the UAM does a CDB to the SRM (115) for data. The SRM then tells the PSM (118) that overall CDB checking is needed. The SRM retrieves the first block of data from the database and gives it to the PSM for data-de-

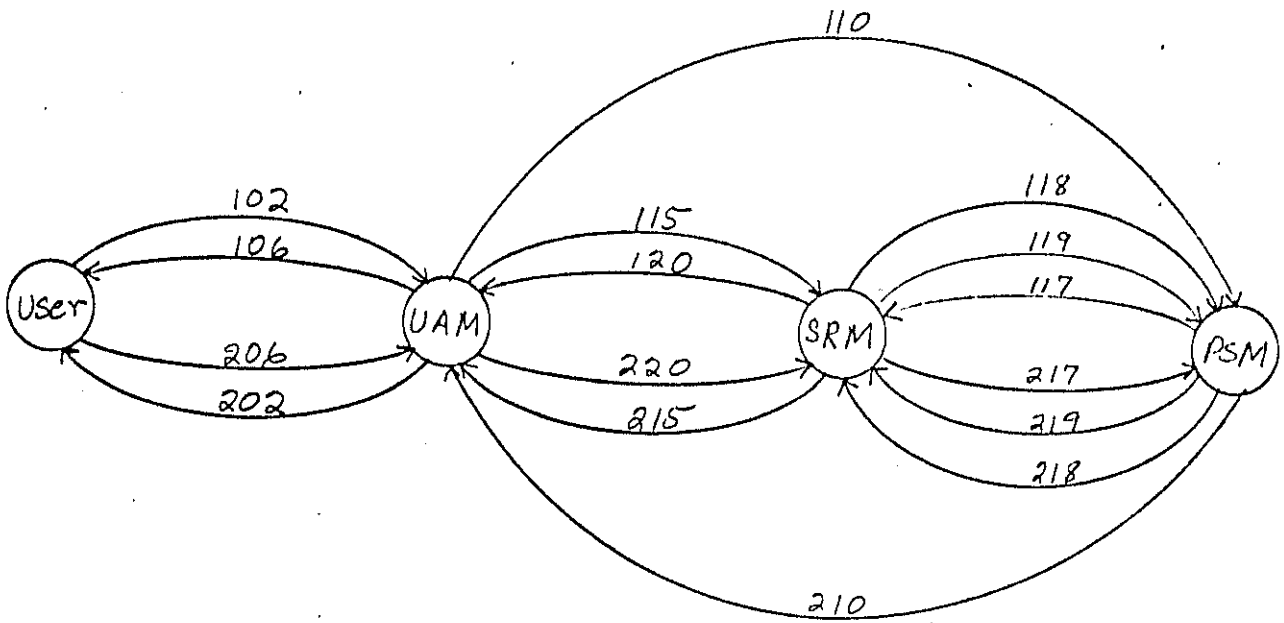


Figure 11. Nesting of Request/Response Pairs within the Data Access Subtype.

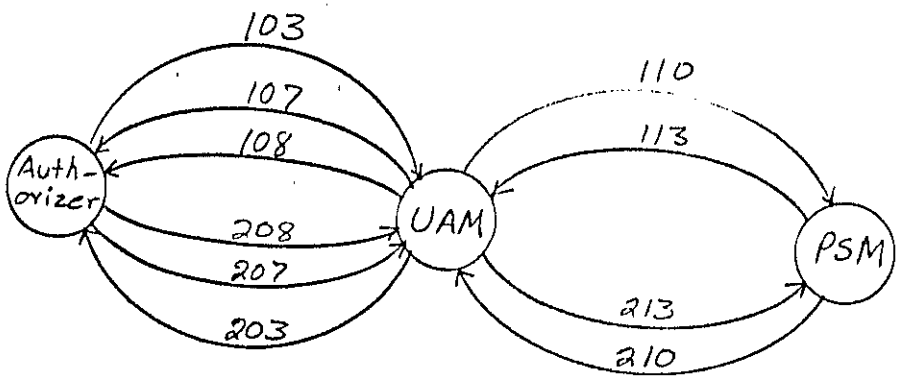


Figure 12. Nesting of Request/Response Pairs within the Display Subtype.

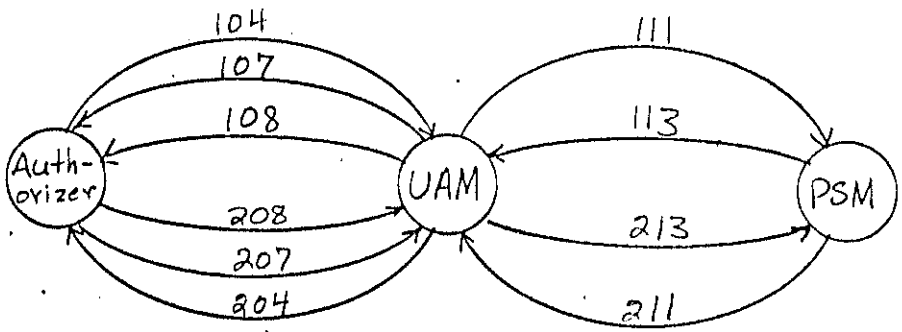


Figure 13. Nesting of Request/Response Pairs within the Change Subtype.

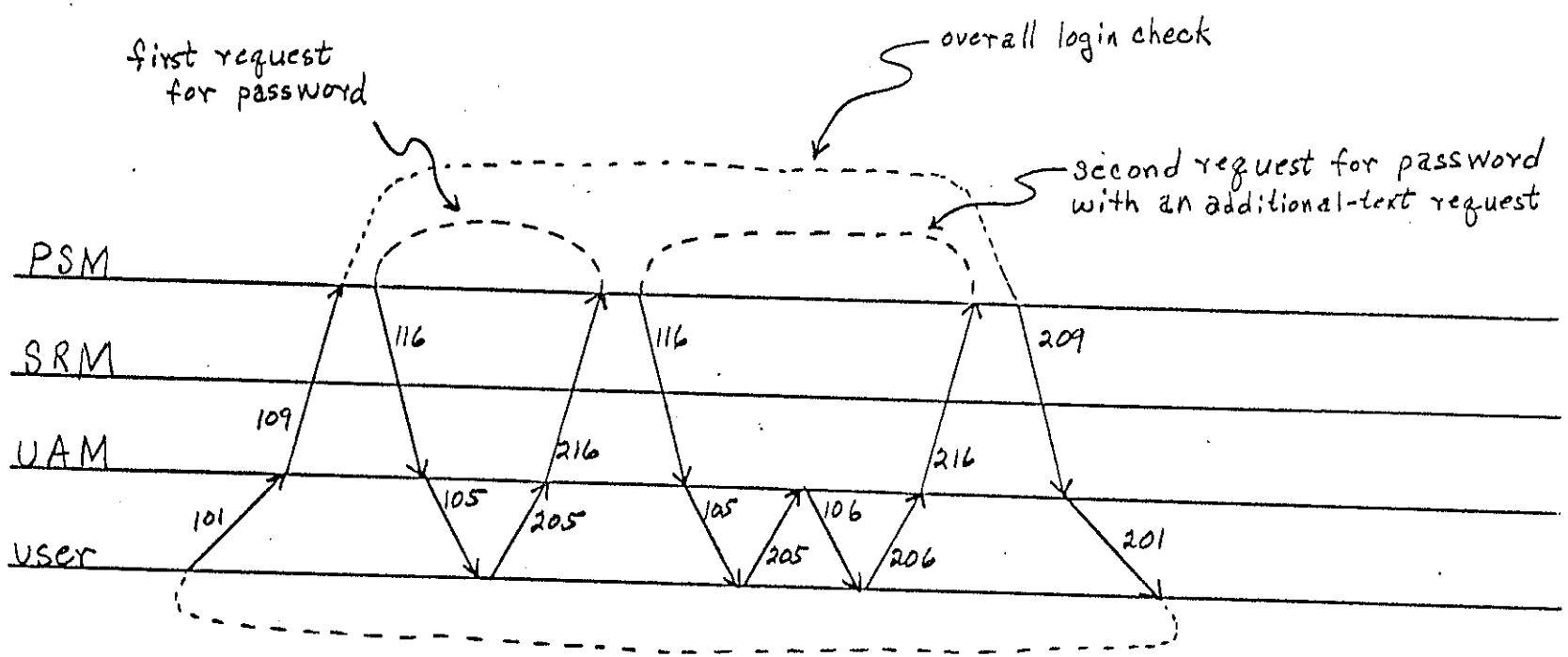


Figure 14. An Example of a Message Sequence for the Login Subtype.

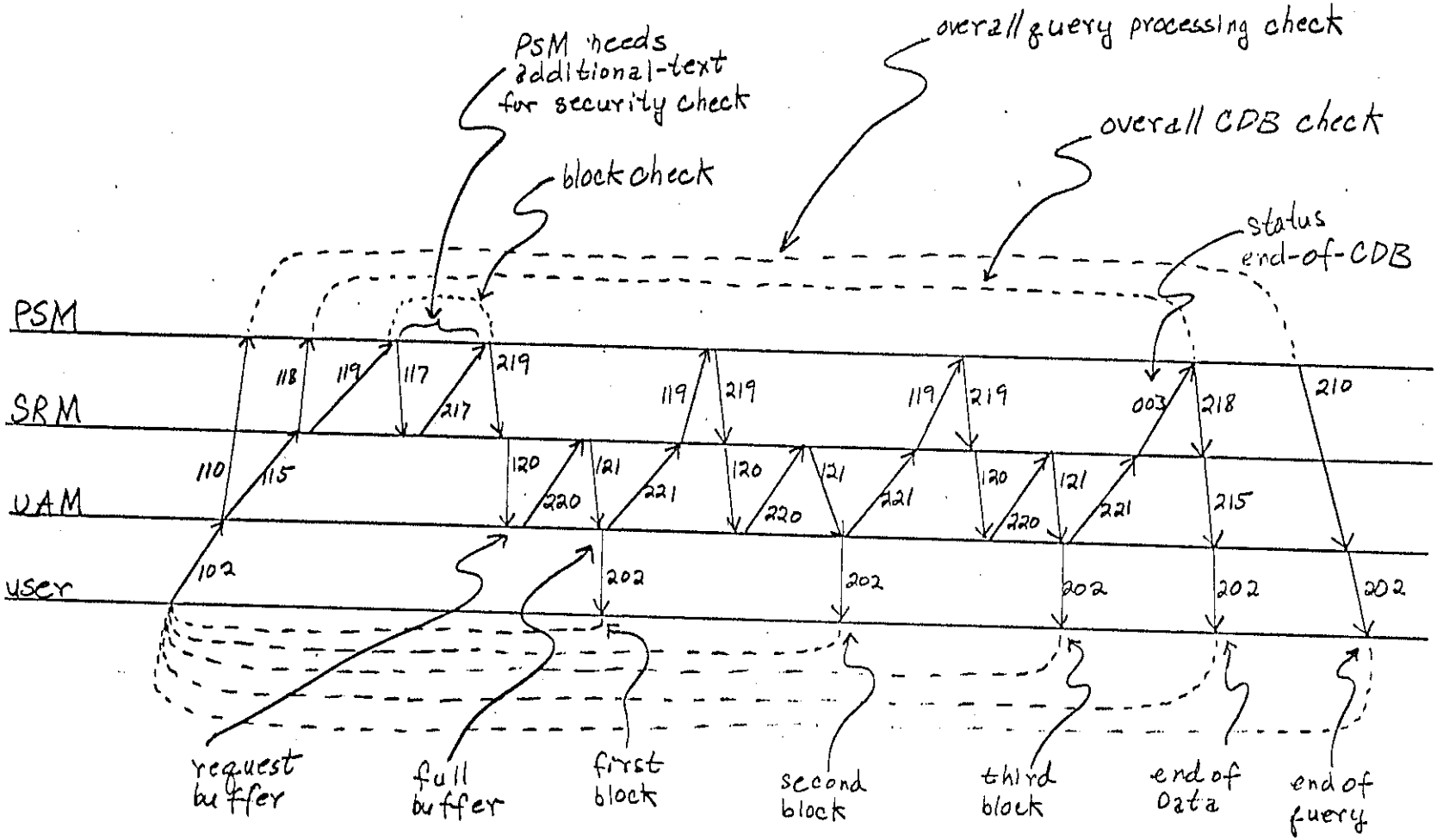


Figure 15. An Example of a Message Sequence for Query Processing in which Three Blocks are Processed.

pendent checking (119). For this particular query the PSM needs additional information from the database before data-dependent checking is completed. The message pair (117, 217) furnishes the PSM with the needed information. The PSM completes its security checking and tells the SRM (219) to give this block of data to the UAM. The SRM requests (120) and receives (220) a buffer area in the UAM memory. Upon receiving the first block of data from the SRM (121) and acknowledging its acceptance (221), the UAM gives the information to the user (202). This process is repeated twice for the second and third blocks of data. When the CDB request has been satisfied, the PSM is notified by an end-of-CDB message (003). This message causes the PSM to finalize the overall CDB checking (218) and the overall query processing checking (210). The SRM also informs the UAM (215) that the CDB has ended, and finally the user is notified (202).

Sometimes blocks of data are not authorized. This results in only certain blocks of data being passed on to the UAM and the user. Such a situation is illustrated in Figure 16. In this illustration, only the second block of data receives security clearance and is passed on to the UAM and the user. The first, third, and fourth blocks of data fail security clearance and thus these blocks venture no further than the SRM or PSM.

It is possible for data access attempts to be denied early in the query process. Figure 17 illustrates the message sequence where a data access attempt is denied before the database is accessed.

As an example, consider the case mentioned earlier in which the user is allowed to occupy the system only between

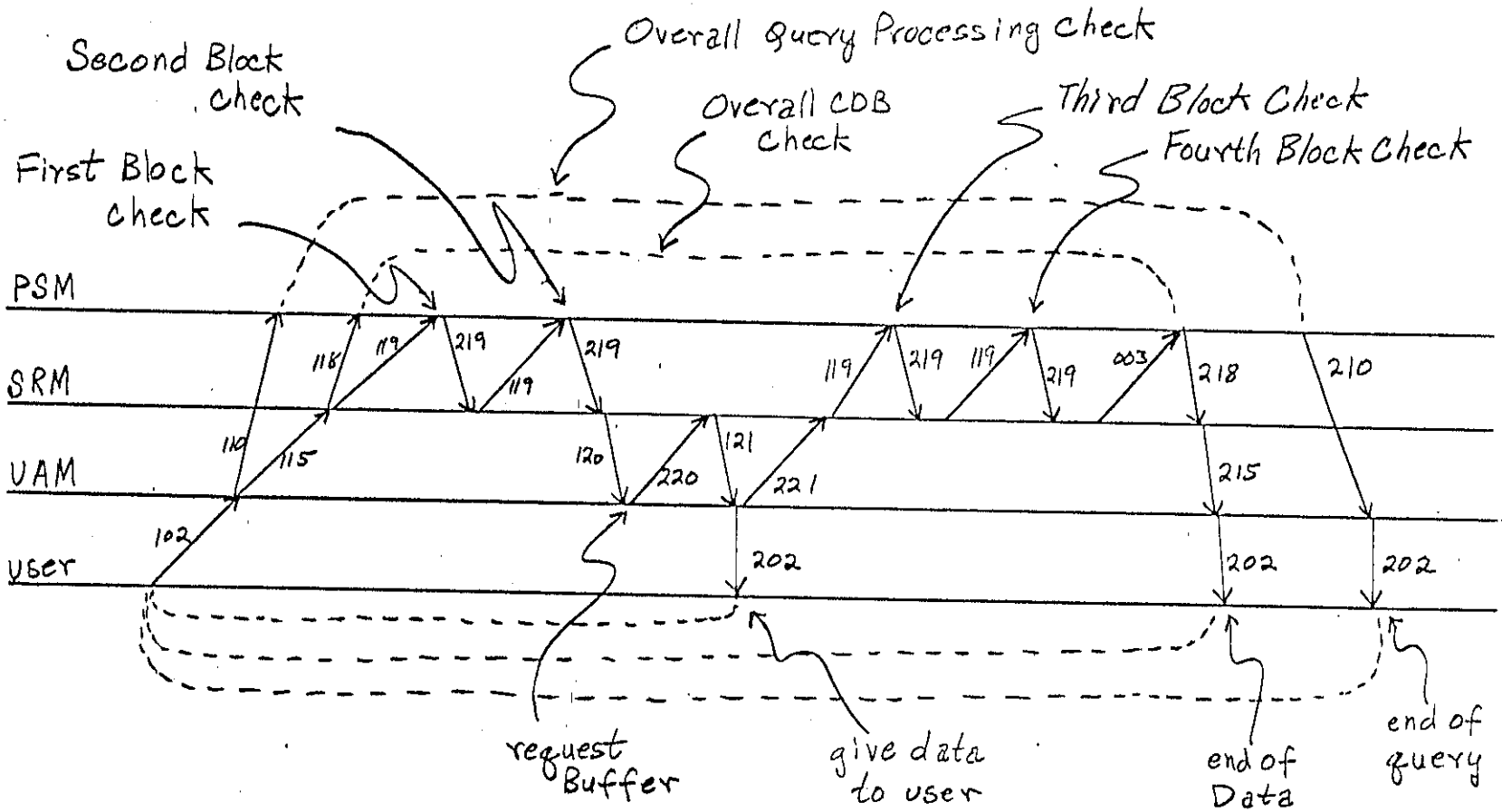


Figure 16. An Example of a Message Sequence for Retrieving Four Blocks with only the Second Block Passed to the User.

8:00 a.m. and 5:00 p.m. A potential penetrator logs in just before 5:00 p.m. in hopes of returning later, when the office is empty, to print a copy of an "eyes-only" file--an action which could readily be detected by co-workers during normal working hours. The plan is defeated by the system occupancy check which re-binds the login condition (just before the final 210 on the PSM line) at each request.

Figure 18 illustrates the situation where data access is denied after a block of data has been retrieved from the database. The latter situation can occur because the SRM and PSM operate concurrently. That is, a data access attempt is denied after the SRM had already begun data retrieval concurrently with the PSM checking. It should be well noted that concurrent processing by all three modules (UAM, SRM, and PSM) can and does occur even though the above figures of message sequences do not show this concurrency.

Illustrated in Figure 19 is a message sequence for processing a display request for some authorization. The authorizer makes a request to the UAM (103) for an authorization to be displayed. The UAM passes the request on to the PSM (110). In the process of evaluating and checking the request, the PSM needs some authentication information from the authorizer. In this example, the authentication process results in two subsequences. The first subsequence (113, 107, 207, 213) provides insufficient information to the PSM which leads to the second subsequence (113, 107, 207, 108, 208, 213). In the second subsequence the authorizer fails to return sufficient text to the UAM which causes the additional text pair (108, 208) to occur. Once the PSM has the needed information the authentication process is completed along with any system occupancy checks (between the adjacent 213 and the 210 messages on the PSM line). If

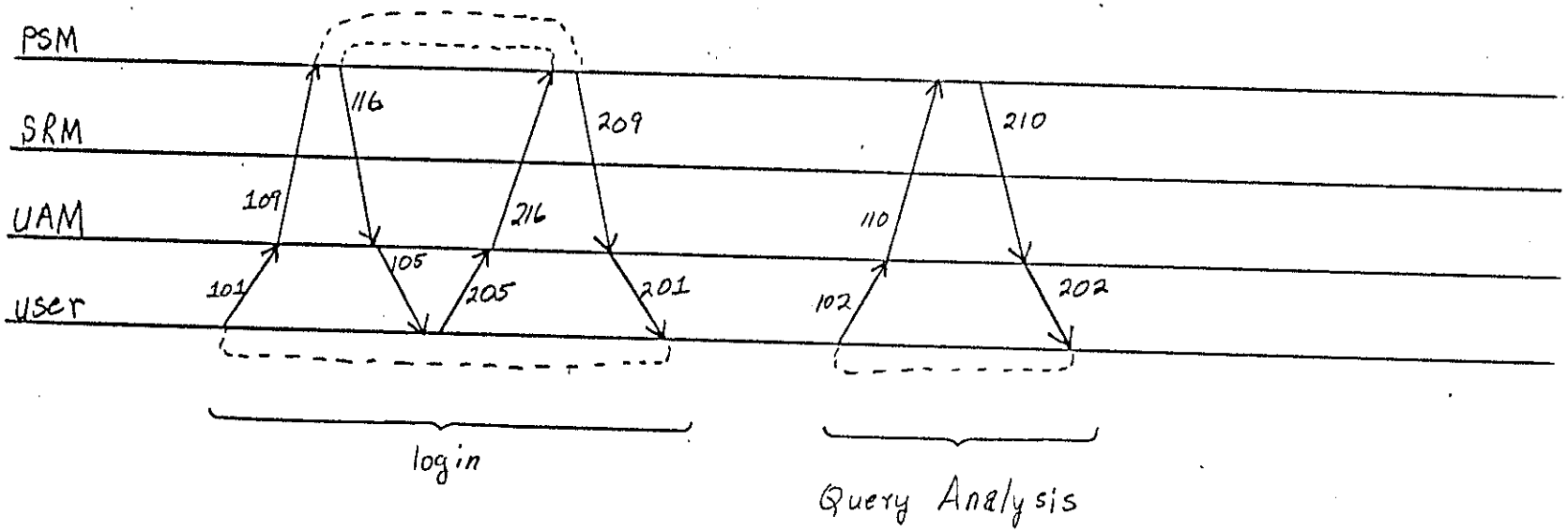


Figure 17. An Example of a Message Sequence for Valid Login and Denied Data Access Attempt before the Database is Accessed.

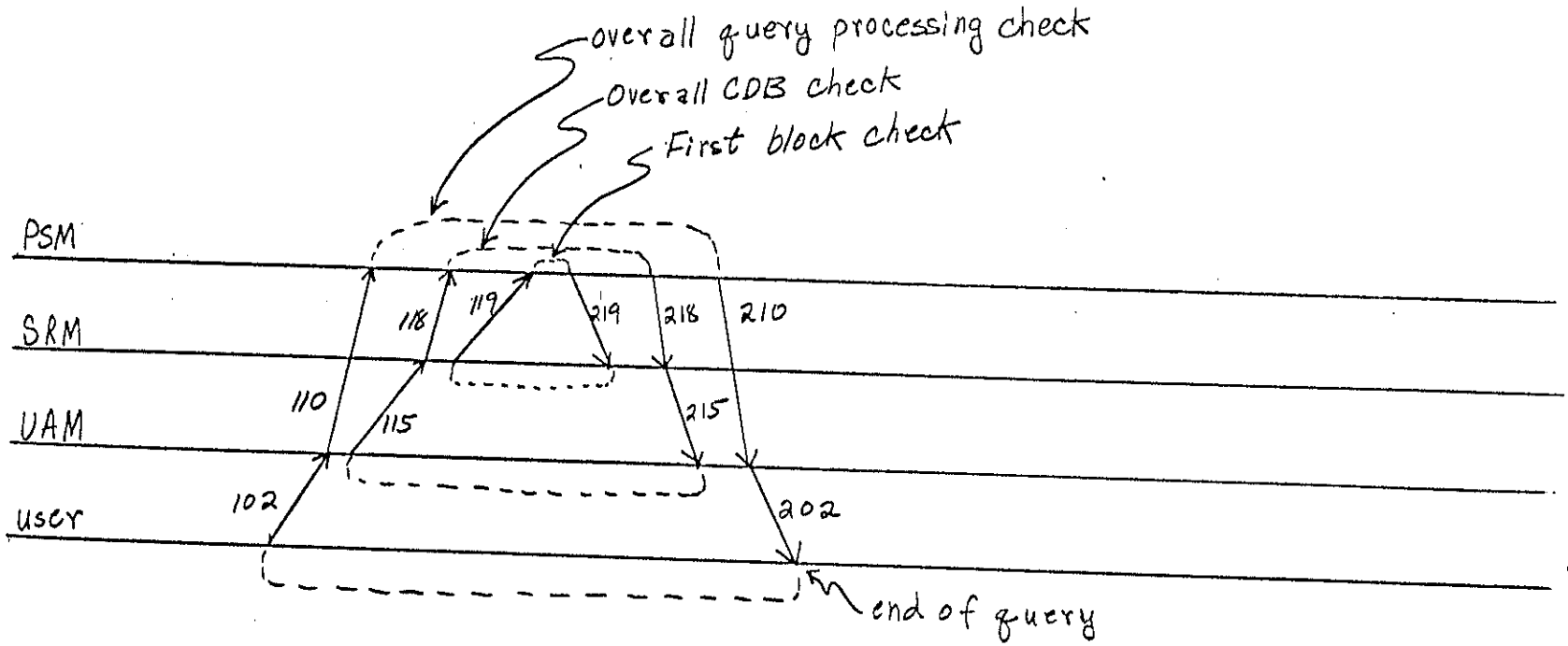


Figure 18. An Example of a Message Sequence for Valid Login and Denied Data Access Attempt after the Database is Accessed.

the authorizer is authenticated, his display request is honored through the 210 and 203 messages. If, however, the authorizer is not authenticated, his display request is ignored and he is notified via the 210 and 203 messages.

An example of a message sequence for the change subtype is given in Figure 20. The authorizer makes a change request (104) which is passed to the PSM (111). The PSM makes a request for authentication which produces the subsequence of messages 113, 107, 207, 108, 208, and 213. (Notice that this subsequence is identical to the second subsequence in Figure 19 for the display subtype.) When the PSM receives the information, the authentication process and the system occupancy checks are completed (between the adjacent 213 and 211 messages on the PSM line). If the change request clears security, then the indicated authorization in the request is changed; otherwise, no changes take place and the authorizer is notified.

3.4.3 Sequence Language

A message sequence language and the grammar for the language are discussed in this section. The grammar rules must preserve the axioms for both message sequences and the request/response pairs, and they must provide a means for generating nested pairs. Also, the grammar rules must not conflict with the syntax rules for generating a single message classification (see earlier section on classification language). In essence, the grammar must generate only legal, secure message sequences; that is, elements of MS.

The syntax rules (given in Appendix B and Appendix C) for generating a secure message sequence form a context-sensitive grammar. (Appendix B and Appendix C are identical

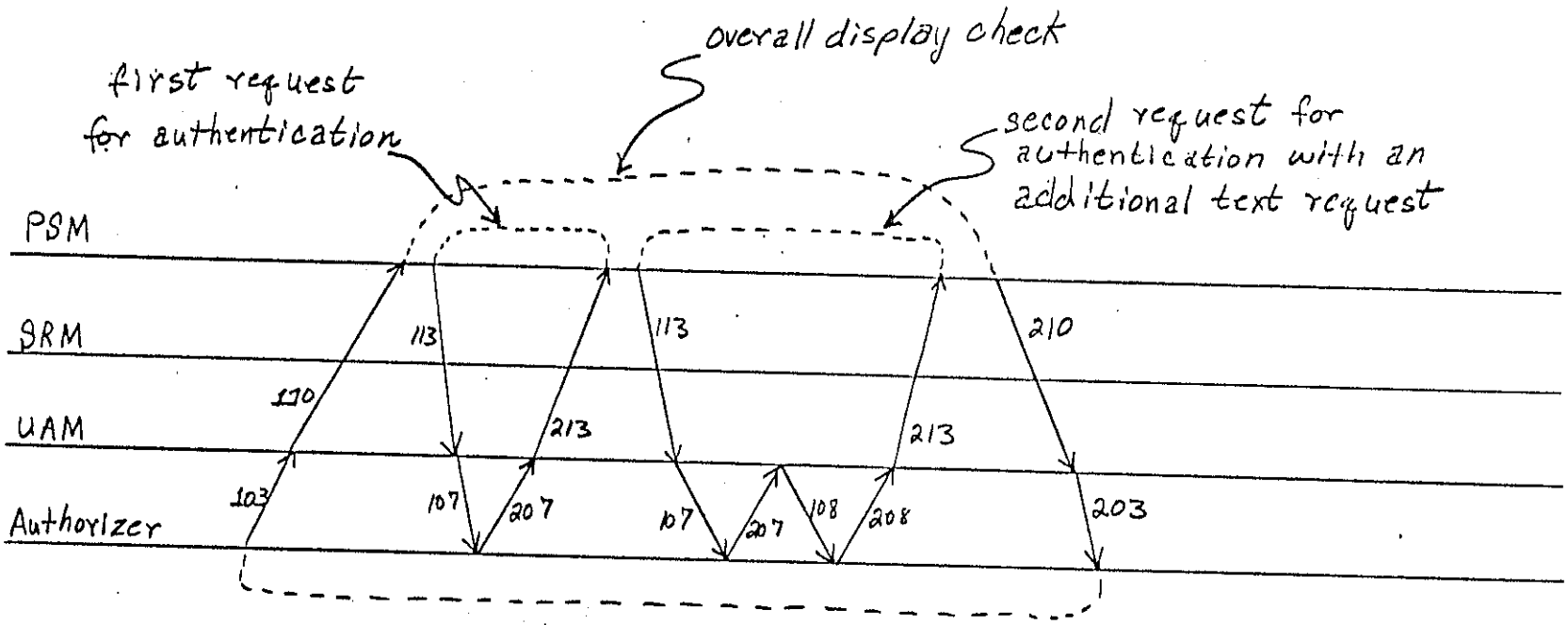


Figure 19. An Example of a Message Sequence for the (Authorization) Display Subtype.

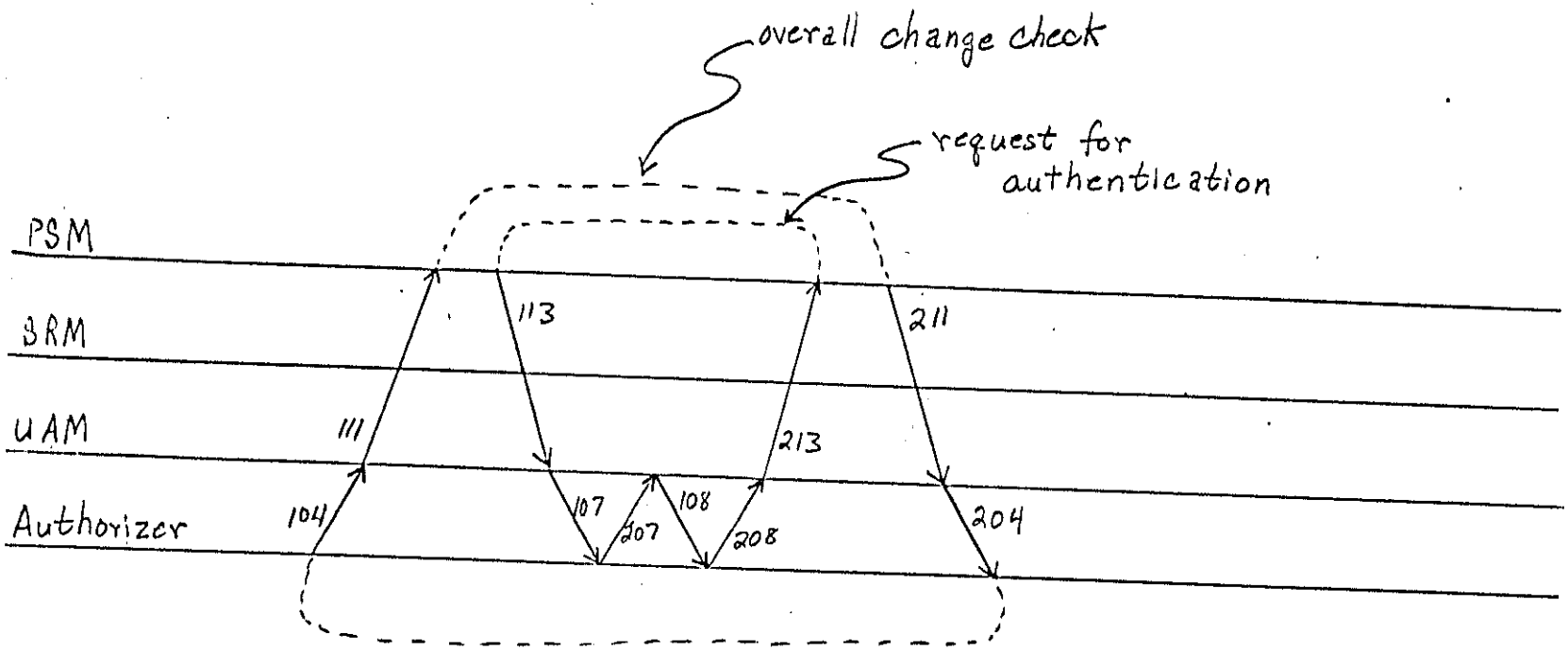


Figure 20. An Example of a Message Sequence for the (Authorization) Change Subtype.

except that Appendix C is an abbreviated version, using the SMC numbers.) These syntax rules generate the secure message sequences which are elements of the set MS. To illustrate the generation process consider the message sequence shown in Figure 14. This message sequence is generated by applying syntax rules 1, 2, 6, 10, 10, 13, 13, and 16 of Appendix B (or Appendix C). A derivation tree is given in Figure 21 which uses the grammar rules in Appendix C.

The grammar rules by themselves do not provide all the constraints necessary for generating a secure message sequence. Particularly, the grammar rules provide no relationship among the message ID's in a sequence. For example, suppose for the moment that there are two separate users on two separate terminals and that they are both in the login process. Then how does the system distinguish one user from the other? The message ID in each message descriptor provides the needed distinction. That is, messages in the same sequence have the same message ID. This is expressed in one of the axioms given earlier in the definition for message sequence and request/response pairs. Under the combined constraints of the axioms and the grammar rules, it is possible to construct only secure message sequences.

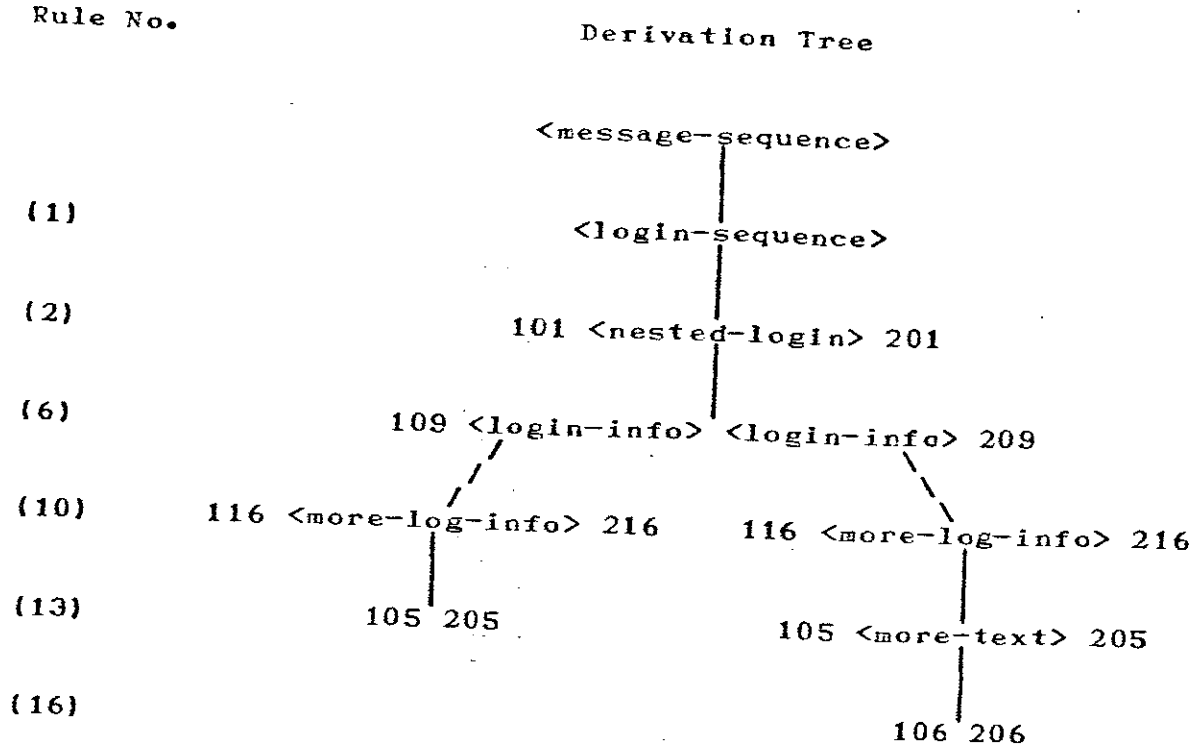


Figure 21. Derivation Tree for the Message Sequence Given in Figure 14.

4. ON-GOING WORK

4.1 Petri Net Modelling

Petri nets [PETEJ77] provide a very useful set of tools for describing and analyzing, at many levels, dataflow networks with concurrent activities. Space limitations do not allow for the development of Petri nets and their applications here. The marking of a Petri net with tokens is related to the concept of system state. It is hoped, in the current work, that formal proof methods applicable to the reachability [LANDI78] of markings in Petri nets will be adaptable to proof methods for the security of message sequences within the proposed system architecture. The idea is to prove that, given only valid message sequences in the data flow, no unsafe system states can be reached. To the knowledge of the authors, this approach has not been previously considered, and, if successful, has the potential to be a significant contribution to the state-of-the-art. (The use of Petri nets has, however, been suggested [AZEMP78] as a means to verify communication procedures.) Petri net representations also appear to be highly suitable for representing the multiple access decision binding times. In addition, there are also relationships between Petri nets and formal language aspects of message sequences.

4.2 Distributed Database Environment

A single arrangement of three modules (UAM, SRM, and PSM) will now be called a "station." The system configuration is easily expanded to a distributed environment by allowing multiple stations which communicate by interstation dataflow. The new problems introduced by this expansion are:

(a) What network configurations can be used?
(b) By what path(s) is interstation commmunication achieved or allowed?
(c) Can interstation messages be classified?
(d) Can valid interstation message sequences be characterized?
(e) How are authorization and enforcement properly distributed over various stations in the network?

Generally, the issues of question (a) above are not in the scope of the current work. Network topology is a field in its own right and the authors plan to take advantage of some of its results. This work will assume a completely connected configuration, of which all other configurations (star, chain, etc.) are subsets (substructures). Figure 22 shows one possible way to connect three stations in a network.

Although it might seem convenient to have SRM's connected together or to have one station's UAM able to make a request directly to another's SRM, the system remains secure if and only if every station appears (functionally) as a user to each other station. This implies that stations are always connected UAM to UAM. In this way, every request, whether arriving directly from a user or by way of another

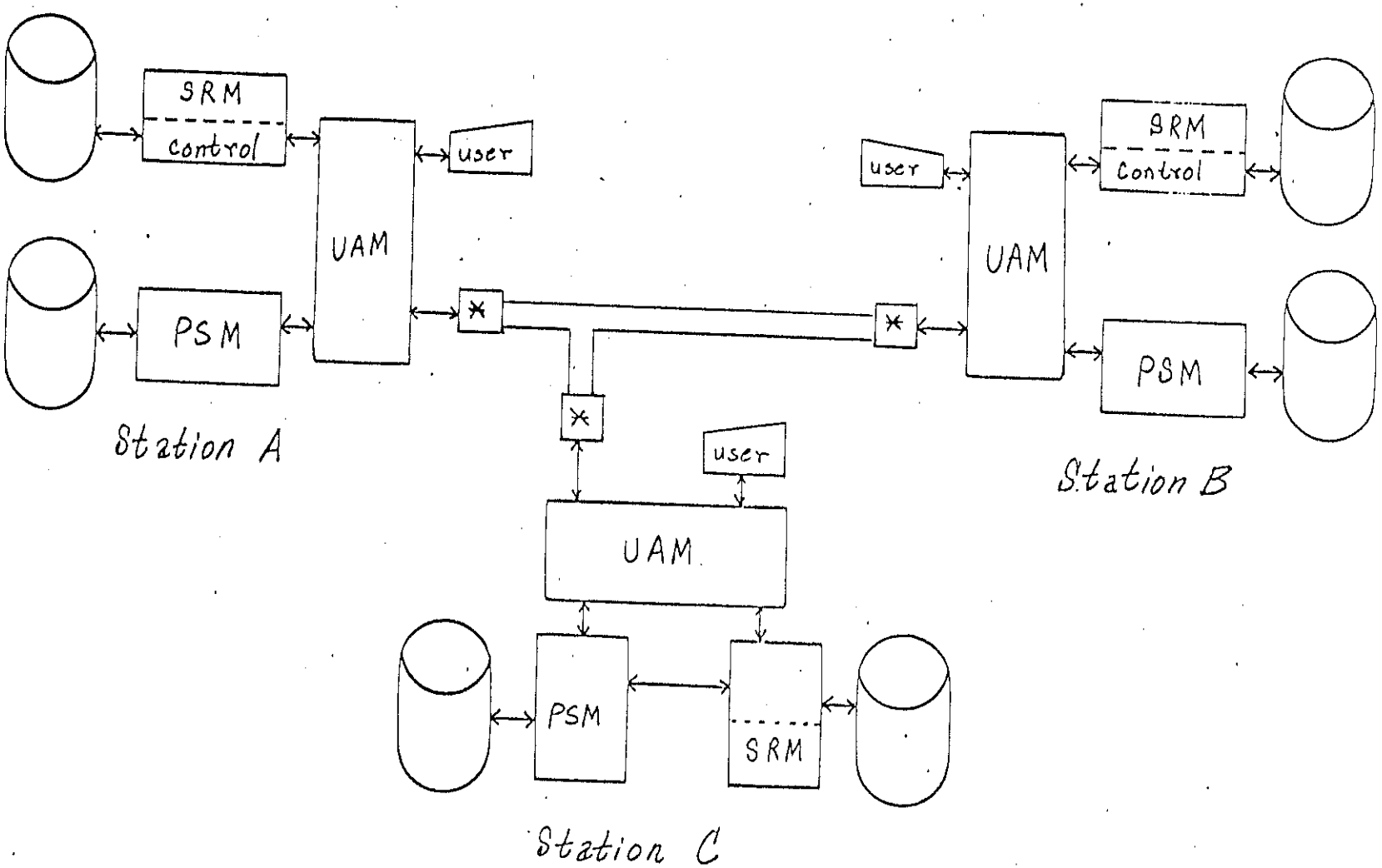


Figure 22. Network of Stations.

station, requires an access decision from the PSM in the station to which the request is directed. Certain front-end message handling operations will also require an interface between each station's UAM and the rest of the network. In Figure 22 that message handling interface is shown as a box enclosing a star. The network communication problems of store-and-forward, message handling, packet switching, and message queuing have been pursued by other researchers and are not considered here.

Interstation messages can generally be classified by function:

(1) database access (store, retrieve, and update)
(2) locate data and other resources (using centralized directory, distributed directory, or general broadcast of request)
(3) user-to-user messages for personal communication
(4) authorizations (from remote authorizers)
(5) enforcement information (both requesting and sending this information)
(6) forwarding a message of any type from another station
(7) acknowledgement (handshaking between message handlers)

The authors expect that this classification will be refined as the research progresses. Further, it is a goal to apply formal grammars to describe valid interstation message sequences, just as we have done for intrastation messages in the centralized case.

The question of how authorization and enforcement are to be properly distributed in the network is interesting, and its answers can have serious effects on overall security. We presently have some preliminary principles which are currently under investigation. The principles are based on the assumption that data is located at a particular station usually because it was created or entered there, or at least because those who will control its use typically are users local to that station. Two of those principles are:

(a) Authorization for access to data stored at a given station will be done at that station.
(b) Enforcement for access to data stored at a given station will be done at that station.

The first principle does not imply that the authorizer must be a local user of the station in question; a remote authorizer may enter authorizations by way of interstation messages. However, authorization information for data will remain in the station containing that data. Most importantly, enforcement will be carried out at the station containing the data to be accessed. This is in contrast to carrying out enforcement at the station local to the requesting user. The authors believe it is more secure to associate the enforcement function with the data being protected rather than with the requesting user.

A third principle under investigation is that messages are to be considered as resources, like data. All message handling will then be governed by access control rules. Both the sending and receiving of messages will be subject to enforcement checking. The enforcement decision can be based on attributes such as source, target, message class, and message content, as well as other more general system conditions. An interesting side effect of this general

approach to communication protection is that access rules can now be used to impose user views of the network configuration.

5. CONCLUSION

This is an interim technical report discussing an ongoing project. As such, it is incomplete and probably contains errors and inconsistencies--especially since some of the concepts concerning message sequences were in a highly changeable state as the report was being written.

Within the framework of a proposed dataflow system architecture called MULTISAFE, security of messages and message sequences has been defined in terms of axioms and formal grammars. A PUSH/PULL hardware mechanism (implemented by a private memory organization) is described to enforce proper binding of data and messages. Messages are classified in a hierarchical taxonomy and Petri nets will be used to model and verify message security. Extensions are being considered for distributed protection of distributed data.

REFERENCES

AZEMP78 Azema, P., J. M. Ayache, and B. Berthomieu, "Design and Verification of Communication Procedures: A Bottom-Up Approach," Proc. of the 3rd Conf. on Software Engineering (May 1978) Atlanta, 168-174.

BERRB74 Berra, P. Bruce, "Some Problems in Associative Processor Applications to Data Base Management," Proc. of the National Computer Conf. (1974), pp.1-5.

DEFIC74 DeFiore, Casper R., and P. Bruce Berra, "A Data Management System Utilizing an Associative Memory," Proc. of the National Computer Conf. (1973) pp. 181-185.

ENSLP77 Enslow, P. H., "Multiprocessor Organization--A Survey," ACM Computing Surveys, 9, 1 (March 1977), pp. 103-129.

FERNE75 Fernandez, Eduardo B., Rita C. Summers, and Charles D. Coleman, "An Authorization Model for a Shared Data Base," Proc. ACM-SIGMOD International Conf. on Management of Data, San Jose, Ca., (May 1975), pp. 23-31.

FONGE77 Fong, Elizabeth, "A Database Management Approach to Privacy Act Compliance," NBS Special Publication 500-10, (June, 1977).

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.

HARTH76 Hartson, H. Rex, and David K. Hsiao, "Full Protection Specifications in the Semantic Model for Database Protection Languages," Proceedings of the Annual Conference of the ACM, Houston, Texas, (October 1976), pp. 90-95.

HOFFL77 Hoffman, Lance J., "Privacy Laws Affecting System Design," Memorandum No. UCB/ERL M77-11, Electronics Research Laboratory, University of California, Berkeley, (February 8, 1977).

LANDL78 Landweber, L. H., and E. L. Robertson, "Properties of Conflict-Free and Persistent Petri Nets," Journal of the ACM 25, 3 (July 1978), 352-364.

PETEJ77 Peterson, James L., "Petri Nets," ACM Computing Surveys, Vol. 9, No. 3, (September 1977), pp. 223-252.

TRUER77 Trueblood, Robert, and H. Rex Hartson, "A Study of Multiprocessor Architectures for Supporting Secure Database Management," Tech. Report No. CS77009-R, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, (December 1977).

TURNR76 Turn, Rein, and W. H. Ware, "Privacy and Security Issues in Information Systems," IEEE Transactions on Computers, Vol. C-25, No. 12, (December 1976), pp. 1353-1361.

APPENDIX A

MESSAGE CLASSIFICATION SYNTAX

This appendix contains the syntax for generating message classifications. The syntax presented here is a regular grammar. The strings generated by this grammar are the five-tuples in the set SMC. Each grammar rule is identified by the number in parenthesis on its left. This identification is not part of the grammar rule.

```
(1)  <message-classes> ::= status <status-attribute>
                         | request <request-attribute>
                         | response <response-attribute>
(2)  <status-attribute> ::= UAM <UAM-attribute>
                          | SRM <SRM-attribute>
(3)  <request-attribute> ::= user <user-attribute1>
                           | authorizer <auth-attribute1>
                           | UAM <UAM-attribute1>
                           | PSM <PSM-attribute1>
                           | SRM <SRM-attribute1>
(4)  <response-attribute> ::= user <user-attribute2>
                            | authorizer <auth-attribute2>
                            | UAM <UAM-attribute2>
                            | PSM <PSM-attribute2>
                            | SRM <SRM-attribute2>
(5)  <UAM-attribute> ::= PSM <UAM-PSM-type>
                       | user <UAM-user-type>
(6)  <SRM-attribute> ::= PSM <SRM-PSM-type>
(7)  <user-attribute1> ::= UAM <user-type1>
(8)  <auth-attribute1> ::= UAM <auth-type1>
(9)  <UAM-attribute1> ::= user <UAM-user-type1>
                        | authorizer <UAM-auth-type1>
                        | PSM <UAM-PSM-type1>
                        | SRM <UAM-SRM-type1>
(10) <PSM-attribute1> ::= UAM <PSM-UAM-type1>
                        | SRM <PSM-SRM-type1>
(11) <SRM-attribute1> ::= PSM <SRM-PSM-type1>
                        | UAM <SRM-UAM-type1>
(12) <user-attribute2> ::= UAM <user-UAM-type2>
(13) <auth-attribute2> ::= UAM <auth-UAM-type2>
(14) <UAM-attribute2> ::= user <UAM-user-type2>
                        | authorizer <UAM-auth-type2>
                        | PSM <UAM-PSM-type2>
                        | SRM <UAM-SRM-type2>
(15) <PSM-attribute2> ::= UAM <PSM-UAM-type2>
                        | SRM <PSM-SRM-type2>
(16) <SRM-attribute2> ::= UAM <SRM-UAM-type2>
                        | PSM <SRM-PSM-type2>
(17) <UAM-PSM-type> ::= termination
(18) <UAM-user-type> ::= announcement
(19) <SRM-PSM-type> ::= end-of-data
(20) <user-type1> ::= access <user-sub1>
(21) <auth-type1> ::= authorization <auth-sub1>
(22) <UAM-user-type1> ::= enforcement <UAM-user-enf-sub1>
                        | additional-text
(23) <UAM-auth-type1> ::= enforcement <UAM-auth-enf-sub1>
                        | additional-text
(24) <UAM-PSM-type1> ::= enforcement <UAM-PSM-enf-sub1>
```

```
                                | authorization <UAM-PSM-auth-sub1>
(25) <UAM-SRM-type1> ::= access <UAM-SRM-acc-sub1>
(26) <PSM-UAM-type1> ::= enforcement <PSM-UAM-enf-sub1>
(27) <PSM-SMM-type1> ::= enforcement <PSM-SRM-enf-sub1>
(28) <SRM-PSM-type1> ::= enforcement <SRM-PSM-enf-sub1>
(29) <SRM-UAM-type1> ::= data-flow <SRM-UAM-data-sub1>
(30) <user-UAM-type2> ::= enforcement <user-UAM-enf-sub2>
                        | additional-text
(31) <auth-UAM-type2> ::= enforcement <auth-UAM-enf-sub2>
                        | additional-text
(32) <UAM-user-type2> ::= access <UAM-user-acc-sub2>
(33) <UAM-auth-type2> ::= authorization <UAM-auth-auth-sub2>
(34) <UAM-PSM-type2> ::= enforcement <UAM-PSM-enf-sub2>
(35) <UAM-SRM-type2> ::= data-flow <UAM-SRM-data-sub2>
(36) <PSM-UAM-type2> ::= enforcement <PSM-UAM-enf-sub2>
                       | authorization <PSM-UAM-auth-sub2>
(37) <PSM-SRM-type2> ::= enforcement <PSM-SRM-enf-sub2>
(38) <SRM-UAM-type2> ::= access <SRM-UAM-acc-sub2>
(39) <SRM-PSM-type2> ::= enforcement <SRM-PSM-enf-sub2>
(40) <user-sub1> ::= login | data
(41) <auth-sub1> ::= display | change
(42) <UAM-user-enf-sub1> ::= information
(43) <UAM-auth-enf-sub1> ::= information
(44) <UAM-PSM-enf-sub1> ::= login-decision
                          | data-decision
                          | display-decision
                          | change-decision
(45) <UAM-PSM-auth-sub1> ::= display | change
(46) <UAM-SRM-acc-sub1> ::= CDB
(47) <PSM-UAM-enf-sub1> ::= information
(48) <PSM-SRM-enf-sub1> ::= information
(49) <SRM-PSM-enf-sub1> ::= overall-decision
                          | block-decision
(50) <SRM-UAM-data-sub1> ::= buffer | data
(51) <user-UAM-enf-sub2> ::= information
(52) <auth-UAM-enf-sub2> ::= information
(53) <UAM-user-acc-sub2> ::= login | data
(54) <UAM-auth-auth-sub2> ::= display | change
(55) <UAM-PSM-enf-sub2> ::= information
```

(56) <UAM-SRM-data-sub2> ::= buffer | data

(57) <PSM-UAM-enf-sub2> ::= decision

(58) <PSM-UAM-auth-sub2> ::= display | change

(59) <PSM-SRM-enf-sub2> ::= overall-decision
| block-decision

(60) <SRM-UAM-acc-sub2> ::= CDB

(61) <SRM-PSM-enf-sub2> ::= information

APPENDIX B

MESSAGE SEQUENCE SYNTAX

This appendix contains the syntax for generating message sequences. The syntax presented here is a context-sensitive grammar. The strings generated by this grammar are the secure message sequences. Each grammar rule is identified by the number in parenthesis on its left. This identification is not part of the grammar rule. Appendix C contains these same rules in the same order except that the rules are constructed with SMC numbers.

```
(1) <message-sequence> ::= <login-sequence>
                         | <data-sequence>
                         | <display-sequence>
                         | <change-sequence>

(2) <login-sequence> ::= request user UAM access login
                         <more-text>↑n <nested-login>
                         response UAM user access login

(3) <data-sequence> ::= request user UAM access data
                        <more-text>↑n <nested-data>
                        response UAM user access data

(4) <display-sequence> ::= request authorizer UAM
                           authorization display
                           <auth-text>↑n <nested-display>
                           response UAM authorizer
                           authorization display

(5) <change-sequence> ::= request authorizer UAM
                          authorization change
                          <auth-text>↑n <nested-change>
                          response UAM authorizer
                          authorization change

(6) <nested-login> ::= request UAM PSM enforcement
                       login-decision <log-info>↑n
                       response PSM UAM enforcement
                       login-decision

(7) <nested-data> ::= request UAM PSM enforcement
                      data-decision <data-info>
                      response PSM UAM enforcement
                      data-decision

(8) <nested-display> ::= request UAM PSM authorization
                         display <authentication>↑n
                         response PSM UAM authorization
                         display

(9) <nested-change> ::= request UAM PSM authorization change
                        <authentication>↑n
                        response PSM UAM authorization change

(10) <log-info> ::= request PSM UAM enforcement information
                    <more-log-info>↑n
                    response UAM PSM enforcement information

(11) <data-info> ::= request UAM SRM access CDB
                     <more-data-info>
                     response SRM UAM access CDB

(12) <authentication> ::= request PSM UAM enforcement
                          information <more-authen-info>↑n
                          response UAM PSM enforcement
                          information

(13) <more-log-info> ::= request UAM user enforcement
                         information <more-text>↑n
                         response user UAM enforcement
                         information

(14) <more-data-info> ::= request SRM PSM enforcement
                          overall-decision <data-block>↑n
                          response PSM SRM enforcement
                          overall-decision

(15) <more-authen-info> ::= request UAM authorizer
                            enforcement information
                            <auth-text>↑n
                            response authorizer enforcement
```

information

(16) <more-text> ::= request UAM user additional-text
response user UAM additional-text

(17) <auth-text> ::= request UAM authorizer additional-text
response authorizer UAM additional-text

(18) <data-block> ::= request SRM PSM enforcement
block-decision <block-check>†n
response PSM SRM enforcement
block-decision <buffer>

(19) <block-check> ::= request PSM SRM enforcement
information
response SRM PSM enforcement
information

(20) <buffer> ::= request SRM UAM dataflow buffer
response UAM SRM dataflow buffer
request SRM UAM dataflow data
response UAM SRM dataflow data

APPENDIX C

MESSAGE SEQUENCE SYNTAX WITH SMC NUMBERS

This appendix contains the same syntax rules in the same order as Appendix B except that the rules in this appendix are constructed with SMC numbers. Each grammar rule is identified by the number in parenthesis on its left. This identification is not part of the grammar rules.

(1) <message-sequence> ::= <login-sequence>
| <data-sequence>
| <display-sequence>
| <change-sequence>

(2) <login-sequence> ::= 101 <more-text>↑n
<nested-login> 201

(3) <data-sequence> ::= 102 <more-text>↑n
<nested-data> 202

(4) <display-sequence> ::= 103 <auth-text>↑n
<nested-display> 203

(5) <change-sequence> ::= 104 <auth-text>↑n
<nested-change> 204

(6) <nested-login> ::= 109 <login-info>↑n 209

(7) <nested-data> ::= 110 <data-info> 210

(8) <nested-display> ::= 113 <authentication>↑n 213

(9) <nested-change> ::= 114 <authentication>↑n 214

(10) <log-info> ::= 116 <more-log-info>↑n 216

(11) <data-info> ::= 115 <more-data-info> 215

(12) <authentication> ::= 116 <more-authen-info>↑n 216

(13) <more-log-info> ::= 105 <more-text>↑n 205

(14) <more-data-info> ::= 118 <data-block>↑n 218

(15) <more-authen-info> ::= 107 <auth-text>↑n 207

(16) <more-text> ::= 106 206

(17) <auth-text> ::= 108 208

(18) <data-block> ::= 119 <block-check>↑n 219 <buffer>

(19) <block-check> ::= 117 217

(20) <buffer> ::= 120 220 121 221