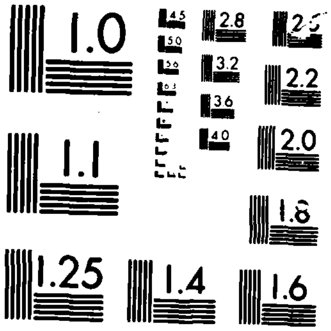
AD-A169 187   MULTILEVEL OBJECT SECURITY MODEL(U) SYTEK INC MOUNTAIN   1/2
VIEW CA E C SULLIVAN ET AL. MAR 86 SYTEK-TR-85015
RADC-TR-86-10 F30602-85-C-0001

UNCLASSIFIED   F/G 9/2   NL



MICROCOPY [illegible] CHART

RADC-TR-86-10
Final Technical Report
March 1986

MULTILEVEL OBJECT SECURITY MODEL

SYTEK, Incorporated

Elisabeth C. Sullivan, Teresa F. Lunt and Norman Proctor

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-86-10 has been reviewed and is approved for publication.

APPROVED: David F. Trad

DAVID F. TRAD
Project Engineer

APPROVED:

GERALD D. LONG, Colonel, USAF
Chief, Command & Control Division

FOR THE COMMANDER: Richard W. Pouliot

RICHARD W. POULIOT
Plans & Programs Division

DESTRUCTION NOTICE - For classified documents, follow the procedures in DOD 5200.22-M, Industrial Security Manual, Section II-19 or DOD 5200.1-R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

AD-A169 187

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release; distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
TR-85015

5. MONITORING ORGANIZATION REPORT NUMBER(S)
RADC-TR-86-10

6a. NAME OF PERFORMING ORGANIZATION
SYTEK, Incorporated

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
Rome Air Development Center (COTC)

6c. ADDRESS (*City, State, and ZIP Code*)
1225 Charleston Road
Mountain View CA 94039-7225

7b. ADDRESS (*City, State, and ZIP Code*)
Griffiss AFB NY 13441-5700

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Rome Air Development Center

8b. OFFICE SYMBOL (*If applicable*)
COTC

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F30602-85-C-0001

8c. ADDRESS (*City, State, and ZIP Code*)
Griffiss AFB NY 13441-5700

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
35167G	1065	01	01

11. TITLE (*Include Security Classification*)
MULTILEVEL OBJECT SECURITY MODEL

12. PERSONAL AUTHOR(S)
Elisabeth C. Sullivan, Teresa F. Lunt, Norman Proctor

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM Jan 85 TO Nov 85

14. DATE OF REPORT (*Year, Month, Day*)
March 1986

15. PAGE COUNT
116

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
Computer security
multilevel security
formal security model

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This report is a summary of the Multilevel Object Security Model (MOSM) project. The MOSM effort involved the development of an intermediate-level security model which satisfies the requirements of the Bell-Lapadula model and the NRL model, while incorporating the ease of verification of the Bell-Lapadula model and the flexibility of the user interface of the NRL model. The project consisted of three tasks: development of the model, an abstract implementation of the model in terms of the NRL model, and an analysis of the Diamond Document Handling system with respect to the model. Each of these tasks is described in detail and a formal statement of the MOSM appears in the appendix.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☒ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
David F. Trad

22b. TELEPHONE (*Include Area Code*)
(315) 330-2925

22c. OFFICE SYMBOL
RADC (COTC)

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

CONTENTS

1. INTRODUCTION

The Multilevel Security Object Model project, performed for RADC by SYTEK, under contract number F30602-85-C-0001, consists of the development of an intermediate-level security model which satisfies the requirements of the Bell-Lapadula model and the NRL model, while incorporating the ease of verification of the Bell-Lapadula model and the flexibility of the user interface description of the NRL model. This report is a summary of the project. The three tasks of the project were the development of the model, an abstract implementation of the model in terms of the NRL model, and an analysis of the Diamond Document Handling system with respect to the model. Each task is described in detail in the following report, and a formal statement of the model developed appears in the appendix.

THE MLO MODEL--TASK 1

THE DEVELOPMENT OF THE MODEL

Elisabeth C. Sullivan

2. TASK 1: Development of the MLO Model

A formal security model describes security requirements for a trusted computer system. Often it is formally verified that a computer system obeys the rules of its security model. To be useful for verification, a model must be designed so that application of a verification technique is feasible. For many applications, currently existing models either present verification difficulty or have certain deficiencies which restrict their usefulness. This paper describes a security model which facilitates system verification and addresses some of these deficiencies.

Models can be described in terms of *subjects* (individuals or processes) who perform *operations* on *objects* (databases or variables). Most formal models of security requirements have considered each system object to have one security level which represents the classification of the object as a whole and of each of its parts. In some instances this view is quite adequate. In other instances, security requirements must consider *multilevel objects*, or objects having having parts with differing security levels. The Multilevel Object Security (MLO) Model acknowledges the existence of multilevel objects, and hence allows more flexibility in system development. It is designed to meet security policy needs which are especially applicable for message systems or document handling systems. The MLO model implements multilevel objects using an object hierarchy and a novel construct called a reference mechanism which provides an access path through the hierarchy. The MLO model permits downgrading in special instances and allows the use of application-specific security constraints. This flexible model has been designed to meet the DoD Trusted Computer System Evaluation Criteria for A1 systems [1].

2.1 Background

The primary focus of this task was to develop a multilevel object security model which could be used for formal verification of message or document systems. However, the model describes other useful concepts, as seen in the list of goals for the model below.

- The model design must be such that verification of systems implementing the model is feasible.

- The model must represent and process multilevel objects.

- The model must allow selective downgrading without violating the '*' property.

- The model must allow for the addition of application-specific security rules.

- The model must meet the criteria for models for A1 certification in [1].

The following background information provides insight into why these five criteria became objectives for the MLO model development.

2.1.1 Ease of Verification

A security model is most useful when it is feasible to verify systems which implement it. One contributing factor to verifiability is the architecture of the system. A security model describes the security constraints for the interface to the security mechanism. Hence, the use of a security model makes an assumption about the architecture of the system for which the model is used. For example, the widely used Bell-Lapadula model describes security constraints on a trusted computing base which controls read and write access to storage objects. The security constraints must be sufficient to guarantee that the security will be maintained regardless of the properties of the untrusted software which makes use of the mechanism modeled by the Bell-Lapadula model. The advantage of the Bell-Lapadula model is that the mechanism described by the model is very primitive. Hence verification is more feasible than if the entire system had to be verified. There is, however, a penalty associated with modeling such a primitive mechanism. The penalty is that for more primitive mechanisms stricter security constraints are required to guarantee that untrusted software using the mechanism can not cause security violations.

The NRL model describes security constraints for a particular type of system, a military message system, at the user level. Because the model describes the security constraints at the user interface level, it can describe exactly the restrictions required for security. As a result, it allows maximum user flexibility consistent with security. The penalty associated with a user level model such as the NRL model is increased verification difficulty because all software up to the user interface must be verified.

2.1.2 Multilevel vs. Single Level Objects

Computer security constraints are typically modeled after the constraints used in the "people-paper" world, where user rules are modeled after rules for human users of classified objects. An example of a classified object in the "people-paper" world is a classified document. In addition to the security level of the document as a whole, each paragraph within the document has its own security level. Typically the levels of some paragraphs are lower than the level of the document as a whole. Generally, paragraphs whose security level is lower than the document level can be viewed by individuals having a clearance level which is at least as high as the security level of the paragraph, but not necessarily as high as the security level of the document as a whole. If a computer application of a document handling system models a document as having a single security level, it may deny an individual's access to paragraphs he is really qualified to view. For this example, viewing a document as having a single security level may be too restrictive.

The failure of most security models to account for objects which can have one security level and parts which can have their own individual levels has made these models overly restrictive for many applications. The need for multilevel objects was a topic of study for Group 2 of the 1982 Air Force Summer Study on Multilevel Secure Databases. This group noted that multilevel objects were an essential ingredient of any formal security model to be used for such applications. The need for multilevel objects was also noted in the work of Heitmeyer and Landwehr of NRL [2]. Group 2 and the NRL model have both influenced the development of multilevel objects in the MLO model.

2.1.3 Downgrading

Two standard properties desired in a security model are the simple security property and the '*' property. For many people, these two properties are synonymous with the Bell-Lapadula model. The simple security property forbids a subject to read information at a security level which is higher than the subject clearance. This requirement corresponds directly to a standard security requirement in the "people-paper" world--an individual cannot view information which is classified at a level higher than his clearance. The '*' property expressly forbids a subject to write information at a security level which is lower than the subject clearance. This requirement is intended to place constraints on software code, as cleared people are trusted to write classified data at an appropriate level. The '*' property effectively describes constraints to prevent untrusted code from making information of one security level available to untrusted code at a lower level. Unfortunately, the property is often overly restrictive, for it strictly forbids downgrading (lowering of classification), and some real systems require the ability to reduce the classification of information.

One way of writing systems which adhere to the '*' property and allow some downgrading has been the use of "trusted processes". A mechanism is designed which strictly adheres to the Bell-Lapadula model. Trusted processes are created which allow downgrading and reside outside the mechanism implementing the Bell-Lapadula model. The difficulty is that assurance of the security of the system relies on assurance that the mechanism follows the policy properly, and that the trusted processes perform their tasks properly, and are only invoked by properly cleared users. Often the verification of trusted processes is less rigorous than the verification of the mechanism, or is simply ignored.

The NRL model explicitly allows only properly qualified users to downgrade data, and forbids information from being copied to an object of a lower level than the information. Because the NRL models the user interface, subjects are individuals who are trusted to write data at an appropriate security level as in the "people-paper" world. There are no subjects which are processes, so the concept of trusted or untrusted processes does not arise, and therefore the '*' property is not relevant to the NRL model.

2.1.4 Application-Specific Security Rules

Systems often require permissions to perform particular tasks, such as releasing messages or altering specific internal tables. The Bell-Lapadula model has no mechanism for deciding if a user is authorized to perform a certain task. The NRL model controls such access using the concept of "roles". A role is the job or task a user is performing. To act in a given role, the user must be authorized for it. With each role comes authorization to perform certain operations. Using this mechanism, it is possible to have control over which users can perform what tasks. Such a capability increases the usefulness and flexibility of a model.

2.1.5 A1 Evaluation Criteria

In 1983, the DoD published a collection of evaluation criteria for trusted computer systems [1]. The highest rating, called A1, lists a formal security policy model as one of its requirements. For many applications, it is desirable to achieve this rating from the government. An A1 rating for a system requires many steps of verification and accountability. One step is the existence of a formal security policy model which meets the definition below, found in [1].

> Formal Security Policy Model - A mathematically precise statement of a security policy. To be adequately precise, such a model must represent the initial state of the system the way in which the system progresses from one state to another, a definition of a "secure" state of the system. To be acceptable as a basis for a TCB, the model must be supported by a formal proof that if the initial state of the system satisfies the definition of a "secure" state and if all assumptions required by the model hold, then all future

states of the system will be secure.

Meeting the above definition is a necessary but not sufficient requirement for A1 certification. However, developing a computer system using a model which has already been established to meet the above definition reduces the number of steps which must be performed to earn A1 certification. One model which meets the definition above is the Bell-Lapadula model. However, this model is not appropriate for systems addressing multilevel objects. The NRL model is appropriate for multilevel object systems but there is no formal proof that the NRL model meets its axioms.

The MLO model has been formally specified and a verification that the model meets its axioms has been performed for RADC. This work, which establishes that the MLO model meets the above definition, can be seen in section 2.3.

2.2 The MLO Model

The Multilevel Object Security Model meets the objective of ease of verification by describing an intermediate level mechanism which lies between the primitive level mechanism of the Bell-Lapadula model and the user level mechanism of the NRL model. The diagram below illustrates the three types of mechanism.

```
+----------+ <--NRL    +------------+            +-----------+
|          |           | UNTRUSTED  |            |           |
| TRUSTED  |           | MULTILEVEL |            | UNTRUSTED |
| MILITARY |           | OBJECT     |            |APPLICATION|
| MESSAGE  |           | PROCESSING |            |           |
| SYSTEM   |           | SYSTEM     |            +-----------+
+----------+           +------------+ <--MLO     |           |
|          |           |            |            | UNTRUSTED |
| TRUSTED  |           |  TRUSTED   |            |OS FUNCTION|
|OPERATING |           | MULTILEVEL |            |           |
| SYSTEM   |           |   OBJECT   |            +-----------+ <--Bell-
+----------+           |  HANDLER   |            |           |    Lapadula
                       +------------+            |  TRUSTED  |
                                                 | COMPUTING |
                                                 |   BASE    |
                                                 +-----------+
```

Figure 1. Interface Described by NRL, MLO and Bell-Lapadula Models

This model incorporates the ease of verification afforded by the Bell-Lapadula model, as it describes objects at a fairly primitive level. Describing user level functions in terms of the operations of the MLO model is a straightforward task. Thus the MLO model provides flexibility in describing user security constraints, without the large verification penalty of the NRL model.

The following paragraphs describe the design of the MLO model.

2.2.1 Representing Multilevel Objects Using MLO Entities

A multilevel object is described within the MLO model by a hierarchy of MLO *entities*. Entities are the single-level primitive units of the model, and can be atoms or containers. Atoms contain data, while containers contain a description of the entities or parts which make up the container. A container may contain atoms and other containers. Thus an object can be represented by a single MLO atom or by a hierarchy of MLO entities. The diagram below describes an example of a hierarchy representing a multilevel object. The entities whose labels begin with a "c" represent containers and those beginning with an "a" represent atoms.

```
                 c11
                  |
         ---------+--------
         |     |     |    |
        a21   c22   a23  c24
               |          |
           +--+--+     +--+--+
           |  |  |     |     |
          a31 a32 a33 c34   a35
                       |
                    +--+--+
                    |     |
                   a41   a42
```

Figure 2. A Multilevel Object Represented as a Hierarchy of MLO Entities

Each entity has a unique entity identifier (symbolic name or descriptor) associated with it, contains a single security level, is clearly marked as an atom or a container, has a status field and a text field. The status field could be interpreted to be a "released" status, for example. The hierarchy of multilevel objects for this model could be interpreted as a collection of messages in a message processing system or documents in a document handling system.

Although the MLO model is described in terms of entities and operations on entities, it is designed assuming that the entities will be organized into hierarchies representing multilevel objects. To facilitate the manipulation of these object requires a rule describing the correct relationship between the entities of the hierarchy. This requires careful definition of security levels and the relations between them.

Security levels are represented as members of a set which is partially ordered with respect to a relation higher-or-equal. Now higher is a partial ordering, and hence is only defined for certain pairs of security levels. Other pairs are not comparable, and the relation higher-or-equal is not defined for them. In this paper, we shall say that security level A dominates security level B if:

> A and B are comparable and
> A is higher-or-equal to B

To "raise" a security level means to change the security level to one which dominates it, while the to "lower" a security level means to change the security level to one which it dominates. The precise definition of dominates depends on the construction of the security level itself, and is application-specific.

Using the above definitions, we can define a security level hierarchy rule as

> The security level of a container must dominate
> the security level of anything it contains.

In essence this rule requires that a total ordering of security levels must exist within an entity hierarchy.

2.2.2 Reference to an Entity

Entities represent the information stored within the trusted multilevel object handler described in figure 1. Reference to an entity is carefully controlled by the mechanism implementing the MLO model. Usually reference to the entity means directly accessing it. However, requirements may dictate that a subject have access to certain containers of the entity in order to access the entity itself. These requirements may be in addition to the mandatory and discretionary access requirements for the entity. In one example, access to a member of an entity hierarchy requires access to the entire hierarchy. This requirement is independent of security levels. Another example is "container clearance required (CCR)". One might be required to have the proper clearance to view an entire message before being permitted to view the contents of a single unclassified field of the message. The subject must be cleared to the level of a CCR container of an entity before he can view the entity. For an example of a CCR requirement, suppose the set of security levels is the standard set {UNCLASSIFIED, CONFIDENTIAL, SECRET, TOP SECRET} and suppose a document classified TOP SECRET contains paragraphs marked at all different levels. If the document is marked CCR, an individual whose clearance is CONFIDENTIAL would be denied access to the UNCLASSIFIED and CONFIDENTIAL paragraphs as well as the paragraphs at a higher level. If the document is not marked CCR, then an individual whose clearance is CONFIDENTIAL could have access to the CONFIDENTIAL and UNCLASSIFIED paragraphs. Another use of CCR is a SECRET document in which all the paragraphs are labeled UNCLASSIFIED, yet an individual must be cleared to the level of SECRET to read any part of the document.

To model requirements such as CCR, we have chosen to describe a *system reference mechanism* which grants reference to any entity by identifying a pathway through the hierarchy containing the entity. Upon request by a subject, the system reference mechanism provides the proper path to an entity. The reference mechanism necessarily has knowledge about the structure of the hierarchy, the security constraints of the data entities, and specific information such as container clearance requirements. When access to an entity is requested by a subject, the system reference mechanism provides an *entity reference* to the entity, which is the pathway described above. The pathway is an ordered collection of entity descriptors, the last of which is the descriptor of the entity requested for viewing, called the *final entity*.

The returned pathway may refer to a single entity, or it may describe a series of containers and supercontainers of the desired entity. If there is no access restriction for entity, then the pathway could consist of the single entity. If there is an access restriction such as CCR, the pathway could describe several containers of the entity. If no pathway is identified, then the subject does not have the special access restrictions. If a pathway is returned by the reference mechanism, then the security levels of all the entities on the pathway can be identified. Using this information, it can be determined if the subject has the proper clearance. If he does, then he can obtain the entity he requested, and no other entity on the pathway. In this manner, the reference mechanism provides an auxiliary access control check which can be used for access restrictions such as CCR.

To avoid overly restricting the implementation of the MLO model, the reference mechanism is purposefully modeled abstractly. There are many options for implementation. In the event that there are no auxiliary requirements such as CCR, the reference mechanism could be an identity function. Whatever implementation is chosen, it must meet certain requirements. They are listed below.

- If a subject requests reference to an entity which does not exist or there is no valid pathway by which a subject can reference an entity, the reference mechanism does not return a reference.

- If there is a valid pathway to an entity for a subject, the reference returned to a subject is the reference he must use at that point in time.

- The subject can only access the final entity of the returned reference, which is the entity he requested, and the access is controlled by the constraints of the operation he is invoking. He can only determine the security level of other entities in the reference.

- References cannot be created or altered by operations or by subjects.

- A returned reference is a correct pathway through the entity hierarchy.

  That is to say each successive entity is contained directly in the previous entity, and the final element of the sequence is the descriptor of the desired entity.

Verification that the subject has container clearance to an entity can be modeled by use of the reference mechanism in the following way. When a subject requests access to an entity, the reference provided includes the descriptors of CCR containers on the pathway to the entity. To view this entity using the path provided, the subject clearance must be greater than or equal to the highest level of any entity in the reference.

2.2.3 Subjects and Downgrading

The objective of allowing downgrading is made possible by controlling subjects within the model. Downgrading is permitted or denied by the constraints of operations within the model. The use of security levels for subjects makes the evaluation of constraints possible.

One form of subject defined for the MLO model is a user, or individual who is properly authorized to use the system being modeled. How a user is allowed to access and manipulate data is determined by his security clearance. In the MLO model, we wish to implement the '*' property, avoid the need for trusted processes and maintain the security level hierarchy rule. At first glance, this seems impossible. For example, suppose a security level is one of the set {UNCLASSIFIED, CONFIDENTIAL, SECRET, TOP SECRET} and a user who is functioning at the TOP SECRET level wishes to alter information in an UNCLASSIFIED paragraph of a SECRET document. If the user writes the altered paragraph at the UNCLASSIFIED level, he is violating the '*' property by writing information at a level which is lower than his clearance level, but the security level hierarchy is maintained. If he writes the paragraph at the TOP SECRET level, he follows the '*' property but he violates the security level hierarchy rule by writing a paragraph at a level which is greater than the level of the document itself.

By allowing qualified users to generate processes to perform at the lower level, we trust the user to generate a modification of the entity which does not alter the security marking or·level, and to have the update performed without violating either the '*' property or the security level hierarchy rule. Such spawned processes act solely on behalf of the sponsoring user. Any permissions required by the spawned process are controlled by the sponsor and maintained by the mechanism implementing the model. The sponsoring user assigns a container clearance and a data clearance to the process it generates. The container clearance assigned is the level the user would need to access the final entity of the reference provided to him. Using the example above, if the users reference to the UNCLASSIFIED paragraph is SECRET (i. e. there is a SECRET level CCR on the document), then the appropriate container clearance is SECRET. The user is trusted to assign an appropriate container clearance to a process. An appropriate container clearance is the highest level reference the process can request. The sponsoring user is trusted to assign an appropriate data clearance to the process it generates. An appropriate data clearance is the level at which the spawned process is actually to function and is equal to the level of the entity to be manipulated. It is the highest level of entity the process can view or write to. In the example, the appropriate data clearance would be UNCLASSIFIED. Now the untrusted process can reference the entity because it is authorized to do so by the user, but the process can only view or write at the UNCLASSIFIED level, and can only perform the specific manipulation assigned to the specified entity. Hence the data is properly classified and neither the spawned process nor the user violate the '*' property or the security level hierarchy rule.

Implementation details for processes include the creation of user/process tables and a vehicle for users to instruct processes. For the operations performed by processes, it must be shown that the subject is valid; that is the process container clearance is less than on equal to the clearance of the sponsoring user and the process data clearance is less than or equal to the process container clearance.

2.2.4 The MLO Operations

The operations of the MLO model are operations of subjects upon entities. Using these intermediate level operations as building blocks, it is possible to construct the more sophisticated user interface operations on multilevel objects, or hierarchies of entities. Since a table of descriptors of entities in the database will be available outside the model-enforcing mechanism, it is possible for a sequence of operations, equivalent to a scan-database operation, to be invoked.

The MLO operations are presented in two groups. The first group of operations are operations available to processes for accessing and interacting with the database, and the create-process operation. This group represents the crux of the model. The second group of operations represent operations on processes themselves and operations on data items other than the entity database. They are operations whose specification may facilitate implementation of the model, but are not strictly a part of the model.

MLO model operations all require the proper discretionary access permissions. These permissions can be entity access, proper role, operation access, or any combination of permissions specified in the discretionary access table. Some of the mandatory access descriptions are included below. The database operations are as follows:

- Read Entity (atom or container)

  This operation allows a properly qualified process to read an entity.

- Write Entity (atom or container)

  This operation allows a properly qualified process to alter the data contents of an entity to a value specified by a sponsoring user. If the entity is a container, this operation can add, delete or alter the descriptor list of the container. If the entity is an atom, the data field can be written. In either case, the security level of the entity is unchanged.

- Upgrade Entity (atom or container)

  This operation allows a properly qualified process to raise the security level of an entity to the level provided by the process.

- Downgrade Entity (atom or container)

  This operation allows properly qualified processes to lower the security level of an entity to the level provided by the process.

- Change Entity Status (atom or container)

  This operation allows properly qualified processes to alter the status field of an entity to the value provided by the process.

- Create Entity (atom or container)

  This operation allows the creation of a new, empty entity. The security level of the new entity is equal to the data clearance of the invoking process and its status is "new".

- Destroy Entity (atom or container)

  This operation allows a properly qualified process to destroy an entity.

- Create Process

  This operation is available only to users, and it creates or spawns a process and assigns the container and data clearances to the process and initiates the process table.

The second group of operations are

- Modify Discretionary Access Table

- Change Clearance

- Kill process

- Instruct Process

- Display to User

- Query

2.2.5 The MLO Security Policy

The MLO security policy is described in terms of the security level hierarchy rule and access control. The MLO model meets the objective of allowing the addition of application-specific security rules by using a discretionary access control table. This table allows the addition of constraints on access to operations, roles, objects, or subjects. Because operations are expressed in terms of entities, the security properties are expressed in terms of entities. The policy is stated below and its formalization can be seen in [3].

- The Security Level Hierarchy Rule

  The security level of an entity must dominate the security level of anything it contains.

- Access Control

  - Operation Invocation

    Only a valid subject can invoke an operation. A subject is a valid subject if

    The subject is a user

The subject is a process having a sponsor and the sponsor clearance dominates the process container clearance, which in turn dominates the process data clearance.

- Discretionary Access Control

Discretionary access controls are extremely application-specific. Making discretionary access constraints overly specific within the model restricts the usefulness of the model to particular situations. Therefore, this model represents discretionary access control measures in an abstract sense allowing general or specific information to be added for each application of the model. Invocation of operations is controlled using privileges called "subject roles". The job a subject is performing is called a role, and a subject is always associated with exactly one role at any instant. A user can assign a role to which he is authorized to a process he is authorized to spawn. A user can change his own role during a session to any role he is authorized for. Some roles can be assumed by only one subject at a time. With each role comes the ability to perform certain functions. Of particular interest is the role of downgrader. A subject operating in this role can access a special function which lowers the security level level of an entity. Certainly the subject, its current role, the operation, and the entity to be accessed will play a part in deciding discretionary access. These items appear as parameters to the boolean discretionary access table within the MLO model. The model allows subjects with the proper role, or privilege, to add or delete entries from the discretionary access control table.

Mandatory Access Control

For a subject to access the final entity of a reference, the subject container clearance must dominate the security level calculated for the reference to the entity. Other mandatory access control features of this model are standard in nature: properties similar to the simple security property and '*' properties of the Bell-Lapadula model are included. The mandatory requirements to entities are:

- Container Reference

The subject container clearance dominates the security level for the reference to the entity.

- Read Access

The subject must have container reference to the entity and the subject data clearance must dominate the entity security level.

- Write Access

The subject must have container reference to the entity and the entity security level must dominate the subject data clearance.

2.3 The MLO Model and the Requirements for A1 Certification

In addition to providing a model which can be readily used for formal verification, addresses multilevel objects, and addresses some of the issues which have made other models overly restrictive, the Multilevel Object Security Model is designed to meet the model requirements for A1 certification as described in the Department of Defense Trusted Computer System Evaluation Criteria [1]. These requirements are as follows:

- The model must be a precise mathematical statement of a security policy
- The model must describe the initial state of the system
- The model must describe how the system progresses from one state to another
- The model must define a secure state
- The model must be sufficient to support the security policy
- The model must be supported by a formal proof that the model supports its axioms

The MLO model has been formally specified and a verification that the model meets its axioms has been performed. The following paragraphs describe in detail the implementation of the above listed requirements.

2.3.1 The Formalization of the Model

The first four requirements are met by the formalization of the MLO model. The formalization itself is a precise mathematical description of the MLO policy stated above. The nature of the specification language facilitates the definition of an initial state, a description of state transition, and the definition of a secure state.

The MLO model is written in SYSPECIAL, a dialect of the language SPECIAL of the Hierarchical Development Methodology, HDM. A full version of the formal specification can be seen in Appendix A of this document. SYSPECIAL uses the bulk of constructs available to SPECIAL. Constructs with ambiguous semantics have been eliminated. The resulting subset of SPECIAL was then augmented with a built-in constructor for describing sequences and several operators on sequences. Details about SYSPECIAL can be seen in [4].

The system state is completely described by the values of the VFUNs declared in the specification. The value of a VFUN can be altered by specification operations. The current values of the VFUNs define the system state at any time.

The type of a VFUN and the possible values for that type are specified. The initial value of a VFUN must be one of the possible values of the type. The language allows the initial value of a VFUN to be further constrained or a particular initial value to be specified. The language requires that any changes in the value of a VFUN as the result of an operation stay within the possible values of the VFUN's type.

Allowable state transitions are described by operations, which are specified using a construct called an OFUN or an OVFUN. These operations describe preconditions and postconditions of VFUNs. The postconditions of OFUNs can describe new values assigned to VFUNs. The postconditions of OVFUNS can describe new values assigned to VFUNs and values returned to the caller. Because these operations describe altered values of VFUNs, they describe how the system progresses from one state to another. The sum of all the operations defines all the possible kinds of state transitions of the system. The system can only progress from one state to another if an operation has been explicitly specified that would accomplish that change in the values of the VFUNs.

A secure state for the system is completely defined by the ASSERTIONS paragraph of the specification. The assertions in this paragraph describe conditions whose truth implies that the system is in a secure state. The assertions paragraph of the specification allows three types of statements. First are parameter assertions which state properties of what are called "parameters" of the specification but do not directly help to specify the secure state. The other two kinds of assertions describe conditions involving VFUN values. The second kind consists of invariant assertions -- a description of conditions which must be true at any time. For example, the security level hierarchy rule is stated as an invariant assertion. The third set of assertions are called constraints and describe conditions which must be true for each transition between states. The mandatory and discretionary access properties are described as constraints. A secure state is one for which all the invariant assertions are true and which either is the initial state or was reached from another secure state by a transition obeying all the constraint assertions.

That the model is sufficient to describe the security policy can be seen by careful inspection of the invariant and constraint assertions.

2.3.2 The Formal Proof that the MLO Model is Internally Consistent

The final requirement is to show that the initial state of the model and each possible subsequent state meets the axioms of the model. The formalization has been processed by the MUSE verification system. Output from the proof process can be seen in [3]. This system generates formulas for proving that any system implementing the model maintains a secure state according to the assertions, or axioms, of the model.

Two kinds of formulas are generated. The first is generated for the initial state. It states that the invariant assertions are all true for any initial state satisfying the conditions specified for the initial values of the VFUNs. The second kind of formula deals with an operation. One of these formulas is generated for each operation. The formula states that every invariant assertion is true after the operation if it was true before the operation and that every constraint assertion is true for any state transition that the operation could carry out.

An induction argument is used to illustrate that invariant assertions are true for the MLO model. If the assertion is true in the initial state and in every possible subsequent state as described by the model operations, then the model adheres to the assertion. Each invariant assertion must be shown to be a system invariant: it must be shown to be true in the initial state of the system and that it remains true after every event or operation in the system takes place.

Given the foregoing concepts, it is possible to apply a basic paradigm of verification. To show that some formula P is a system invariant, simply proceed as follows:

- Show that P is true in the initial system state.

- Let S be any state for which P is true. Show that if any operation is invoked in state S, resulting in a new state S', THEN P is also true in S'.

Upon completion of such a proof, P will have been established as an invariant, that is, P will be true in every state. This result follows by induction on the number of operation invocations (or state transitions).

The proof of the first kind of formula shows that each invariant assertion is true for any possible initial state. The proofs of the formulas of the second kind show that each invariant assertion is true for the resulting state (S') assuming it was true for the starting state (S). Taken together, the proofs of the formulas show that the invariant assertions are actually system invariants as their name implies.

To prove that the model obeys a constraint, one must simply show that every operation obeys the constraint. The proof of a formula of the second kind shows that an operation obeys all the constraint assertions. Taken together, the proofs of all those formulas show that the model as a whole obeys the constraints.

In summary, since the specified operations describe all the possible state transitions and the invariant and constraint assertions are the axioms that define the secure state, proving all the formulas is also a proof that the model is secure according to its axioms.

The formulas were processed by the MUSE theorem prover in order to discover proofs and check the proofs for validity.

Extensive use of automated tool support was used to verify the MLO model. The SYSP tool, designed to perform syntax and semantic checking on a SYSPECIAL specification, is similar to the SRI standard HDM tool, CMO. The SYFG tool is the MUSE formula generator, and SYTP is the theorem prover which can be used to attempt proof of the formulas generated by SYFG.

Figure 7 shows the tools to be employed and the flow of information between them.

```
         +--------+      +---------+symtab    +--------+ proved
 model   |        |      |         |---------->|        |-------->
-------->| SYSP   |----->|   SYFG   |formulas  | SYTP   | unproved
         |        |      |         |---------->|        |-------->
         +--------+      +---------+          +--------+
```

Figure 3. Tools for Model Verification.

SYTP yields validated proofs with any errors noted The steps illustrated by the above diagram are as follows:

- The model in SYSPECIAL is input to SYSP

- SYSP yields parsed model in an internal form 1

- Parsed internal form of model is input to SYFG

- SYFG yields formulas as described above and a symbol table

- The symbol table and formulas are input to SYTP

- SYTP yields validated proofs with any errors noted

THE MLO MODEL--TASK 1

Detailed information about the tools used can be seen in [4].

THE MLO MODEL--TASK 2

AN ABSTRACT IMPLEMENTATION OF THE MLO MODEL

Teresa F. Lunt

Sytek, Incorporated

3. TASK_2:__An_Abstract_Implementation_of_the_Model

In order to demonstrate that the model developed in Task 1 provides an adequate mechanism for implementing a military message system of the type described by the NRL model [2], we will develop an abstract implementation of the model developed in Task 1 to describe the NRL message system architecture. For this abstract implementation, each of the objects of the NRL model will be represented by some expression using objects of the Task 1 model, and the operations of the NRL model will be represented by combinations of operations in the Task 1 model. We will analyze the abstract implementation to determine whether additional restrictions need to be placed on the military message system because of the use of the model described in Task 1. We will also determine what, if any, trust needs to be placed in the software implementing the NRL military message system in terms of the Task 1 model.

In order to avoid having to model and verify the entire user interface for the NRL military message system, the abstract implementation will encompass only the security-critical portions of the NRL message system.

Our approach is to provide the functionality of the NRL model to the user in a much simpler manner than is provided for by the NRL data structures and operations. The use of the Task 1 model allows us to greatly simplify the Landwehr data structures and thus to provide the same functionality with far fewer operations. This simplicity makes for a conceptually cleaner model. The abstract implementation will translate the Landwehr operations on messages into the operations of the Task 1 model.

3.1 Description_of_the_Abstract_System

The system we will implement in terms of the Task 1 model is a military message system that will include users acting in a number of different roles (for example message drafter, security administrator, etc.), with a variety of clearances, and with permission to exercise a particular function for a particular message or other data item depending on a wide variety of discretionary access policies. Below we discuss this system in detail in terms of its data structures and operations, user roles, and discretionary access policies.

3.1.1 Data Structure

Here we discuss in general terms the organization of the data to be contained in the database.

3.1.1.1 Messages

The database will contain message fields and associate them with the message to which they belong. To identify a message, we will use a keyfield, or unique identifying number, we will call the message ID. For each message ID record, the database will contain a number of records, one for each message field associated with the message. Each such dependent record will contain a keyfield indicating the position in the message, the field contents, and the classification.

Several message types, each with a corresponding set of standard message fields, could be accommodated. In this case, the message ID record would include an indicator identifying the message type.

A typical message might consist of the following message fields:

- originator
- action addressee
- information addressees
- date-time group
- subject
- paragraphs of text
- precedence
- annotations

The message classification must be at least as high as the highest classification of the message fields. The message classification is not a message field, but appears in the message ID record.

The message ID record will also contain an indicator for the "message status." A message status might be "available only to the drafter;" "coordination copies sent, awaiting coordination action;" "message coordinated, awaiting release;" "message released;" "formal message sent." Including such a status indicator in the message ID record eliminates the need for the system message files and the various user message files of the Landwehr model. One advantage of a DBMS is the reduction of data redundancy. Our DBMS-based data structure avoids the data redundancy of the Landwehr model by eliminating the need for citations, filters, citation files, filter files, system message files, etc., and all the attendant operations on those object types. With a much smaller, cleaner, set of operations, our approach provides the same functionality as the Landwehr model.

3.1.1.2 Message_Templates

Standard message templates will be included with the system. Each will have a classification and an access control list (of groups who can use it). In addition to these will be user-created templates, with a classification, and available only to the user who created them. Templates will be stored in the database in the same manner as are messages, and will be referred to by name (analogous to the message ID for a message). With the template name will be stored an indicator as to whether the template is a standard system template, and if not, which user the template belongs to. This indicator will be used for enforcing discretionary access control to the templates. Using such an indicator in the database allows a simplification of Landwehr's files of system-owned templates, files of user-owned templates, and all the attendant operations on these files.

3.1.1.3 System_Tables

Included in the database will be tables containing security-related information available only to the security administrator and system-related information available only to the system administrator. These are:

- A table associating user roles with the operations available to users acting in those roles. This table can only be accessed by the system administrator.

- A table associating user names with user groups (user groups are those designators used for message originators and addressees). This table can only be accessed by the system administrator.

- A table associating user names with their clearances. This table can only be accessed by the security administrator.

- A table associating user names with the user roles permitted for that user. This table can only be accessed by the security administrator.

These tables will be collectively referred to as the system database.

3.1.2 User-Level Operations

A number of operations will be available to the user, with appropriate discretionary access controls applied to restrict certain operations to a certain type of user (user-role) or group of users. These operations will not all consist of trusted software, but will make use of the secure Task 1 model operations. In other words, the user-level operations will be constructed from Task 1 model operations.

The system is required to allow users to handle data of several classifications, up to their clearance level, during a single computer session. The applications-level operations will operate in such a way as to allow a user to operate on a multilevel object, such as editing a message, while the trusted software will actually implement this as several single-level transactions against the database.

3.1.2.1 Operations on Messages

A number of operations on messages will be available to the user, with appropriate discretionary access controls applied to restrict certain message operations to a certain type of user or to a message originator or addressee. These operations are briefly described below.

- Create Message (without using a message template; see the "get template" function listed under "operations on message templates" to create a new message using a message template)

- Edit Message (modifies the as yet un-released message; within this function the individual message fields will be edited separately, one at a time)

- Copy and Edit Message (creates a new message by copying and editing an existing message; within this function the individual message fields will be edited separately, one at a time)

- Archive Message(s); Un-Archive Message(s). Options might include specifying a dtg window, classification, search keyword for the subject field, or list of originators, or sort instructions. Or a specific message(s) can be archived or unarchived by indicating a (list of) message ID(s).

- Display Message

- Reply (to create a message in reply to another message)

- Forward Message with annotations (for information only; doesn't require release approval)

- Readdress Message (new action addressee; requires release approval)

- Coordinate Message (to send a copy of a proposed message for comment)

- Chop Message, with annotations (return to originator, with comment and/or chop action)

- Approve Message for Release

- Send Message (requires release approval)

- Downgrade Message (allows the security administrator to change the classification of a message by changing the classifications of the relevant message fields)

3.1.2.2 Operations_on_the_Message_Database

A "Scan Messages" operation will be available to users to retrieve information about the message database. This operation displays the message ID, originator, date-time-group (dtg), classification, and subject message fields for messages for which the user is an addressee and is cleared. Options will include specifying a dtg window, classification, search keyword for the subject field, or list of originators, or sort instructions.

The "scan messages" operation eliminates the need for the data objects "message citations," files of message citations, "filters," and files of filters, and all the attendant operations, of the Landwehr model. Allowing selective retrieval on keywords in the subject field eliminates the need for operations for assigning, retrieving, and de-assigning keywords, as in the Landwehr model.

3.1.2.3 Operations_on_Message_Templates

Standard message templates will be associated with a classification and an access control list (of groups who can use it). In addition to these will be user-created templates, with a classification, and available only to the user who created them. Templates will be stored in the database in the same manner as are messages, and will be referred to by name (analogous to the message ID for a message). Below we briefly describe the operations on message templates available to the user.

- Get Template (to create a message using the named template)

- Show Template (to display the named template)

- Scan Templates (lists templates available to that user)

- Create Template

- Remove Template (for user-created templates)

- Edit Template (modifies the named user-created template)

- Copy and Edit Template (creates a new template by copying and editing an existing template)

- Rename Template (renames a user-created template)

3.1.2.4 Operations on the System Database

Operations will be provided to the system administrator and the security administrator for the purpose of maintaining the system database. The system administrator will be able to read, add, delete, and change records in the tables associating user roles with available operations and associating user names with user groups. The security administrator will be able to read, add, delete, and change records in the tables associating user names with clearances and associating user names with user roles. The system administrator and security administrator will not be permitted to directly invoke operations on the database, but will be provided with a specific set of operations, such as "change user clearance", that will be translated into basic Task 1 model operation. The abstract implementation will enumerate these operations and their translations. The system databases will also be referenced for user and device authentication.

3.1.3 User Roles

In a military message system, there are various types of users with differing responsibilities, functions, and privileges. These user roles are part of the discretionary access control policy for the system. The concept of user roles in the discretionary access control policy, as opposed to the concept of privileges given to specific individuals, will be reflected in the model as a restriction on the functions that will be available to an individual acting in a specific user role. These restrictions are in addition to any restrictions that may be imposed on an individual's access to a particular data object. It is important to note that a user may act in one role with respect to a particular message and in another role with respect to another message; for example, a given user may act as a coordinator for one message (that is, asked to agree, disagree, or provide comment) and may act as drafter for other messages.

The user roles drafter, coordinator, releaser, system administrator, and security administrator, and the message operations permitted to users acting in those roles, are described below.

3.1.3.1 Drafter

The drafter of a message is the user who initially creates and prepares a message for coordination and release signature. The message operations available to the drafter are:

- create message
- edit message
- copy and edit message
- display message
- reply
- forward message
- coordinate message

3.1.3.2 Coordinator

The message coordinator is one asked to provide comment, to concur or not concur on a message sent to that user's agency for coordination. The coordinator's action, or chop, is the decision indicated in the message annotations. The message operations available to the coordinator are:.

- display message
- forward message
- chop message

3.1.3.3 Releaser

The message releaser is a person authorized to approve the sending of the message as an official communication from the originating agency. The message operations available to the releaser are, in addition to those available to the drafter:

- readdress message
- approve message for release
- send message

3.1.3.4 System_Administrator

The system administrator is assigned responsibility for the administration of the system. The message operations that will be available to the system administrator, in addition to those available to the drafter, are:

- readdress message
- archive message
- unarchive message

3.1.3.5 Security Administrator

The security administrator has the responsibility for associating users with clearances and roles, and can in addition to the message operations available to a drafter, use the "downgrade message" function.

3.1.4 Discretionary Access Controls

Discretionary access controls will be applied in several ways. For messages, message fields for a message will include the originator, action addressee, and information addressee. The users associated with these user groups will be contained in the system database. For a delivered message, these constitute the access control list for the message. For an as-yet undelivered message, access will be limited to, as appropriate, the drafter, coordinator(s), and releaser. Indicator(s) as to which user(s) are in these roles relative to the message will be included in the record for the message ID. The message status indicator indicates whether a message is delivered. For user roles, the users permitted to assume the various user roles will be contained in the system database, as well as the specific operations allowed for each user role.

3.1.5 Mandatory Access Controls

The military message system will allow users to handle information of several classifications simultaneously. The system will allow a user access to a message classified up to the user's clearance level. A message, however, is a multi-level object, containing message fields of various classifications less than or equal to the message classification. Trusted software will compute the message classification from the classifications of the message fields and will compare this to the user's clearance level contained in the system database to allow or deny the user access to the message. This prevents "reading up," or a user's having read access to information classified higher than the user's clearance level. This also prevents "writing down," since user operations on messages can result in only add, delete, or replace (delete and add) operations for message field records. That is, changes to message field records in the database will be prohibited. A user's change to a message field will be implemented by deleting the old message field record and adding, or replacing it with, a new one, with the classification specified by the user. These restrictions prevent writing down.

3.2 Constructing the Message System Data Structures From the Task 1 Model Data Structures

The message system data structures will be constructed from the data structures of the Task 1 model. Below we describe how the message system data structures will be mapped to the Task 1 data structures.

3.2.1 Messages

A message will be represented in terms of the Task 1 model data structures by a container, corresponding to the message ID record. The message fields will be represented by atoms. The container value will include the entity IDs of the atoms representing the message fields, an indicator for the message type, the message classification, and the message status. The container entity ID will be uniquely mappable to the message ID.

Atoms representing message fields will belong to only one message in the system. This is because military messages are formal documents, and we can't allow a situation in which changing a paragraph of text in one message results in the same change in an identical paragraph in another message. In general, users will have different privileges with respect to different messages. Moreover, messages that have been approved for release or have been delivered can no longer be changed.

3.2.2 Message Templates

A message template will be represented in terms of the Task 1 model data structures by a container, in much the same manner as a message. The template fields will be represented by atoms. These fields will be either empty or fixed. The value of the container will include the template classification. It will also include an indicator that it is a template (rather than a message). The container entity ID will be uniquely mappable to the template name.

The system tables will associate the owner user with the template name and will also indicate whether the template is system owned or user owned. The system tables will also associate an access control list with system-owned templates.

3.2.3 System Tables

The system tables will be represented in terms of the Task 1 model data structures by means of the boolean discretionary access control tables.

3.3 Building the Applications-Level Operations from the Task 1 Model Operations

User operations on messages, message templates, the message database, and the system database will be built from the basic operations of the Task 1 model. When a user invokes a user operation requiring a query or transaction against the database, this will be translated to one or a sequence of Task 1 model operations on the database.

Below we describe how each user operation will be constructed from the Task 1 model operations.

- Create Message: The "create message" operation can be constructed from the Task 1 model operations as follows: The create_entity operation is invoked to create a new message_ID record. The upgrade (or downgrade) operation will be used to adjust the security level of the new record to be the message classification "seclevel" supplied by the user. Then for each message field, the create_entity operation is invoked to create a new record for the message field, and the upgrade (or downgrade) operation will be used to set the message field classification "seclevel" supplied individually for each message field by the user; then the write operation is invoked for the new message field to put the user-supplied value into the message field; and the write operation is invoked insert the new message field into the message ID record.

- Edit Message: The "edit message" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables (system database) are referenced to ensure that the user is the drafter for the message. The read operation is invoked for the message associated with the user-specified message ID, and for each message field identified in the message ID record, thus retrieving all message fields corresponding to the message. As the user edits the individual message fields, the write operation is invoked to modify the individual message fields, the destroy operation is invoked for message fields to be deleted, the create, write (to update) and write (to insert) operations are invoked for message fields to be added. If modifying a message field would result in the message field having a new classification, then the model will force this to be accomplished by destroying the old message field and then creating a new one at the new security level.

- Copy and Edit Message: The "copy and edit message" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables (system database) are referenced to ensure that the user has read permission for the message being copied. The create_container operation is invoked to create a new message_ID record at the same message classification "seclevel" as the message being copied. The read operation is invoked to read the container record for the message being copied, and the read operation is invoked for each message field to retrieve all message fields corresponding to that message. Then for each message field, the create operation is invoked to create a new record for the message field for the new message, with the message field classification matching that for the corresponding message field in the message being copied; then the write (to update) operation is invoked for each message field to copy the value of the message field from the corresponding field of the message being copied; and the write (to insert) operation is invoked to associate each new message field with the message ID record for the new message. At this point a new message has been created, which is a duplicate of the message being copied. As the user edits the individual message fields of the new message, the write (to update) operation is invoked to modify the individual message fields, the destroy operation is invoked for message fields to be deleted, the create, write (to update) and write (to insert) operations are invoked for message fields to be added. If modifying a message field would result in the message field having a new classification, then the model will force this to be accomplished by destroying the old message field and then creating a new one at the new security level.

- Archive Message(s): The "archive message(s)" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables (system database) are referenced to ensure that the user has permission to archive the indicated message(s). For each message to be archived, the read operation is invoked for the message associated with the user-specified message ID, and the read operation is invoked for each message field to retrieve all message fields corresponding to the message. After the message has been copied to archive, the destroy operation is invoked for each message field, and the destroy operation is invoked for the message ID.

- Unarchive Message(s): The "unarchive message(s)" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables (system database) are referenced to ensure that the user has permission to unarchive the indicated message(s). For each message being unarchived, the create operation is invoked to create a new message_ID record. The upgrade (or downgrade) operation is invoked to adjust the message classification to the classification "seclevel" of the message being copied from archive. Then for each message field, the create operation is invoked to create a new record for the message field for the new message, and the upgrade (or downgrade) operation is used to adjust the message field classification to match that for the corresponding message field in the message being copied from archive; then the write(to update) operation is invoked for each message field to copy the value of the message field from the corresponding field of the message being copied from archive; and the write (to insert) operation is invoked to associate each new message field with the message ID record for the new message.

- Display Message: The "display message" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables (system database) are referenced to ensure that the user has read permission for the message. The read operation is invoked for the message associated with the user-specified message ID, and the read operation is invoked for each message field to retrieve all message fields corresponding to the message.

- Reply: The "reply" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the message for which a reply is being generated, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission for the message for which a reply is to be created. Then the create operation is invoked to create a new message_ID record, and the upgrade (or downgrade) operation is invoked to adjust the classification to the classification "seclevel" supplied by the user. "Seclevel" must be at least as high as the classification of the message being replied to. The reply operation copies certain information from the original message into the new message, for example, information for the "originator", "action addressee", "subject" and "information addressees" message fields. For each such message field, the create operation is invoked to create a new record for the message field, and the upgrade (or downgrade) operation is used to adjust the message field classification to the classification "seclevel" obtained for each such message field from the corresponding message field of the original message. Then for each such message field, the read operation is invoked to obtain the contents of the corresponding message field. Then for each such message field, the write (to update) operation is invoked to put the contents of the corresponding message field into the new message field. The write (to insert) operation is invoked to associate the new message field with the message ID record.

  Then for each user-supplied message field, the create operation is invoked to create a new record for the message field, and upgrade (or downgrade) is called to adjust the the message field to the classification "seclevel" supplied individually for each message field by the user. Then the write (to update) operation is invoked for the new message field to put the user-supplied value into the message field; and the write (to insert) operation is invoked to associate the new message field with the message ID record.

- Forward Message with annotations: The "forward message" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the message being forwarded, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission for the original message. The create operation (and upgrade or downgrade) are invoked to create a new message_ID record at the same message classification "seclevel" as the message being copied. The read operation is invoked to read the container record for the message being copied, and the read operation is invoked for each message field to retrieve all message fields corresponding to that message. Then for each message field, the create operation (and upgrade or downgrade) are invoked to create a new record for the message field for the new message, with the message field classification matching that for the corresponding message field in the message being copied; then the write (to update) operation is invoked for each message field to copy the value of the message field from the corresponding field of the message being copied; and the write (to insert) operation is invoked to associate each new message field with the message ID record for the new message. At this point a new message has been created, which is a duplicate of the message being copied. In order to append annotations, the create, write (to update), and write (to insert) operations are invoked. The annotations are considered as additional message fields in the forwarded message.

  The original originator, action addressee, and information addressees message fields are included as paragraphs of text in the message to be forwarded, using the create, write (to update), and write (to insert) operations, and the user supplies new originator and information addressees message fields (there is no new action addressee) by means of the write (to update) operation.

- Readdress Message: The "readdress message" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the message being readdressed, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission for the original message. The create operation (and upgrade or downgrade) are invoked to create a new message_ID record at the same message classification "seclevel" as the message being copied. The read operation is invoked to read the container record for the message being copied, and the read operation is invoked for each message field to retrieve all message fields corresponding to that message. Then for each message field, the create (and upgrade or downgrade) operations are invoked to create a new record for the message field for the new message, with the message field classification matching that for the corresponding message field in the message being copied; then the write (to update) operation is invoked for each message field to copy the value of the message field from the corresponding field of the message being copied; and the write (to insert) operation is invoked to associate each new message field with the message ID record for the new message. At this point a new message has been created, which is a duplicate of the message being copied.

The write (to update) operation is invoked to substitute the new user-specified action addressee. The change_status operation is invoked to set a flag in the message ID record to require release approval.

- Coordinate Message: The "coordinate message" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the message to be coordinated, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission for the original message. The create (and upgrade or downgrade) operations are invoked to create a new message_ID record at the same message classification "seclevel" as the message being copied. The read operation, is invoked to read the container record for the message being copied, and the read operation is invoked for each message field to retrieve all message fields corresponding to that message. Then for each message field, the create (and upgrade or downgrade) operations are invoked to create a new record for the message field for the new message, with the message field classification matching that for the corresponding message field in the message being copied; then the write (to update) operation is invoked for each message field to copy the value of the message field from the corresponding field of the message being copied; and the write (to insert) operation is invoked to associate each new message field with the message ID record for the new message. At this point a new message has been created, which is a duplicate of the message being copied. The create, write (to update), and write (to insert) operations are invoked to append coordination instructions.

  The original originator, action addressee, and information addressees message fields are included as paragraphs of text in the message to be forwarded, using the create, write (to update), and write (to insert) operations, and the user supplies new originator and addressee message fields by means of the write (to update) operation.

- Chop Message: The "chop message" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the message being chopped, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission and permission to chop for the message. The create (upgrade or downgrade) operations are invoked to create a new message_ID record at the same message classification "seclevel" as the message being copied. The read operation is invoked to read the container record for the message being copied, and the read operation is invoked for each message field to retrieve all message fields corresponding to that message. Then for each message field, the create operation is invoked to create a new record for the message field for the new message, with the message field classification matching that for the corresponding message field in the message being copied; then the write (to update) operation is invoked for each message field to copy the value of the message field from the corresponding field of the message being copied; and the write (to insert) operation is invoked to associate each new message field with the message ID record for the new message. At this point a new message has been created, which is a duplicate of the message being copied. The create, write (to update), and write (to insert) operations are invoked to append comments and/or chop action.

  The original originator, action addressee, and information addressees message fields are included as paragraphs of text in the message to be forwarded, using the create, write (to update), and write (to insert) operations, and the user supplies new originator and addressee message fields by means of the write (to update) operation.

- Approve Message for Release: The "approve message for release" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to read the container record for the message and obtain the classification of the message, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission and release authority for the message. The read operation is invoked to read the container record for the message, and the read operation is invoked for each message field to retrieve all message fields corresponding to that message. If the user indicates release approval, the write_status operation is invoked to change the message status indicator in the message ID record.

THE MLO MODEL--TASK 2

- Send Message: The "send message" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to read the container record for the message and obtain the classification of the message to be sent, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission and permission to send the message. The read operation is invoked to read the message ID record to check the "approved for release" flag. The write_status operation is invoked to set a flag in the message ID record to indicate the message has been sent. The read operation is invoked to read the container record for the user-specified message ID, and the read operation is invoked for each message field to retrieve all message fields corresponding to the message.

- Downgrade Message: The "downgrade message" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the message being downgraded, to ensure the user is cleared to read it. The discretionary access control tables (system database) are referenced to ensure that the user has read permission for the message and is acting as security administrator. The read operation is invoked for the message associated with the user-specified message ID, and the read operation is invoked for each message field to retrieve all message fields corresponding to the message. The user can indicate a change of classification for individual message fields by destroying the old message field and then creating a new one at the new security level. The read operation is invoked to retrieve the contents of the message field. The create operation is invoked to create the new message field at the new security level. The write (to update) operation is invoked to copy the contents of the old message field to the new message field. The destroy operation is invoked to remove the old message field. The write (to insert) operation is invoked to include the new message field in the message.

- Scan Messages: The "scan messages" operation can be constructed from the Task 1 model operations as follows: The user can generate a table of contents for the database outside the security enforcing mechanism. He can program a filter to restrict data he sees to a certain date or other constraint. He issues reads on each entity in the table of contents, and those to which the has proper access will be returned to him. He can then process this data with his filtering program. If he attempts access of data entities to which he does not have access, or which contain empty entities, no information will be returned to him. In this way he can only access the parts of the database to which he has the proper accesses. He can restrict the information further.

- Get Template: The "get template" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the template, to ensure the user is cleared to read it. The discretionary access control tables are referenced to ensure that the user has read permission for the template. The read operation is invoked to retrieve the record for the indicated named template. The read operation is invoked for each template message field to retrieve all fixed and blank message fields. The create operation is invoked to create a new message_ID record, including the message classification "seclevel" supplied by the user. Then for each message field indicated in the template, the create operation is invoked to create a new record for the message field, including the message field classification "seclevel" supplied individually for each message field by the user; then the write (to update) operation is invoked for the new message field to put the user-supplied value into the message field; and the write (to insert) operation is invoked to associate the new message field with the message ID record. If the template indicates fixed values for certain message fields, the write (to update) operation is invoked to copy those values from the template description into the appropriate message fields.

- Show Template: The "show template" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the template to ensure the user is cleared to read it. The discretionary access control tables are referenced to ensure that the user has read permission for the template. The read operation is invoked to retrieve the record for the indicated named template. The read operation is invoked for each template message field to retrieve all fixed and blank message fields.

- Scan Templates: The "scan templates" operation can be constructed from the Task 1 model operations as follows: The discretionary access tables are referenced to obtain the names of the templates available to that user. The read operation is invoked for each template available to that user to retrieve the template record. Template names and descriptive information from the template records are used to display to the user the templates available to him or her.

- Create Template: The "create template" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables are referenced to ensure that the user has permission to use the "create template" operation. The create operation is invoked to create a record for the new template. The create operation is invoked to create each template message field. The write (to update) operation is invoked for the user to specify information for the fixed message fields; all others remain blank. The write (to insert) operation is called for each template message field to include them as fields of the template.

- Remove Template: The "remove template" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables are referenced to ensure that the user has permission to remove the template. The destroy operation is invoked for each template message field, and the destroy operation is invoked for the template.

- Edit Template: The "edit template" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the template, to ensure the user is cleared to read it. The discretionary access control tables are referenced to ensure that the user has read permission for the template. The read operation is invoked to retrieve the record for the indicated named template, and the read operation is invoked to retrieve each template message field. As the user edits the individual template message fields, the write (to update) operation is invoked to modify the individual fields, the destroy operation is invoked for fields to be deleted, and the create, write (to update), and write (to insert) operations are invoked for fields to be added. If modifying a field would result in the field having a new classification, then the model will force this to be accomplished by destroying the old field and then creating a new one at the new security level.

- Copy and Edit Template: The "copy and edit template" operation can be constructed from the Task 1 model operations as follows: The read operation is invoked to obtain the classification of the template, to ensure the user is cleared to read it. The discretionary access control tables are referenced to ensure that the user has read permission and permission to edit the template. The read operation is invoked to retrieve the record for the indicated named template, and the read operation is invoked to retrieve each template message field. The create operation is invoked to create a record for the new message template at the same classification "seclevel" as the template being copied. Then for each template message field, the create operation is invoked to create a new field for the new template, with the classification matching that for the corresponding field in the template being copied; then the write (to update) operation is invoked for each field to copy the contents of the field from the corresponding field of the template being copied; and the write (to insert) operation is invoked to associate each new field with the new template. At this point a new template has been created, which is a duplicate of the template being copied. As the user edits the individual fields of the new template, the write (to update) operation is invoked to modify the individual fields, the destroy operation is invoked for fields to be deleted, the create, write (to update), and write (to insert) operations are invoked for fields to be added. If modifying a field would result in its having a new classification, then the model will force this to be accomplished by destroying the old field and then creating a new one at the new security level.

- Rename Template: The "rename template" operation can be constructed from the Task 1 model operations as follows: The discretionary access control tables are referenced to ensure that the user has permission to rename the template.

  Since the template name is actually the entity reference for the container, in terms of the Task 1 model, a new container with a new name, or entity reference, has be created, the atoms belonging to the original template have to be associated with the new template and disassociated with the original template, and then the old template has to be destroyed.

  The read operation is invoked to retrieve the record for the template to be renamed. The create operation is invoked to create a record for the new message template identical to and at the same classification "seclevel" as the original template, but with the new name as reference. Then the write (to insert) operation is invoked for each field of the original template to associate the fields of the original template with the new container record for the template. Then for each template field, the write (to remove) operation is invoked to disassociate the template fields from the original template container record. Then the destroy operation is invoked to delete the container record for the original template.

- Operations on the system database: The delete_discretionary_access and add_discretionary_access operations are used to maintain the system tables by authorized users.

THE MLO MODEL--TASK 3

AN ANALYSIS OF THE DIAMOND DOCUMENT HANDLING SYSTEM

Teresa F. Lunt

Elisabeth C. Sullivan

4. Task 3: A Security Mechanism for the The Diamond System

The first two tasks of this program are directed toward the development of a general model for systems of multilevel objects, and its application to a military message system model. This last task applies this work to the Diamond distributed multimedia document system. For this task, we will develop an abstract implementation of the model developed in Task 1 to describe the the Diamond distributed multimedia document system and will propose a security mechanism to be implemented within the Diamond architecture.

The Diamond system is a prototype system being developed by Bolt Beranek and Newman, Inc., for the Defense Advanced Research Projects Agency and Rome Air Development Center. The Diamond distributed architecture, and use of personal computers for advanced user interfaces. In addition, multilevel security is a goal. In order to provide demonstrable security and provide the complex functionality required, it is necessary to analyze the Diamond architecture with respect to security so that the security mechanism can be isolated from the rest of the system. This task will use the model developed in the earlier tasks to analyze the Diamond system.

The first step in analyzing multilevel security in the Diamond system will be the definition of the user level requirements for security in Diamond. To do this, we will develop an abstract implementation of the Task 1 model describing the functional requirements and constraints to be satisfied by a security mechanism for Diamond. For this abstract implementation, each of the objects of the Diamond system will be represented by some expression using objects of the Task 1 model, and the Diamond operations will be represented by combinations of operations in the Task 1 model.

As part of this task we will also analyze the Diamond architecture to determine how a security mechanism for Diamond can be developed so as to make it as simple as possible and isolated from other Diamond software. Our architectural analysis will include investigation of the use of special purpose devices such as security markers and filters to allow the separation of the security mechanism from other Diamond software. This is particularly important in the case of Diamond because of the complexity of the software necessary to meet the state-of-the-art goals of the Diamond program. The result of this task will be a candidate architecture for security enforcement in future versions of the Diamond system, based on an implementation of the Task 1 model in terms of the Diamond functionality and constraints.

4.1 Description of the Diamond System

The Diamond system is basically a document handling system with support for sending documents as messages. It is designed to operate in a distributed environment, called a cluster, with internet communications capability so that users of the cluster can send and receive messages from non-cluster users. The objects handled by Diamond will be multimedia, consisting of text, graphics, voice, and image data.

Diamond is partitioned logically into three parts: the Diamond core, the user interfaces, and a collection of additional capabilities for enhanced document and message services. The user interfaces will reside on personal computers and on host computers, and will include tools to compose and edit multimedia messages. We will concern ourselves primarily with the Diamond core, since the other parts are largely individual tailorings and enhancements that make use of core services.

The Diamond core is composed of a document store, a registry (of users and groups, local and remote), an authentication service, a device control service (e.g. for printers), and an internet message processing module (MPM).

The document store contains the Diamond documents. The registry contains information about users and is referenced for purposes of authentication and access control. The message processing module (MPM) supports message communication with users external to the cluster. Below we discuss this system in detail in terms of its data structures and operations.

4.1.1 Data Structure

The Diamond system has five types of data objects: documents, folders, the registry, user descriptions, and group descriptions. All objects belong to either a folder or the Registry. A citation is not an object but a reference to an object, and doesn't have a UID.

Each of the five object types can be annotated. For documents, an annotation is associated with a point within the document. For other object types, an annotation is associated with the object as a whole. Since different users can comment on the same object, the annotations are stored as an ordered list associated with the object. The annotations may contain text, graphics, image, and voice data.

The document store holds one copy of every document. When a document is sent to a list of users, they each receive a citation to that single copy, not a separate copy of the document. Once an object has been entered into the document store, it cannot be modified, with the exception of folders. Editing an object results in a new object.

Objects in the Diamond system are represented by the following standard set of information:

(UID, Value, UIDRefList, AnnotationList, AccessControlInfo, RefCount)

where

UID is the unique object identifier (includes the object type)

Value is the contents of the object. It is necessary to know the object type to interpret its value.

UIDRefList is a list of the UIDs of objects (if any) referenced within the value of the object (e.g. folders refer to documents and folders)

AnnotationList is a list of annotations to the value of the object. Annotations are considered additions rather than changes to the object and are, in fact, separate objects themselves. An annotation is a list of the form:

(ReferencePoint, AnnotationUID)

where ReferencePoint specifies a point within the object being annotated and AnnotationUID is the identifier of the object that annotates this object.

AccessControlInfo includes an access control list (ACL) and other security information

RefCount is the count of outstanding references to the object. References to objects can appear in citations, UIDRefLists and AnnotationLists. The RefCount is updated whenever a reference is created or destroyed. When the RefCount is zero, the document is deleted from the document store.

Access control lists (ACLs) are used for access control, listing users and groups. Access to all objects is controlled by ACLs. An ACL is an ordered pair of the form (authority, access rights). An authority is a principal (user or diamond component) or group. Access rights are modes of access permitted to the object for the given authority. A separate ACL is associated with each object. A security check is made to ensure that the authority attempting to perform an operation on an object appears on the ACL for the object and that the rights associated with the authority include the right required to perform the requested operation.

A group is a list of principals (users) and/or other groups, and has a symbolic name. Groups can be used in ACLs and distribution lists. A group can include the internet mailbox names (symbolic names) of non-cluster users; these are ignored when the group is used in an ACL. The symbolic names of non-cluster users all map to a single "special" user description for non-cluster user (for access control and authentication). The symbolic names, however, are meaningful to the message processing module (MPM) (for addressing).

Some objects, such as the Registry and user inboxes, and others designated as "critical" by a user, will be duplicated by one or more identical "backup" copies.

The object types are described below.

4.1.1.1 Documents

A document is an data structure for a multimedia document, describing the objects making up the document and the relationship between those objects. A form is a special kind of document. The objects making up a document may be atomic data objects or other documents. Documents have header and body. The summary component of a document citation is computed from the document header (see the discussion of citations, below).

There are four atomic object types, one for each of the media types text, voice, graphics and image. Atomic objects have empty UIDRefLists.

Information, for example the media, about the individual parts of the document is preserved in the document.

4.1.1.2 Folders

A folder is a data structure that holds citations to objects. Documents, for example, are accessed by citations which are stored in folders.

A citation is a reference to an object that includes an abstract or summary of the object. A citation may reference 2 types of objects: a document, or a folder of citations to objects.

Predefined folders for a user include folders called office (root folder), inbox, and desk. Users may define additional folders, including folders within folders.

Documents in a folder have ordinal positions (1,2,3,etc.). Operations on documents can reference them using a range (20:24), field contents (subject:weather), or by (a list of) symbolic names, including "*", and can use combinations of logical operations (and, or, not) and time selectors (before, after).

A citation has three parts: a reference (the UID of the object referenced); a summary (contents depend on the type of object referenced. For documents, the summary is the list of header fields. The user interface decides, based on a profile for the user, which may be null, how to present the summary.); and a symbolic name (optional for citations to documents; required for citations to folders).

No two citations in the same folder may have the same symbolic name.

Document citations in a folder have ordinal positions (1,2,3,etc.).

4.1.1.3 The_Registry

The registry contains information used for authentication, access control, and addressing (associating user names with inbox locations).

For each user there is a user description record, and for each group there is a group description record.

Each principal and group has a UID identifying it, in addition to a symbolic name. These UIDs are stored in the user and group description records in the Registry. The multiple symbolic names (mailboxes) for non-cluster users all have the same UID in their user description records. The UIDs are used in the ACLs and are bound to processes acting for users in an access control database.

Each process also has a UID identifying it. Processes are bound to users by entries in an access control database matching user UIDs and process UIDs. Records in the access control database contain:

- process UID

- user UID

- GroupsOf(user)

The principal UID and the set of group UIDs is called an access group set (AGS). The AGS is used in access control. Server processes can access the access control database using the

BindingOf operation, which returns the principal UID and the AGS that are bound to a particular process. Mechanisms are provided to permit bindings for processes in the access control database to be modified; for example, a process can cause groups to be removed from its AGS in order to reduce its capabilities.

4.1.1.4 User Descriptions

A user description is a record in the registry. A user description includes a formal name (full name); login name; password (stored non-invertibly transformed); group membership (a list); office (root) folder location (the name, typically the same as the login name); inbox folder(s) locations (position in the document store, with multiples for backup); internet mail box name (username, domain); and user profiles (containing user preferences).

4.1.1.5 Group Descriptions

A group description is a record in the registry. a group description includes a list of principals and groups that are direct members of the group, a list of groups that this group is directly a member of, and the "root" folder location for the group (position in the document store).

4.1.2 Discretionary Access to Objects

Objects in the Diamond system are associated with an access control list (ACL) identifying which users and groups have which types of access rights to the object. Various types of access rights are defined for the various types of Diamond objects. These types of access rights are enumerated below for each Diamond object type.

- Document
  - Read (required for operations such as Show, Print, Edit, Reply, Send, Export, and Archive)
  - Annotate
  - Forward (required to forward or redistribute)
  - ACLModify (required for Set Access)
  - Reclassify
- Folder
  - Read (required to read citations in a folder)
  - Add (required to File and Import citations into a folder and to name and rename citations)

- Annotate (required to Annotate a folder)
- Remove (required to Delete and Expunge citations and folders)
- ACLModify (required for Set Access)
- Reclassify (required to reclassify a folder)

- User or Group Description
  - Read (required to Show a description)
  - Modify (required to Edit a description)
  - Annotate (required to annotate a description)
  - Remove (required to Delete or Expunge a description)
  - ACLModify (required to modify the ACL of a description)
- Registry
  - Read (required to Show the names of descriptions)
  - Add (required to File a new description and alter a description)
  - Annotate (required to Annotate the registry)
  - Remove (required to Delete or Expunge a description)
  - ACLModify (required to Set Access to descriptions in the Registry)

Folders and the Registry contain predetermined initial ACLs for newly-created objects that they will hold. A folder contains an initial ACL for new folders and an initial ACL each for new objects of each of the other object types. Additionally, a folder's initial ACLs are inherited from the folder in which it is contained. The contents of ACLs and initial ACLs in a folder can be modified by users who have ACLModify rights to the folder. Also, the creator of a new object can explicitly specify an initial ACL for that object.

When a process attempts an operation, its access group set (AGS) from the access control database is compared with the object's ACL to determine whether the process has authorization to perform the operation. In particular, one of the IDs in the AGS must match an ID in the ACL whose corresponding access right is the right required to perform the requested operation.

In addition to the access rights enumerated above, which depend on object type, there are several "generic" rights:

- Read
- Annotate
- Modify (only folders can be modified)
- SetACL (make additions or deletions)
- Send (required to send an object as part of a message) (a user can get around not having send permission for an object if he has read access, by copying the object to a new one, giving himself send permission, and sending the new object. However, it may be possible to detect this, if desired.)

The ACLMerge operation is used when a user places a citation for an object into a folder. The user might do this in order to share access to the object for which he or she has access, by placing it in a "shared" folder. The ACLMerge operation passes this user's rights on to those who are on the appropriate default initial ACL for new objects in the folder. The merged ACL for the object is obtained from the initial ACL for new objects of that type in the folder and from the user's access rights to the object (obtained from the ACL for the object). Alternatively, the user can specify an initial ACL to be used instead of his or her access rights to the object.

When the citation for the object is added to the folder, for each UID on the initial ACL in the folder the intersection between the access rights and the user's access rights is computed. These are unioned with the ACL entry for the UID for the object. If the object references other objects, this is done for every object on its UIDRefList.

A user could be permitted to retrieve a folder but not a document cited in it. A user could be permitted to retrieve a document but not permitted to retrieve an object referred to by the document (in it's UIDRefList). When a process requests retrieval of a document, only the document structure and not any of the objects it refers to is returned. When access to other objects referred to by the document is needed, the process makes additional retrieval requests to obtain them (and permission is validated individually for each request). The document store validates permission by checking the access control database to see whether the process has permission to access the object.

4.1.3 *Operations*

Four activities are supported by Diamond. These are:

- Folder Presentation and Management

- Document Presentation and Composition

- Registry Presentation and Management

- Principal/Group Presentation Manipulation and Management

A number of operations will be available to the user to support these activities. These operations will not all consist of trusted software, but will make use of the secure Task 1 model operations. In other words, the user-level operations will be constructed from Task 1 model operations.

The system is required to allow users to handle data of several classifications, up to their clearance level, during a single computer session. The applications-level operations will operate in such a way as to allow a user to operate on a multilevel object, such as editing a document, while the trusted software will actually implement this as several single-level transactions against the database.

There are generic commands that can be used on multiple types of objects. See Table 3 and Table 4 (although they appear to conflict somewhat).***

4.1.3.1 Operations on Documents

A number of operations on documents will be available to the user. These operations are briefly described below.

- Show

- Print

- Move, Copy (move or copy the document citation to the indicated folder)

- Name, Rename (name or rename the citation with a symbolic name)

- Create (to create a new document)

  A user interacting with a document editor can create a collection of new atomic objects and a document structure referencing them. For each of these new objects a new object is created in the document store and a citation for it is placed in a folder. The editor sends the request

  (Create, ObjectType, ObjectValue, FolderUID)

  to the document store, where ObjectType is AtomicText, AtomicVoice, AtomicGraphics, etc., ObjectValue is the value, and

FolderUID is the UID of the folder to hold the new object. The document store allocates and returns a new UID. The editor adds the new UID to the UIDRefList and inserts it to the proper place in the document data structure. When this has been done for all referenced objects, the editor sends a create request for the document structure to the document store. The RefCount fields of objects referred to by the new document are incremented. The document store returns the new UID. The citations to the new atomic objects referenced by the document could be removed from the folder, since they are now referred to by the document (the citations were only created in the first place to avoid having unreferenced objects).

- Edit (to create a derived document or edit an existing document)

A user could create a new document that references parts of existing documents. This case is similar to creating a new document, except that instead of creating a new object (in the case of a reference to an existing object), the UID of the existing object is added to the RefList of the new document and the UID of the referenced object is inserted into the proper place in the document data structure. The RefCount field of the referenced object is incremented. The new document structure will not be created in the document store unless the user has access to all of the objects referenced.

Editing an existing document results in a new document with references to existing parts of the original document for parts that are unchanged, and the parts that are changed refer to newly created objects.

- Reply, Forward

- Annotate

- Send

When a document is sent as a message, a new object is created differing from the original in that it contains the message header. Some of these fields are user-supplied, and some are supplied by the message system.

Messages are delivered to addressees within the Diamond cluster by placing citations for it into their Inbox folders. For addressees external to the cluster, the message is translated into the proper format and passed to MPM.

Message transmission is initiated by a request of the form

(SendMessage, UID, OKorNotOKToSend)

The OKorNotOKToSend parameter indicates whether the recipient will have further send permission for the message.

Before a user is permitted to send a message, a check is made to see whether the user has send permission for the object and all the objects referenced by it. A citation for the message is placed in a MailBox folder in the document store. When the message transport server finds the message in the MailBox, it uses the Registry to find out how to deliver it to the various addressees. For individual addressees within the cluster, a citation for the message is placed in an Inbox for that addressee, with the ACLMerge operation used to update the ACL of each object referenced by the message as follows: The default initial ACL used is the one that is part of the recipient's inbox folder. The sender's access rights to the object (determined from the object's ACL) are unioned with (read, send) or (read), depending on the value of OKorNotOKToSend. Otherwise, the ACLMerge operation proceeds as described previously.

For messages being sent to users at other clusters, the document is addressed to an Internet mailbox. An Internet mailbox consists of a user name and a domain. The message processing module (MPM) at the remote host accepts the message. For messages to users at the same cluster as the originator, the user name is used as the address. The user description in the Registry is used to find where a user's mailbox is located. User descriptions for users not at the local cluster can also be included in the Registry so that they can be addressed by a simple user name.

The "send" operation gets message destinations from the "to", "cc", and "bcc" fields and puts citations to the document in the inboxes for local users or forwards the document itself to MPM for non-local users. If a group-name appears as an addressee, then a citation is distributed to each member of the group.

- Redistribute

- Import, Export (import or export the contents of a document from or to a file outside the view of Diamond)

- Delete (mark a citation as deleted)

- Expunge (remove a deleted citation from a folder. If this results in no outstanding citations to the cited document, remove the document from storage.)

- Archive

- Set Access (set the ACL for a document)

- Reclassify

4.1.3.2 Operations on Folders

A number of operations on folders will be available to the user. These operations are briefly described below.

- Show (display a list of the citations in the folder)

- Print (hard copy version of "show")

- Move, Copy (move or copy the folder citation to the indicated folder. If the indicated folder does not exist, then create a new empty folder with the indicated symbolic name.)

- Rename (change the symbolic name of a citation to a folder)

- Annotate

- Delete (mark a folder citation as deleted. A folder may not be deleted until all the document in it have been deleted and expunged.)

- Expunge (remove a citation to a deleted folder from the containing folder)

- Archive (archive a folder and all of its documents)

- Set Access (set the ACL for a folder)

- Set Initial Access (set the initial ACL for objects in a folder)

- Reclassify

4.1.3.3 Operations on User Descriptions

A number of operations on user descriptions will be available. These operations are briefly described below.

- Show, Print

- Create, Edit (uses a form to be filled in)

- Annotate

- Delete (mark as deleted)

- Expunge (remove a deleted user description from storage)

4.1.3.4 Operations_on_Group_Descriptions

A number of operations on group descriptions will be available. These operations are briefly described below.

- Show, Print

- Create, Edit (uses a form to be filled in)

- Annotate

- Delete (mark as deleted)

- Expunge (remove a deleted group description from storage)

4.1.3.5 Operations_on_the_Registry

A number of operations on the registry will be available. These operations are briefly described below.

- Show, Print (show or print a specified selection of the Registry contents)

- Annotate

There is also a MembersOf operation that returns either a list of the direct members of a group or a list of all (direct and indirect) members of a group. Direct members of a group are listed as principals on the Group Description Record in the Registry. Indirect members of a group are members of a group that is a member of the group.

There is a GroupsOf operation that, for a specified principal, returns either a list of the groups of which the principal is a direct member or a list of the groups of which the principal is either a direct or an indirect member.

The registry is referenced by the authentication service for purposes of user authentication, as follows. The user supplies a login name and password. The authentication server searches the registry for a user description record for the user. If it finds one, it transforms the password and compares it with the one stored in the user description record. If the passwords match, the authentication server creates a new record for the access control database to bind the user name to processes acting on the behalf of that user. The authentication server returns an acknowledgement to the user.

4.1.4 Mandatory_Access_Controls

A goal of the Diamond system is to allow users to handle information of several classifications simultaneously. There will be a separate document store for each security level (U, C, S, TS), so that each Diamond host can operate at a single level. User workstations, however, will be multilevel. Diamond objects are multilevel.

The "container clearance required" restriction is appropriate for documents and messages. The absence of the "container clearance required" restriction is appropriate for folders of documents and messages. However, the creator of a folder can specify that the container clearance is required, if so desired.

There is a reclassify access control right that can appear on ACLs, required for reclassification of documents in the document store. When a document is in draft, the drafter typically will hold reclassify rights to it. When a document is released, the drafter's reclassify rights are revoked and users acting as security officers have reclassify rights to it.

The document editor will keep track of the classification of parts of existing documents that are inserted into the document being edited, to prevent the user from improperly classifying the new document. Information removed from an atomic object inherits the classification of that object.

User clearances will be stored in the Registry in the user's Principal record.

Processes will be assigned a working security level; process levels will depend on the clearance of the user they represent and on the maximum clearance level of the device on which they are running. Before the access check described above, the user-clearance/object- classification check will be made; for this check the working level of the process will be used for the user clearance level.

4.2 Constructing the Diamond Data Structures From the Task 1 Model Data Structures

The Diamond data structures will be constructed from the data structures of the Task 1 model. Below we describe how a Diamond object is interpreted in terms of the entities of the Task 1 MLO model. Subsequent sections describe mappings for each of documents, folders, the registry, user descriptions and group description objects.

4.2.1 Diamond_Object/MLO_Entity_Mappings

Although a Diamond object is considered to be an atomic data object or a non atomic data object, all such objects are actually composed of several different fields. Recall from section 4.1.1 that objects in the Diamond system each have the following standard information:

- UID/object type

- Value

- UIDRefList

- AccessControlInfo

- AnnotationList

- RefCount

Only one of Value and UIDRefList appear in a Diamond object. Atomic data objects have empty UIDRefLists and non-empty values. Non-atomic data objects have empty values but non-empty UIDRefLists. They refer to other data objects. The Value or UIDRefList field is considered the "body" of the object. The other fields of the Diamond object are called "header" fields.

Every Diamond object can be described using a hierarchy of MLO entities. Recall that an MLO entity has four fields;

- Flag

  Identifies the entity as a MLO container or an MLO atom

- SecurityLevel

- Status

- Data

  If the entity is an atom, the data is "atomic data" (text, voice, etc). If the entity is a container, then the data is a set of parts descriptors.

At the top of the MLO hierarchy representing a Diamond object is a MLO container called a *master entity*. The master entity is marked as a container, and has the security level and status of the Diamond object it represents. The parts descriptors of the container are a set of pointers to an MLO entities representing fields of the Diamond object. These MLO entities are called *field entities*. There is one MLO field entity for each of the standard Diamond object fields listed above. Each field entity inherits the security level and status of its associated master entity. The field entities for UID, Value, ReferenceCount, and AccessControlInformation are marked as MLO atoms. The MLO data area contains the UID, value,count, access information specific to the associated master entity. The field entities associated with the UIDRefList and the AnnotationList are marked as containers. The parts descriptors for these entities are UIDs. If a field is not used in a specific Diamond object, the MLO entity for that field is marked UNUSED, the security level and status fields are undefined and the data area is empty of data.

A Diamond object is referenced by its UID, and contains references to other specific Diamond objects. Ideally, we would like the implementation of the Diamond object to be referenced by its UID, and to contain references to the implementations of the other specific Diamond objects using the UID of the other Diamond objects. For example, Suppose D0 is a Diamond object which has references to other Diamond objects D1, D2, ..., Dn. Then the MLO entity hierarchy implementing a Diamond object D0 will be referenced by the UID of D0 and will have references to the MLO implementations of D1, D2, ..., Dn. An MLO entity is referenced by a descriptor. We simply require that the descriptor of a Diamond object's master entity be equal to the UID of the Diamond object. This, together with the following constraints on the reference mechanism, are sufficient to have reference to the master entity of a MLO implementation of a Diamond object provide the full information set of the Diamond object. Although following constraints are generally not security-significant, they should be considered in order to make the mapping useful.

- Reference to a master entity of a MLO hierarchy implementing a Diamond object should guarantee access to all the associated MLO field entities.

  Violation of this property is a form of denial of service.

- MLO field entities cannot be referenced directly.

  That is to say, one must locate a field entity via the associated master entity. Violation of this property is not security critical as each MLO entity contains identical security markings as the associated master record and Diamond object.

Diagram 1 depicts the implementation of a Diamond object in terms of MLO entities.

Needed within the Diamond system is the capability to build complex objects out of more simple or atomic objects. For example, a message might contain some text and a reference to another message. Folders contain citations, which are references to documents. A document may be a hierarchy of sub documents. A document built of other documents might look like the picture in diagram 2. In this figure, objects will be illustrated as not having annotation lists. It will be assumed that any annotations needed within the document have been incorporated into it. This is to allow the construction of a simpler diagram.

The various Diamond objects of the above hierarchy can have differing security levels, so long as the levels obey the security level hierarchy rule stated in task 1. The reference mechanism can be used, as described in task 1, to control container clearance requirements.

Going one step further, we can visualize the above Diamond object structure, replacing the objects by their MLO implementation. What results is a multilevel hierarchy of singlelevel sub-hierarchies. See diagram 3.

4.2.2 Documents

A document can be a Diamond non-atomic-data document or a Diamond atomic-data document. Annotations are a form of document. They have the same structure and can be atomic or not.

As described above for Diamond objects in general, a Diamond document is implemented by an MLO master entity and a set of MLO atoms and containers which define the document header and body fields. The following list describes the implementation of the Diamond document fields in terms of MLO entities.

- UID/ObjectType

  This document field is implemented by an MLO atom.

- Value

  The value field is implemented by an MLO atom. For atomic-data documents it contains the atomic data. The value field is not needed for a Diamond non-atomic-data document, and the MLO entity is marked UNUSED, with the data area of the MLO atom empty.

- UIDreflist

  This field is implemented by an MLO container. The parts of this container are the UIDs of documents referenced. In the case of atomic-data documents, the MLO entity is marked UNUSED and the parts descriptor list is empty.

- AnnotationList

  This document field is implemented by an MLO container. If there are no annotations for a document, the parts descriptor set is empty. The parts of this container are the UIDs of annotation documents which refer to the Diamond document. The first element of the parts list is a pointer to an MLO atom which contains a pairing of the UIDs and ReferencePoints within the document.

- AccessControl

  This document field record is implemented by an MLO atom.

- RefCtr

  This document field record is implemented by an MLO atom.

The master entity for a Diamond document is identical to the description of master entities above. The following two diagrams depict an implementation of a Diamond document which is non-atomic and one which is atomic. Diagrams 4 and 5 describe atomic and non-atomic documents in terms of MLO entities.

4.2.3 Folders

A folder is a Diamond data structure which holds citations to documents or other folders. Citations are not considered Diamond objects, and have three parts; a UID reference, a summary, and a symbolic name. Summaries for citations referring to documents can be derived from the document "header" fields. Summaries are null for citations referring to folders. Because the summary is readily available in any case, we do not implement them directly here.

A Diamond folder is implemented by an MLO container as a master entity and and a set of MLO atoms and containers which define the folder header and body fields. The master entity is marked as a container, and the classification and status fields are given by the creator. The field entities referred to in the parts list inherit the classification and status of the master entity. The following list describes the implementation of the folder field entities;

- UID/ObjectType

  This folder field is implemented by an MLO atom.

- Value

  The value field is implemented by an MLO entity, but it is not needed for Diamond folders. The MLO entity is marked UNUSED, with the data area of the MLO atom empty.

- UIDreflist

  This folder field is implemented by an MLO container. The parts list contains one pointer and a list of UIDs. The UIDs in the list of parts are the citation UIDs. The pointer is to an MLO atom which contains a table matching UIDs and symbolic names. Any change to the citation UID list necessitates a change to this MLO atom.

- AnnotationList

The folder field is implemented by an MLO container. The parts list contains a list of UIDs of annotations to the folder.

- AccessControl

  This folder field is implemented by an MLO atom.

- RefCtr

  This folder field is implemented by an MLO atom.

Diagram 6 depicts a folder implementation.

4.2.4 The_Registry

The registry will be represented in terms of the Task 1 model data structures by a collection of user description records and group description records. All registry records are addressed by a UID and a symbolic name. The User and Group records are discussed below.

4.2.5 User_Descriptions

A Diamond user description record is implemented by an MLO container as a master entity and and a set of MLO atoms and containers which define the header and body fields. The master entity is marked as a container, and the classification and status fields are given by the creator. The field entities referred to in the parts list inherit the classification and status of the master entity. The following list describes the implementation of the folder field entities;

- UID/ObjectType

  This field is implemented by an MLO atom.

- Value

  The value field is implemented by an MLO container, whose parts list refer to MLO atoms and containers for the following:

  - formal name (MLO atom)
  - Login name (MLO atom)
  - Password (MLO atom)
  - List of groups the individual is a member of (MLO container of atoms for membership identities)

- office (MLO atom)
- folder location (root) (MLO atom)
- internet mailbox name (MLO container of atoms for mailbox names)
- user profile list (MLO container of atoms for profiles)

- UIDreflist

  This field is implemented by an MLO container and represents the user inbox. The parts list contains a list of UIDs indicating folders in the user's inbox.

- AnnotationList

  The field is implemented by an MLO container. The parts list contains a list of UIDs of the user record annotations.

- AccessControl

  This field is implemented by an MLO atom.

- RefCtr

  This field is implemented by an MLO atom.

Diagram 7 depicts a user record implementation.

4.2.6 Group Descriptions

A Diamond group description record is implemented by an MLO container as a master entity and and a set of MLO atoms and containers which define the header and body fields. The master entity is marked as a container, and the classification and status fields are given by the creator. The field entities referred to in the parts list inherit the classification and status of the master entity. The following list describes the implementation of the folder field entities;

- UID/ObjectType

  This field is implemented by an MLO atom.

- Value

  The value field is implemented by an MLO container, whose parts list refer to MLO atoms and containers for the following:

- formal name (MLO atom)
- direct membership list -- a list of groups and individuals belonging to this group (MLO container of atoms for membership identities)
- direct member of list -- a list of groups this group belongs to. (MLO container of atoms for membership identities)
- folder location (root) (MLO atom)

- UIDretlist

This field is implemented by an MLO container, but is not used.

- AnnotationList

The field is implemented by an MLO container. The parts list contains a list of UIDs of the group record annotations.

- AccessControl

This field is implemented by an MLO atom.

- RefCtr

This field is implemented by an MLO atom.

Diagram 8 depicts a group record implementation.

4.3 Building the Applications-Level Operations from the Task 1 Model Operations

User operations on documents, folders, the registry, user descriptions, and from the basic operations of the Task 1 model. When a user invokes a user operation requiring a query or transaction against the database, this will be translated to one or a sequence of Task 1 model operations on the database.

Numerous operations on the Diamond objects have been defined above. For many of these operations, the steps required to describe implementation in terms of MLO operations are similar. Many of the Diamond operations define releasing Diamond data to some external medium, for example. Defining in detail the MLO operations to execute each of these Diamond operations would be repetitive. For this reason, we have chosen to identify similar operations by defining classes of Diamond operations and describing a mapping for representatives of the class, rather than defining a mapping for each such operation. Significant differences in the MLO mappings for different members of a class will be noted.

The classes of Diamond operations defined are as follows:

- Operations releasing Diamond information to external media
- Operations requiring the alteration of field values of Diamond objects
- Operations requiring the creation of new objects
- Miscellaneous operations

In the following sections, the operations for each such class are identified and followed by a mapping of one or more representatives of the class.

4.3.1 Operations_Releasing_Diamond_Information_to_External_Media

The Diamond operations of this class are:

- Show (Display a document)
- Print(Hard copy version of "show" for documents)
- Show (display a list of the citations in a folder)
- Print (hard copy version of "show" for a folder)
- Import, Export (import or export the contents of a document from or to a file outside the view of Diamond)
- Archive (archive a single document)
- Archive (archive a folder and all of its documents)
- Show, Print (user or group description records)
- Show, Print (show or print a specified selection of the Registry contents)
- Reply, Forward (documents sent as messages)
- Send (documents sent as messages)
- Redistribute (documents sent as messages)

The representatives for this class are show and print. For each MLO entity representing a requested field of the requested UID, the MLO read operation is invoked to retrieve the field data and the display operation is then invoked to display the field information to the requestor.

4.3.2 Operations Requiring Alteration of Field Values of Diamond Objects

The operations of this class are:

- Move, Copy (move or copy the document citation to the indicated folder)
- Name, Rename (name or rename a document citation with a symbolic name)
- Delete (mark a citation as deleted)
- Set Access (set the ACL for a document)
- Reclassify (a document)
- Move, Copy (move or copy the folder citation to the indicated folder. If the indicated folder does not exist, then create a new empty folder with the indicated symbolic name.)
- Rename (change the symbolic name of a citation to a folder)
- Delete (mark a folder citation as deleted. A folder may not be deleted until all the document citations in it have been deleted and expunged.)
- Set Access (set the ACL for a folder)
- Set Initial Access (set the initial ACL for objects in a folder)
- Reclassify (a folder)
- Delete (mark as deleted)

The representative operations of this class are the Move or Copy operation and the Delete operation. The style of the data mappings to MLO entities creates the citation list of a folder as a data field, in the same sense that the access information, or the UID or the name are represented by fields. Thus the Move or Copy operation is representative of any Diamond operation which requests alteration to the contents of a field of a Diamond object. In both the Copy/Move operation and the Delete operations, detailed mappings are given for the case where the operation copies (or deletes) a document citation in the folder. These operations are identical in the case of copying or deleting citations to folders in folders.

- Move, Copy (move or copy the document citation to the indicated folder)

The MLO read operation is invoked to read the UIDReflist field of the appropriate folder. This allows identification of the UID-summary cross reference table atom for the citations in this folder. Next, the MLO create operation is invoked to create an atom to hold the information of the new citation. The write operation is invoked to write the UID of the new citation to the MLO atom created. The MLO write operation is invoked to write the symbolic name, summary and UID cross reference information into the MLO cross reference atom. The MLO write operation is invoked to write the descriptor of the new citation into the UIDRefList of the appropriate folder.

- Delete (mark a document citation in a folder as deleted)

The MLO read operation is invoked to read the UIDReflist field of the appropriate folder. This allows identification of the UID-summary cross reference table atom for the citations in this folder and the UID record for the citation to be deleted. Then the MLO write-status operation is invoked to alter the status of the UID record to DELETED. The MLO write operation is invoked to remove the entry for the deleted UID from the UID-summary-symbolic name cross reference atom. The MLO write operation is invoked to remove the descriptor of the UID atom from the UIERefList container.

4.3.3 Operations requiring the creation of new objects

The operations of this class are:

- Create (to create a new document)

- Edit (to create a derived document or edit an existing document)

- Annotate(a document or a folder)

- Create, Edit (uses a form to be filled in)(user description)

- Annotate (a user or group description record)

The representative operation for this class is the create operation. A mapping for the create document operation is given below. The mapping for the Edit command is similar, but does not invoke the MLO create operation, but rather the MLO read operation. Create Folder is similar to the create document operation. Create user or group description is also similar, but an appropriate structure is created, then a "template" is copied into the structure which is then edited. Notice that annotating a record is the creation of a special document, so the Diamond annotation operations are simplifications of this mapping.

THE MLO MODEL--TASK 3

The Diamond Create Document operation creates a document by developing a hierarchy ot diamond objects, each of which is described as a hierarchy ot MLO entities within the mapping below. The Diamond Create Document operation also requires the creation and addition of a citation to a folder, or the deletion of citations from a folder at the time a completed document is addėd to the document store. MLO mappings for these parts of the Create operation are described in detail above, and these parts of the mappings will not be repeated here. The Diamond request (Create, ObjectType, ObjectValue, FolderUID) generates the creation ot objects for the various parts of the document. For each "atomic" unit of information ot the document, a hierarchy of MLO objects is developed as follows. A specific MLO entity contains the value of the next available UID. The MLO read operation is invoked to retrieve the UID for the Diamond atomic object. The MLO create operation is invoked to create a master entity for the Diamond object. The descriptor of the master entity is equal to the UID of the Diamond object. For each field of the Diamond object, the MLO create operation is invoked, creating the appropriate MLO atom or container for the field. The MLO create operation assigns the security level assigned by the sponsor, assigns the appropriate atom or container indicator, assigns default status indicators and leaves the data area empty for each entity created. The descriptor ot each such MLO entity is a function ot the UID ot the master entity (the Diamond object) and the identity of the field itself. MLO atoms are created for UID, Value, Access Control and RetCount, using the MLO create operation. MLO containers are created for UIDRefList and AnnoteList. When all the appropriate MLO entities are created for the master entity, A citation is created for the object, and the move operation, described above by an MLO mapping, is invoked to write the citation into the folder identified by the FolderUID defined above. Now the MLO write operation is invoked, to write the contents into the appropriate MLO entity for each field of the Diamond object record. The annotationlist is left blank, the RefCount is 0, the UID field is the UID assigned above, and the atomic data is written into the MLO entity for the atomic data.

This process is repeated for each Diamond atomic data object of the desired document. When all such information is properly written into hierarchies of MLO entities, a the procedure is repeated, creating a MLO hierarchy to hold the information pertinent to the entire document. A UID for the document is obtained as described above, using the MLO Read function. A master entity and the appropriate field entities are created as above, using the MLO create operation. The MLO create operation is also invoked to create one MLO atom for each Diamond atomic data unit which is a part of this document, represented by an MLO hierarchy. The citation to the document is added to the appropriate folder as above. The MLO field entities are written to using the MLO write operation, as above, with the following exceptions.

- The MLO atom for Value is left blank.
- The MLO entity for the UIDRefList contains the descriptors of each of the MLO atoms created for the Diamond atomic data units, described above.
- The MLO atoms for the Diamond atomic data units have the UIDs of the Diamond objects (the descriptors of the MLO hierarchies representing them) written as the contents of their data fields.

For each entry in the MLO container for the UIDRefList, the RefCount for the atomic data pointed to must be incremented. This is done as follows. The MLO read operation is invoked to read the UIDReflist of the MLO hierarchy representing the document. For each entry in this list, the MLO read operation is invoked to read the value of the entity (it is the MLO descriptor of a Diamond atomic data representation), then the MLO write operation is invoked to alter the contents of the MLO entity for the RefCount field of the identified UID.

Finally, the citations to the atomic Diamond data objects are deleted from the respective folder, as described above, and a citation to the document is copied into the folder, as described above.

4.3.4 Miscellaneous_Operations

The operations in this class are the expunge operations. Mappings for all these operations are similar. When a MLO entity is determined to be marked deleted when it is requested to be expunged (determined by reading the entity using the MLO read operation) then the MLO delete operation is invoked to physically remove the entity from the database.

4.4 A_Candidate_Architecture_for_Future_Versions_of_Diamond

Our recommended architecture for a secure Diamond Document Handling System includes a number of elements addressing different aspects of enforcement of the security policy. To minimize the modifications required to the current Diamond design while providing an easily verifiable, conceptually simple security architecture, the problem can be separated into a collection of smaller problems that can be solved using single-purpose, physically isolated devices.

These devices include a Network Interface (NI) unit to enforce mandatory security for traffic between entities on the local network, a User Interface (UI) unit to establish a secure communications path between a Diamond object manager or authentication server and a user accessing Diamond through an untrusted Access Point, and a secure user workstation. The collective solutions can be combined to provide multilevel security for the distributed Diamond system.

Because we believe it is impractical to expect multilevel hosts to support the Diamond object handlers, we suggest there be several document handlers, each at a single security level. Multilevel documents are composed of a hierarchy of documents, with the documents at the bottom of the hierarchy at a single security level. A trusted object manager would reassemble the multilevel documents from their components in such a way as to enforce the multilevel object model for Diamond. This trusted object manager would be a small, microprocessor-based, physically separate, special-purpose device. Because it serves a limited, well-defined function, the amount of code implementing it could be made small enough so that is is practical to verify. There could be multiple trusted object managers on the network.

Likewise, the Diamond authentication manager, containing the Registry and the access control database, could reside on a small, microprocessor- based, physically separate, special-purpose host. Its functions would be limited, so that the amount of code implementing it could be made small enough to verify. Thoughtful design could result in shared code between the trusted object manager and the authentication manager, reducing the verification effort even further.

The single-level document managers could store documents sealed with cryptographic authenticators. Because documents once entered in the Diamond document store are never changed, sealing them with authenticators is feasible.

Our design is flexible, allowing the gradual addition of security devices to the distributed system to achieve broader functionality while preserving or increasing the overall security of the network. The NI provides a simple means of encompassing within the network a number of devices at different security levels. The UI can be added at the Diamond Access Points to provide a means of enforcing discretionary access control for users desiring access to objects on the object managers. The secure workstation can be added to provide the possibility for users with different clearances to share the same workstation and for workstation users to create objects at a classification lower than their clearance level.

Below we discuss the benefits of the suggested design approach for a secure Diamond system to the verification effort. Following that, we present discussions of some of the security mechanisms and show how they address the various security issues. A more detailed analysis is not possible within the scope of this study.

4.4.1 The Benefits of this Approach to the Verification Effort

The Diamond functions are distributed throughout a number ot different types of machines. We believe the most practical and easily demonstrable security mechanisms for such a system are physically isolated security devices that can be added on to the system with minimal disturbance to the current design.

Our basic design approach is to decompose the overall security problem into a number of separable problems, and to propose solutions to these problems that can be entorced by isolatable security mechanisms. The sum ot these security mechanisms results in a conceptually simple and easily veritiable design that provides a cohesive and tlexible solution to system security.

An advantage to this approach is the ability to physically isolate the security mechanisms from the rest of the system. Physical isolation ot the security mechanisms not only allows for them to be incorporated in tamperproof enclosures but simplifies the veritication effort by minimizing the amount of software to be tormally specified and verified.

The Diamond model presented in a previous section is a system-wide model of security tor Diamond. Submodels for the requirements that must be enforced by trusted components ot the system should be developed and mapped to the system wide model. These maps should be shown to be correct.

Because the entorcement of security will be distributed among separate trusted components, it is necessary to decompose the system security model into models for the individual trusted components. The next step is to show that the collection of submodels implies the system model. This will be done by a proot that if each trusted component is in a secure state, the system will be in a secure state.

The ability to decompose the system model into submodels is an advantage for verifying distributed systems, because the submodels apply to trusted components that are simpler than the system as a whole. Because the system model will have been decomposed into submodels, an FTLS can be developed separately for each trusted component corresponding to a submodel.

4.4.2 Mandatory Security Enforced by Network Interface (NI) Units

Mandatory security refers to the separation of information at different levels of classification and to the policy of access restrictions imposed by the system of user clearances. Within the Diamond context, mandatory security means enforcing a security policy describing the allowable accesses between subjects and objects at various security levels.

THE MLO MODEL--TASK 3

Our suggested mechanism for enforcing mandatory security within Diamond is an NI unit placed between each entity connected to the local network, and the network itself. The NI would reliably mark all traffic entering the network with the correct classification and would allow only those transactions marked with authorized classifications to pass through it from the network into the entity it protects.

Each entity on the network is regarded as either trusted or untrusted. An untrusted device is authorized to contain information at a single security level. All information leaving an untrusted device is considered to be at that security level, whereas trusted devices may be authorized to contain information at a fixed range of classifications and are considered to reliably mark information leaving them with the actual classification. For purposes of our discussion here, a device is regarded as trusted if the security mechanisms it implements have been formally verified to operate correctly.

The NI appends the device name and the correct security level to messages entering the network and then computes an authenticator. In the case of an untrusted, single level device the security level used is the level of the device. In the case of a trusted multilevel device the security level used is the level supplied by the trusted device. The authenticator is a function of the message, the device name, and the security level. The authenticator must also depend on some secret information not available to untrusted entities; otherwise untrusted software could change the security level or content of the message and compute an authenticator for the new message. An authenticator that depends on the message content, device name, security level, and a secret key can reliably detect any modification of the content, device name or security level.

A proven method of generating authenticators is to use the Federal Data Encryption Standard (DES) in Cipher Block Chaining mode. In this mode a the first 64 bit block of the message is added modulo two to a 64 bit initial value. The result is encrypted using DES with a 64 bit key. The result is added modulo two to the next 64 bit block of the message and the sum encrypted. This process continues until the entire message, device name, and security level have been used. The last block of cipher text computed is used as the authenticator.

At the point at which the message leaves the network for presentation to another device, another NI recomputes the authenticator, allowing the transaction to proceed only if the recomputed authenticator matches the original one and if the receiving device is authorized to receive information classified at the level of the message. A matching authenticator means that the message, the device name, and the classification have not been altered in any way.

These controls allow single level untrusted devices at a variety of security levels to coexist and communicate on the same network as trusted devices, maintaining separation of information at different levels of classification while permitting communication among devices in accord with the mandatory security policy.

The NIs would be built as physically separate, self-contained microprocessor based units. Because they are physically separate and self-contained, they can be constructed as tamperproof units. The NIs would use at least three domains of execution. The network interface and the device interface will execute in domains which are separate from the trusted domain which marks, filters, and computes authenticators. The trusted software domain may be further decomposed into separate domains.

Some possible mechanisms for creating separate execution domains are processor-per-domain and a hybrid architecture. The processor-per-domain architecture uses a separate processor for each domain. Communication between domains is accomplished by shared memory. The advantage of this design is simplicity of demonstrating domain separation. The hybrid architecture uses memory mapping separate from the microprocessor to create multiple domains.

Once a message has been entered into the network with an authenticator, it can be stored for later use. That is, untrusted document managers attached to the network can store the data for future access along with its authenticator. If the authenticator matches when the data leaves the network, we know that the data was not contaminated or modified.

4.4.3 Discretionary Access Control

Discretionary access control refers to the restriction of users from objects they have not specifically been authorized access to by name. Within a multilevel secure system, discretionary access controls act to further restrict users from objects for which they may in fact hold the appropriate security clearance. Such controls require a user to have a "need-to-know" for the information they request. Within Diamond, objects have associated access controls lists containing names of users and groups along with the access rights to the object specifically accorded to those users and groups.

As a function, discretionary access control can be broken down into two component functions: user authentication; that is, determining unequivocably the identity of any user; and mediation of user access to system objects. In the Diamond system, responsibility for user authentication and access control lies with the Diamond authentication server, which contains the Diamond Registry and the access control database.

The identity of users accessing Diamond through an untrusted access point can not be reliably determined. Thus for the purpose of making access decisions, reliable information about user identity may not always be available.

The design for a secure Diamond should be able to handle both trusted and untrusted components. An untrusted component can not be relied upon to maintain separation among users or resources on that component. Thus while subjects can be considered to have an associated security level and access point and user identifiers, the design will also have to consider the privileges available to a subject when only the security level and access point can be reliably determined. This would be the case, for example, if untrusted terminal access controllers are used as Diamond access points, or in the case of a user gaining access through an untrusted internet gateway.

One important factor to consider in developing a design for a secure Diamond system is that untrusted machines can not be relied upon to provide correct information about users or resources they control. Of course, we may believe that these untrusted machines behave properly, but by definition we lack the degree of assurance provided by formal verification. It makes no sense to formally verify that Diamond makes a correct decision based on a user's identity if that user identity has been supplied by an untrusted machine.

Information Diamond can use in making access control decisions include security level, the user's access point, and user name. Although using the NIs. Diamond can reliably obtain the device name and security level, and Diamond can reliably obtain user names from a trusted device, user names obtained from untrusted devices can not be trusted to be correct.

Users obtaining access through an untrusted access point or gateway can be given certain generic access rights corresponding to the security level of the untrusted access point or gateway, even though the user name will not have been obtained with a high degree of assurance as to its correctness. Users at different access points or gateways could have different sets of predefined generic access rights, one set corresponding to each untrusted access point or gateway in the Diamond network.

To apply discretionary access controls in the Diamond system it is desirable to be able to obtain reliable values for user names for users accessing Diamond through untrusted access points. However, it is impractical to require trusted access points for the Diamond system. Retaining the ability to handle untrusted access points allows for the flexibility to include a mix of trusted and untrusted user workstations and untrusted access points in the Diamond system. In order to allow users at untrusted devices to access Diamond objects in cases in which the discretionary access policy requires a reliable user identification, a trusted communications channel is needed between the user and the trusted object manager. We may consider the communication channel from the user through the untrusted access point and then through the network to the trusted object manager to be a line subject to active and passive wiretapping. There is a significant body of literature on using cryptograhic methods to establish secure channels in the face of these threats. The appropriate protocols can be implemented on the trusted object manager at one end of this line. At the user end a trusted User Interface (UI) unit is required. Encryption-based protocols could be used for establishing secure channels from remote users to trusted object managers.

In cases in which a terminal concentrator is used as an access point, the UIs would be located at a user terminal. In cases in which an untrusted user workstation is an access point, the UI would be located at the user workstation. The UI could be a freestanding device similar to SYTEK's PFX user authentication device, or it could be a device placed inline between the user and the access point. In any case, the UI would, using encryption-based protocols, reliably obtain the user identity and establish a trusted path for purposes of user identification between the user and the trusted object manager or trusted authentication manager.

4.4.4 Multilevel User Workstations

Multilevel user workstations can be added to the Diamond system to provide the flexibility for users with different clearances to share the same workstation and for workstation users to create objects at a classification lower than their clearance level.

The multilevel workstation would be a multilevel device capable of supporting users, one at a time, with different clearances and discretionary access permissions. Because it incorporates verified multilevel security, the multilevel workstation would be trusted to mark all outgoing transactions with the correct security level and seal them with an authenticator, and it would recompute the authenticator for incoming transactions. Moreover, the multilevel workstation would be able to reliably authenticate its users. The multilevel workstation would be designed with a small Trusted Computer Base (TCB) to regulate all disk, memory, display, printer, and network usage.

The design of the multilevel workstation should meet the following security requirements:

- Login

  Since a number of users might share a workstation, the workstation must be able to identify them and associate that user name with a clearance level and specific set of access rights (tor objects and resources local to the workstation).

- Communication with the Network

  When a user wishes to communicate with another device on the network, the secure workstation will also perform the functions that would otherwise be performed by the NI; that is, it will mark all outgoing transactions with the correct security level and seal them with an authenticator, and it will recompute the authenticator for incoming transactions and check that the security level is permissible for the workstation and user.

- Discretionary Access Control for Local Objects

  The workstation must be able to enforce a policy of discretionary access control for local objects.

- Mandatory Security Controls

  The workstation must maintain separation of information at ditferent levels of classification and must ensure that a user is not permitted access to information for which he or she has no clearance.

- Display

  The video display must be marked with the highest classitication ot information displayed on it.

- Re-use of Memory

  The workstation must purge buffers, RAM, and other reusable memory when there is a change of security level.

4.4.5 Trusted Object Manager

A secure Diamond architecture could include several document managers, each at a single security level, so that these file systems do not have to be trusted. However, since multilevel documents are composed of a hierarchy ot documents, there would be a need tor a trusted object manager to reassemble the multilevel documents from their component documents in such a way as to entorce the multilevel object model for Diamond. This trusted object manager would be a small, microprocessor-based, physically separate, special-purpose device. Because it serves a limited, well-defined function, the amount ot code implementing it could be made small enough so that is practical to verify.

One possible design for the trusted object manager would be based on SYTEK's Trusted Domain Machine (TDM). The Trusted Domain Machine is a generic security device designed to entorce an application-specitic security policy detined when the TDM is configured. The security-relevant software in the TDM is physically isolated in its own processing domain and includes a verified "core" security enforcement mechanism that can be used by an application to enforce its specific security policy. The trusted object manager could be implemented as an application on the TDM.

4.4.6 Trusted Authentication Manager

Like the trusted object manager, the Diamond authentication manager, containing the Registry and the access control database, could reside on a small, microprocessor- based, physically separate, special-purpose host such as the TDM. Its functions would be limited, so that the amount of code implementing it could be made small enough to verify. Thoughtful design could result in shared code between the trusted object manager and the authentication manager, reducing the verification effort even further.

4.5 Task_3_Diagrams

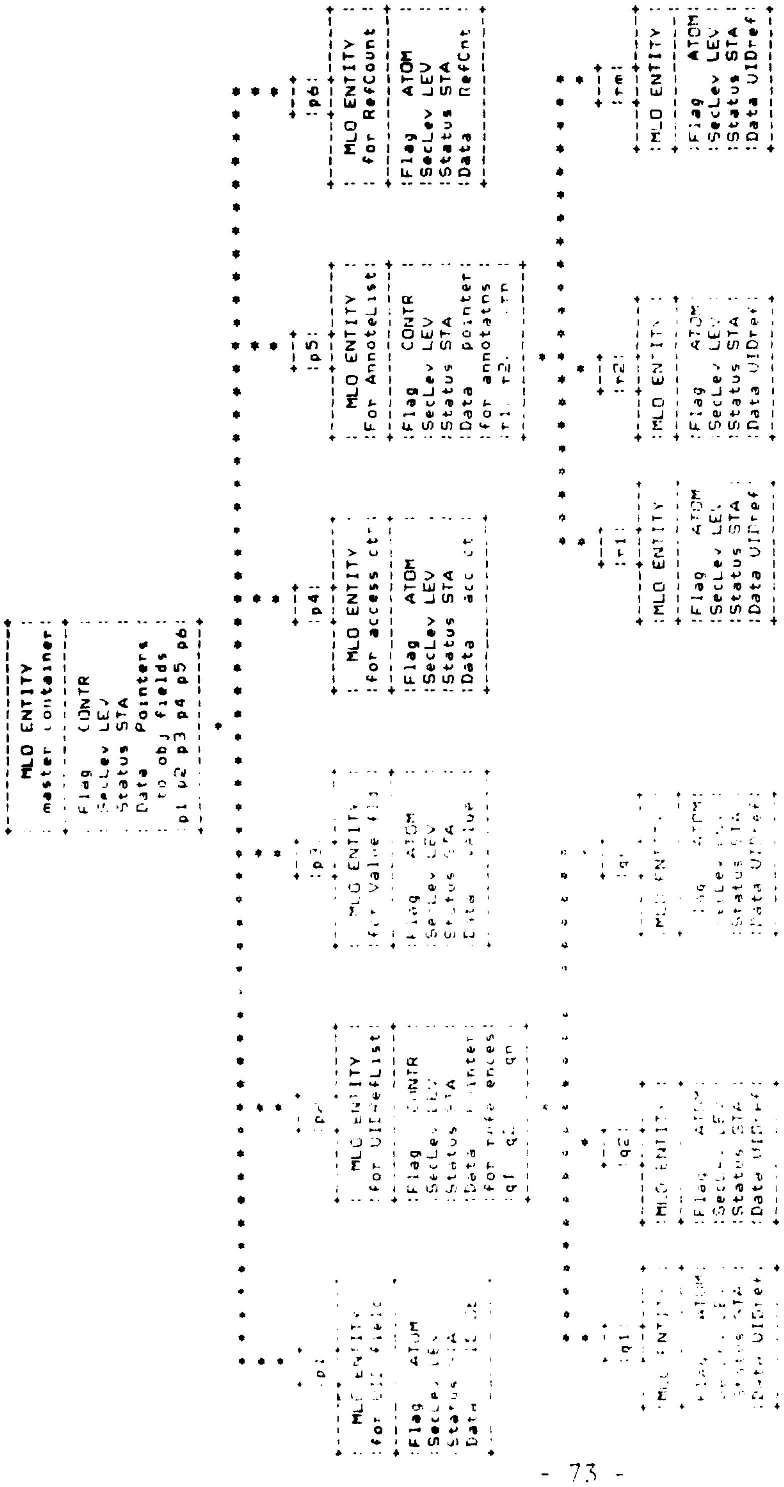


Figure 1: An MLO Implementation of a [illegible]

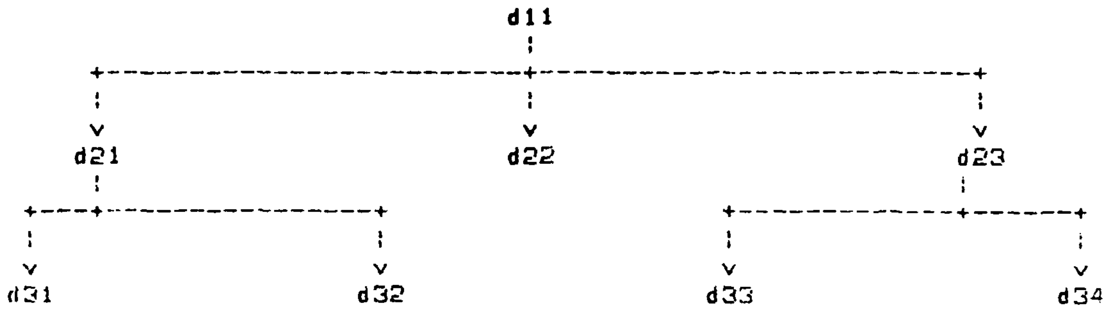


Figure 2. An Example of a complex Diamond Object (Diagram 2)

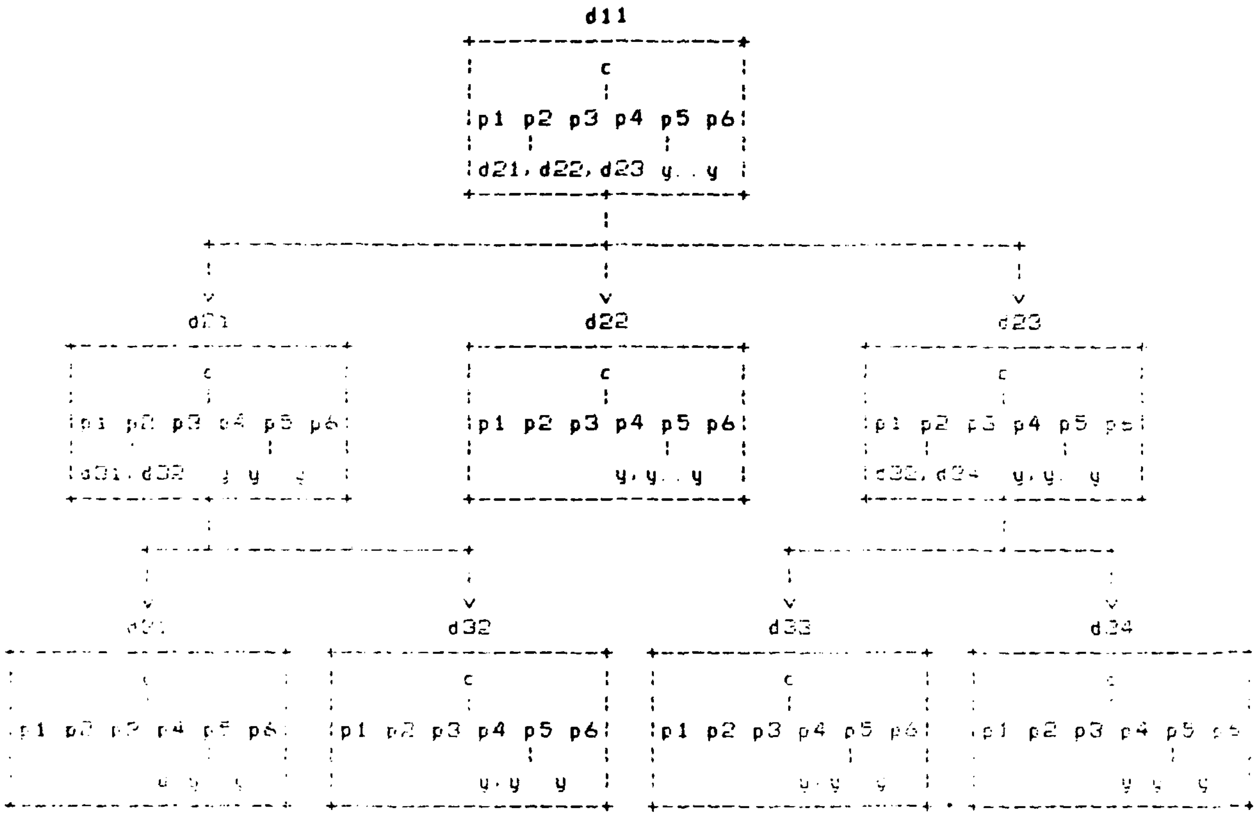


Figure 3 An Example of the MLO implementation of a complex Diamond Object (d3)

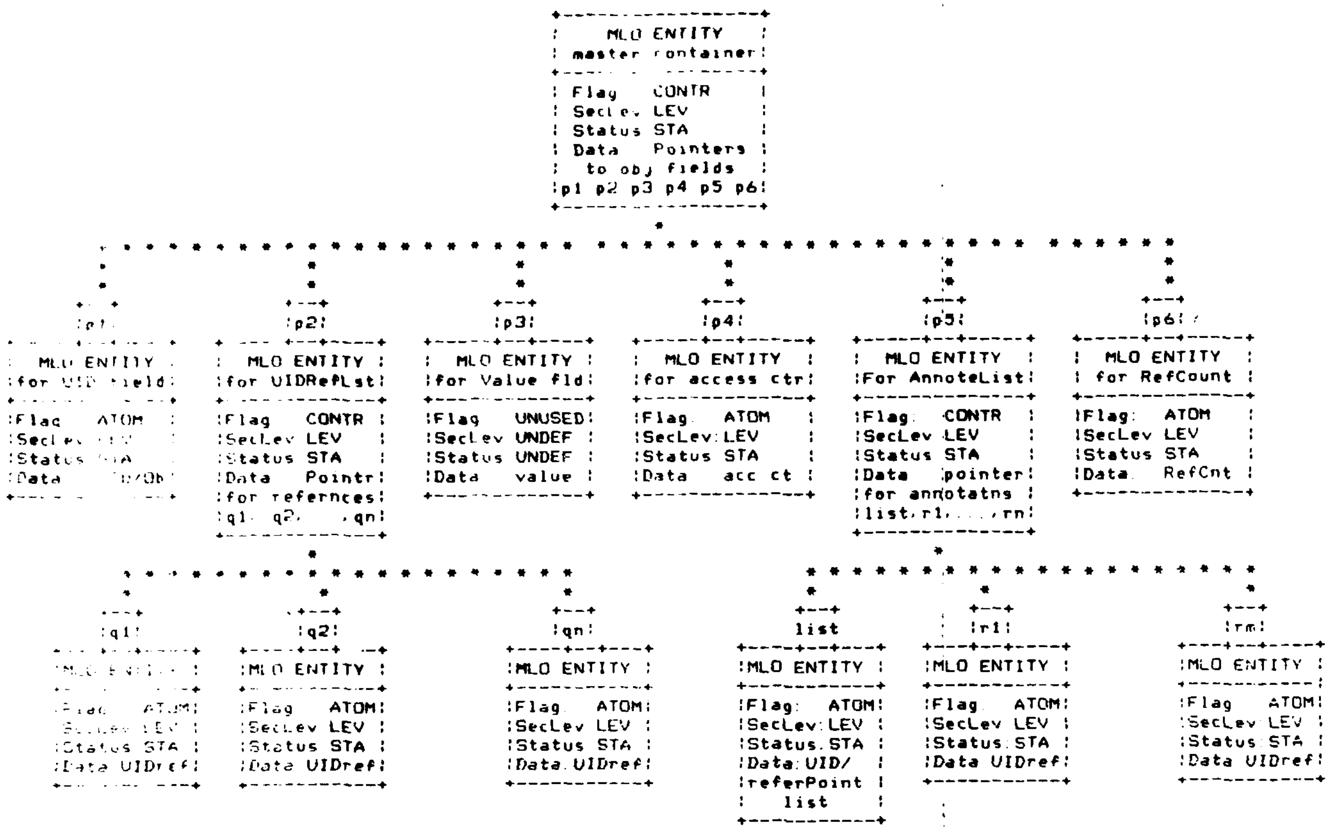


Figure 4 An MLO Implementation of a Diamond non-atomic Document (14)

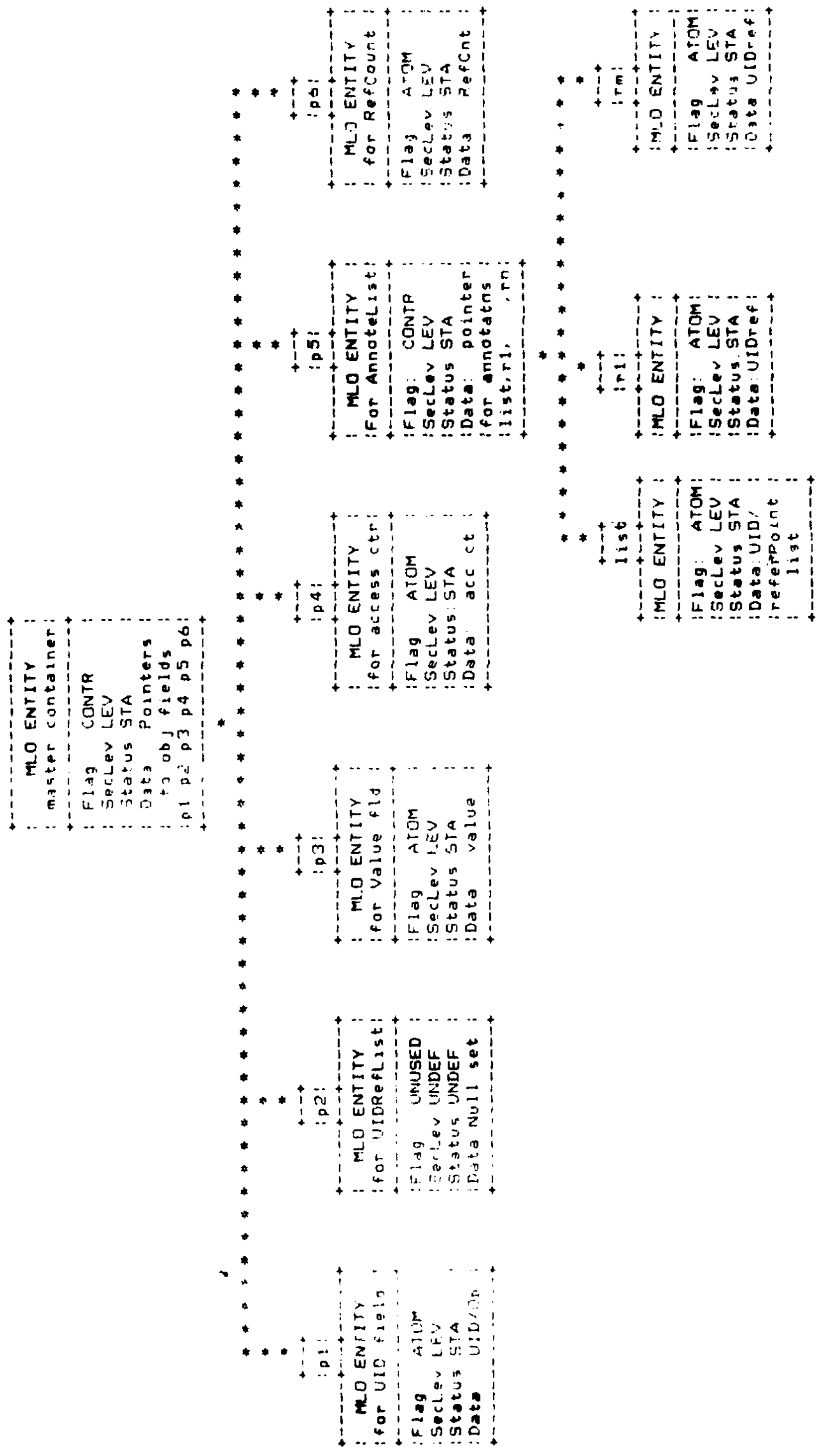


Figure 5 An MLO Implementation of an atomic-data Diamond Document (d5)

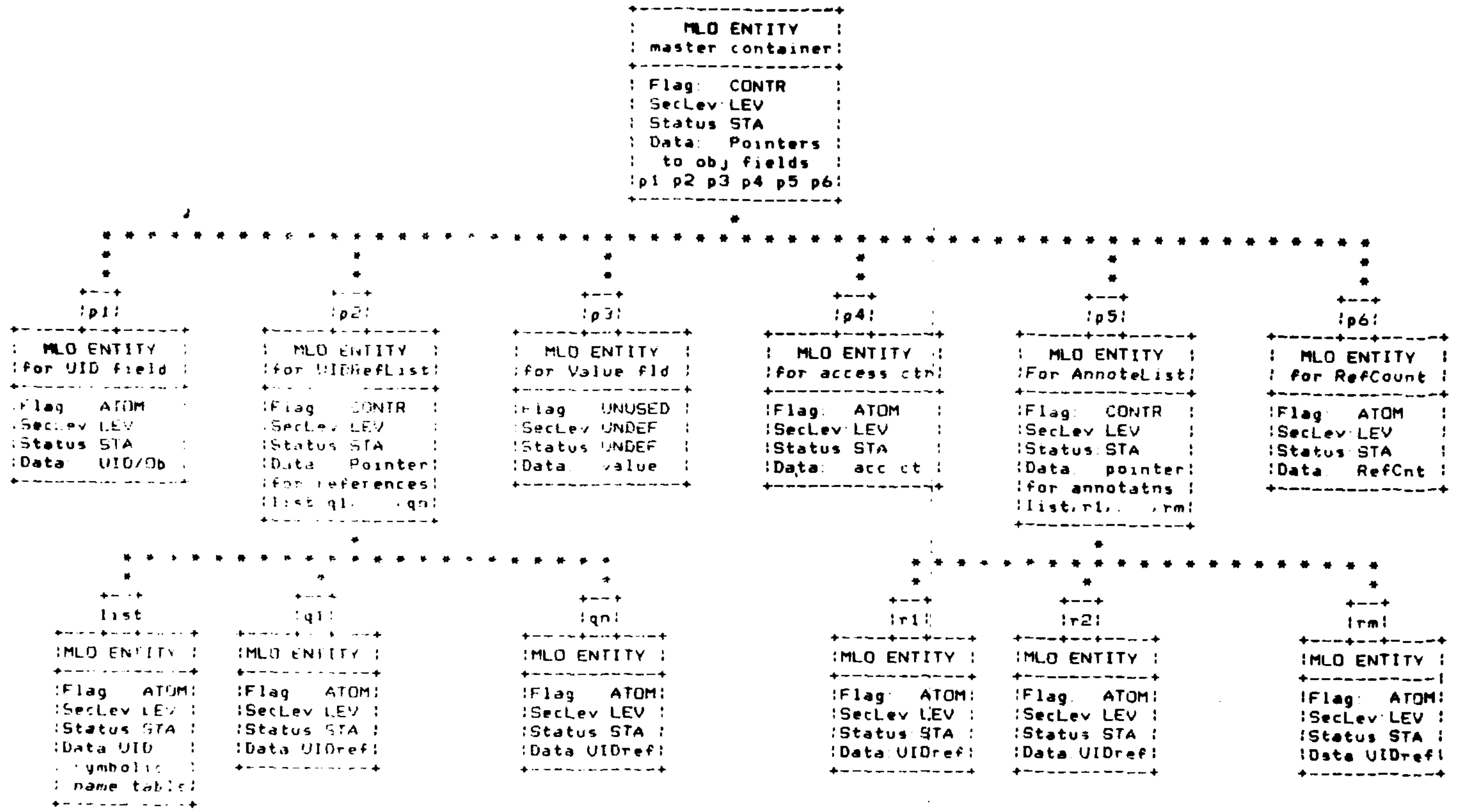


Figure 6 An MLO Implementation of a Diamond folder (Diagram 6)

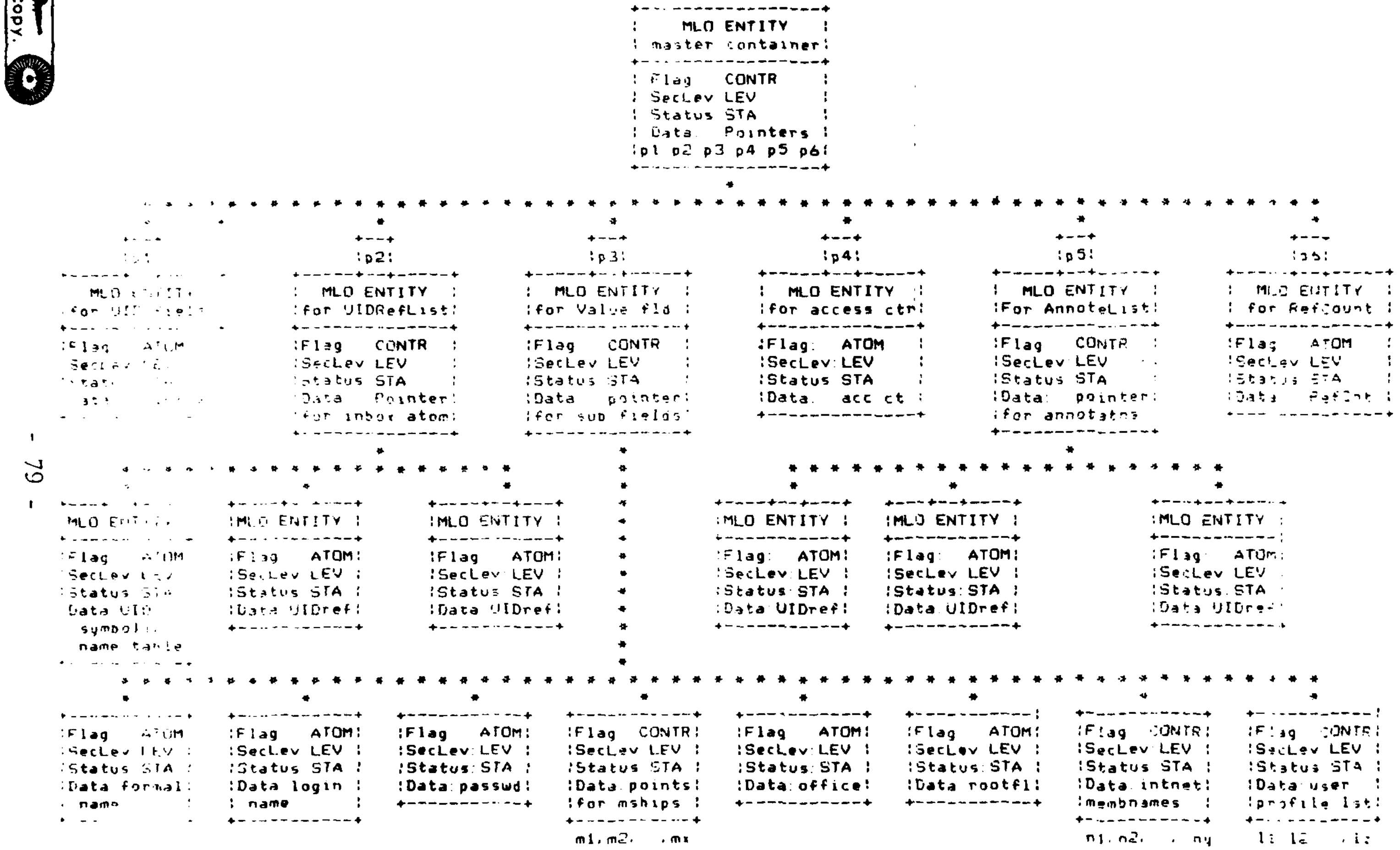
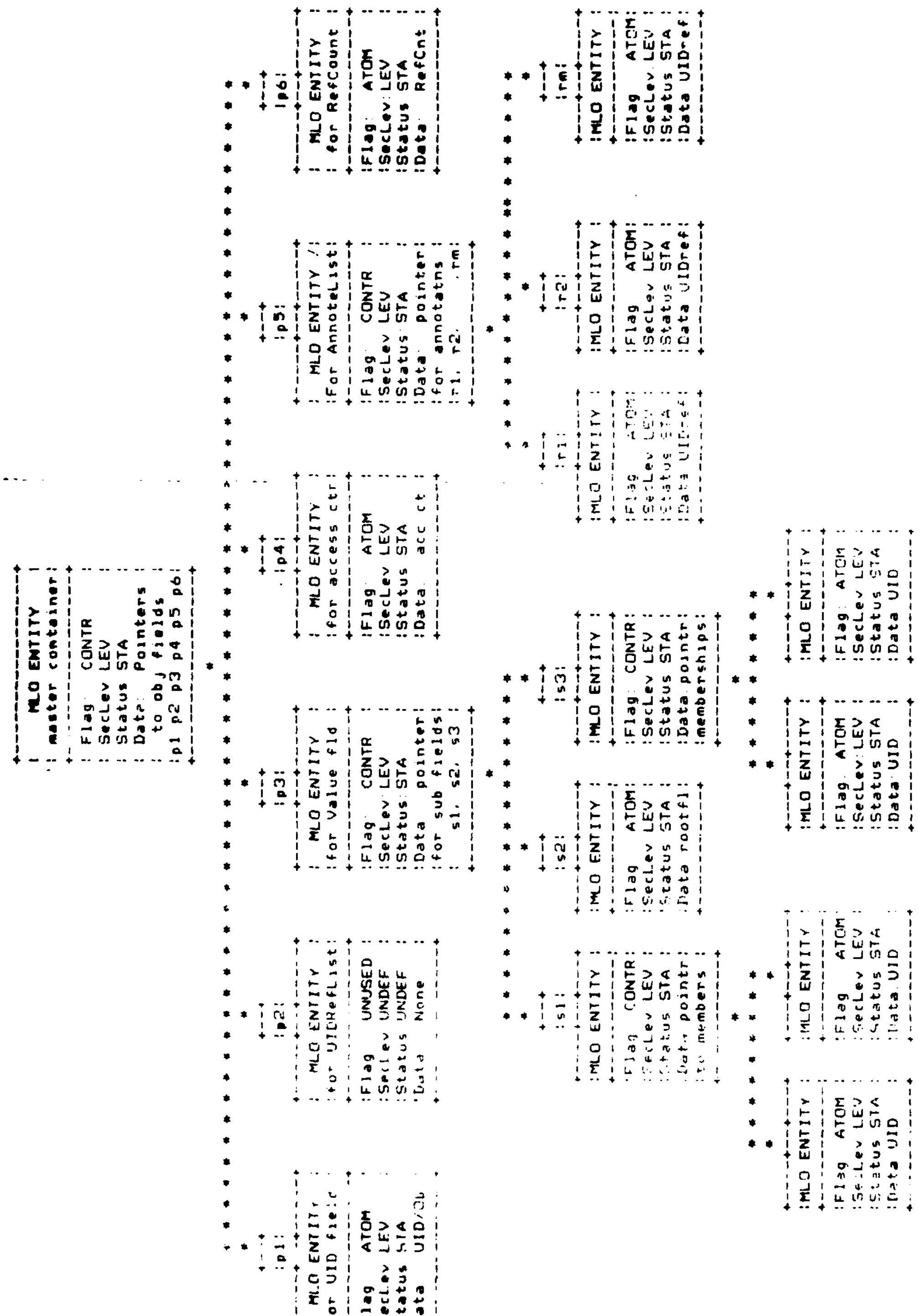


Figure 7 An MLO Implementation of a User Description Recore (d)

THE MLO MODEL--TASK 3



5. REFERENCES

[1] "Department ot Defense Trusted Computer System Evaluation Criteria", 15 August 1983.

[2] Landwehr, C.E., and Heitmeyer, C.L., "Secure Military Message Systems: Requirements and Security Model", NRL Technical Memorandum, Code 7590, April, 1982.

[3] Proctor, N. and Owre, S., "The Formal Verification of the MLO Model", Sytek Technical Report TR-85xxx, November, 1985.

[4] Owre, S. "MUSE: The Sytek Proof Processing System", SYTEK, Inc., Mt. View, Ca, July, 1985.

[5] Forsdick, H.C. and Thomas, R.H., "The Design of Diamond - A Distributed Multimedia Document System", BBN Technical Report 5204, Bolt Beranek and Newman Inc., Cambridge, Mass., October 1982.

THE MLO MODEL--APPENDIX

APPENDIX A

THE FORMALIZATION OF MULTILEVEL OBJECT SECURITY MODEL

THE MLO MODEL--APPENDIX

THE FORMALIZATION OF MULTILEVEL OBJECT SECURITY MODEL

Norman Proctor

Elisabeth C. Sullivan

SYTEK, Incorporated

```
MODULE MULTILEVEL_OBJECT_MODEL

TYPES

User_ID: PENDING;                       /* ID of a person or other
                                           external subject */
Process_ID: PENDING;                    /* ID of an internal subject
                                           acting on behalf of its
                                           sponsoring user */
Role: PENDING;                          /* a limitation of the discretion-
                                           ary rights of its sponsoring
                                           user placed on a process */
Descriptor: PENDING;                    /* ID of an entity in the
                                           database */
Reference_Table: PENDING;               /* all information in the database
                                           used by the reference mechanism
                                           to determine required container
                                           clearances */
Discretionary_Access_Table: PENDING;    /* all the information about
                                           access rights according to the
                                           discretionary policy */
DA_Segment_ID: PENDING;                 /* ID of a unit of the discretion-
                                           ary access table; the table is
                                           modified one unit at a time */
DA_Segment_Update: PENDING;             /* information used to modify a
                                           unit of the discretionary
                                           access table */
Security_Level: PENDING;                /* for example, a DoD classification
                                           with a category set */
DA_Indicators: PENDING;                 /* the part of an entity's
                                           security label that controls
                                           its discretionary access */
Atom_Data: PENDING;                     /* the information an atom has
                                           besides its security labels */
Database_Information: PENDING;          /* all information derived from
                                           the database that a process has
                                           available from previous reads;
                                           possibly also some information
                                           from the sponsor */
Edit: PENDING;                          /* information from the sponsor
                                           that a process can use when
                                           writing an entity's data */
Query: PENDING;                         /* information from the sponsor
                                           that a process can use when
                                           displaying to the sponsor */
```

```
Display: PENDING;                     /* the only information that a
                                         sponsor gets from a process */
Other_Instruction: PENDING;           /* instructions used only for
                                         communication between a process
                                         and its sponsor */
Group_Of_Users: SET_OF User_ID;       /* a fixed set of users with
                                         special privileges */
Place_Holder: {NOTHING};

Subject_Class: {SCHEDULER, USER, PROCESS};

Entity_Tag: {CONTAINER, ATOM};

Descriptors: SEQUENCE_OF Descriptor;  /* the ID's of the entities a
                                         container contains */
Entity_Data: ONE_OF(Descriptors, Atom_Data);
             /* Descriptors for CONTAINER
                Atom_Data for ATOM */

Entity: STRUCT_OF(Security_Level level;
                  DA_Indicators DA_indicators;
                  Entity_Tag tag;
                  Entity_Data data);

Flagged_Entity_Data: ONE_OF(Entity, Place_Holder);

Flagged_Entity: STRUCT_OF(BOOLEAN flag;
                          Flagged_Entity_Data data);

Access_Tag: {CREATE, WRITE, UPGRADE, DOWNGRADE, DA_INDICATORS, READ,
             DESTROY, CLEARANCE, DA_TABLE, KILL, OTHER_TAG};

Level_Change: STRUCT_OF(Security_Level old_level;
                        Security_Level new_level);

Access_Data: ONE_OF(Place_Holder, Level_Change, DA_Indicators);
             /* Level_Change for DOWNGRADE
                DA_Indicators for DA_INDICATORS
                Place_Holder for CREATE, WRITE, UPGRADE, READ or DESTROY
                other tag values do not occur */

Access: STRUCT_OF(Access_Tag tag;
                  Access_Data data);

Create_Data: STRUCT_OF(Descriptor descriptor;
                       Entity_Tag new_tag;
                       Security_Level new_level);

Write_Data: STRUCT_OF(Descriptor descriptor;
                      Edit edit);

Entity_Level_Data: STRUCT_OF(Descriptor descriptor;
                             Security_Level new_level);
```

```
DA_Indicators_Data: STRUCT_OF(Descriptor descriptor;
                              DA_Indicators new_indicators);

Simple_Entity_Access_Data: STRUCT_OF(Descriptor descriptor);

User_Level_Data: STRUCT_OF(User_ID user;
                           Security_Level new_level);

DA_Table_Change_Data: STRUCT_OF(DA_Segment_ID segment_ID;
                                DA_Segment_Update segment_update);


Instruction_Data: ONE_OF(Create_Data, Write_Data, Entity_Level_Data,
                         DA_Indicators_Data, Simple_Entity_Access_Data,
                         User_Level_Data, DA_Table_Change_Data,
                         Process_ID, Other_Instruction);
                  /* Create_Data for CREATE
                     Write_Data for WRITE
                     Entity_Level_Data for UPGRADE or DOWNGRADE
                     DA_Indicators_Data for DA_INDICATORS
                     Simple_Entity_Access_Data for READ and DESTROY
                     User_Level_Data for CLEARANCE
                     DA_Table_Change_Data for DA_TABLE
                     Process_ID for KILL
                     Other_Instruction for OTHER_TAG */

Instruction: STRUCT_OF(Access_Tag tag;
                       Instruction_Data data);

Instructions: SEQUENCE_OF Instruction;

Process: STRUCT_OF(User_ID sponsor;
                   Role role;
                   Security_Level container_clearance;
                   Security_Level data_clearance;
                   Instructions instructions;
                   Database_Information DB_info);

Flagged_Process_Data: ONE_OF(Process, Place_Holder);

Flagged_Process: STRUCT_OF(BOOLEAN flag;
                           Flagged_Process_Data data);
```

THE MLO MODEL--APPENDIX

PARAMETERS

```
Instructions
CHANGE_INSTRUCTIONS(Instructions i1;
                    Instructions i2);

BOOLEAN
CLEARANCE_CHANGE_OK(User_ID u1;
                    Role r;
                    User_ID u2;
                    Level_Change c;
                    Discretionary_Access_Table t);

BOOLEAN
CONTAINS(Descriptor d1;
         Descriptor d2;
         Reference_Table t);


Database_Information
DATA_TO_READ(Entity e;
             Database_Information d);

Atom_Data
DATA_TO_WRITE(Instructions i;
              Entity_Data e;
              Database_Information d);

BOOLEAN
DB_ACCESS_OK(User_ID u;
             Role r;
             Descriptor d;
             DA_Indicators i;
             Access a;
             Discretionary_Access_Table t);

Atom_Data
DEFAULT_DATA();

DA_Indicators
DEFAULT_DA_INDICATORS(User_ID u;
                      Role r);

Display
DISPLAY(Query q;
        Process p);

BOOLEAN
DOMINATES(Security_Level s1;
          Security_Level s2);

DA_Indicators
DUMMY_DA_INDICATORS();
```

```
BOOLEAN
ENTITY_EXISTS(Descriptor d;
              Reference_Table t);

Security_Level
ENTITY_LEVEL(Descriptor d;
             Reference_Table t);

BOOLEAN
INITIAL_ACCESS_RIGHT(User_ID u;
                     Role r;
                     DA_Segment_ID s);

Security_Level
INITIAL_CLEARANCE(User_ID u);

Security_Level
NO_CLEARANCE();

Database_Information
NO_INFORMATION();

BOOLEAN
PROCESS_KILL_OK(User_ID u1;
                Role r1;
                User_ID u2;
                Role r2;
                Discretionary_Access_Table t);

Security_Level
REFERENCE_LEVEL(Descriptor d;
                User_ID u;
                Reference_Table t);

BOOLEAN
SECURITY_ADMINISTRATOR(User_ID u);

BOOLEAN
TABLE_CHANGE_OK(User_ID u;
                Role r;
                DA_Segment_ID s;
                Discretionary_Access_Table t);

Reference_Table
UPDATE_REFERENCES(Descriptor d;
                  Flagged_Entity e;
                  Reference_Table t);

Discretionary_Access_Table
UPDATE_SEGMENT(DA_Segment_ID s;
               DA_Segment_Update u;
               Discretionary_Access_Table t);
```

THE MLO MODEL--APPENDIX

DEFINITIONS

```
Flagged_Process
no_process() IS STRUCT(flag: FALSE,
                       data: NOTHING);

Flagged_Entity
no_entity() IS STRUCT(flag: FALSE,
                      data: NOTHING);

Access
access(Access_Tag tag)
        IS STRUCT(tag: tag,
                  data: NOTHING);

Access
downgrade_access(Security_Level old_level, new_level)
        IS STRUCT(tag: DOWNGRADE,
                  data: STRUCT(old_level: old_level,
                               new_level: new_level));

Access
indicators_change_access(DA_indicators new_indicators)
        IS STRUCT(tag: DA_INDICATORS,
                  data: new_indicators);

BOOLEAN
contains(Flagged_Entity entity;
         Descriptor descriptor)
        IS entity.flag
           AND entity.data.tag = CONTAINER
           AND in_sequence(descriptor, entity.data.data);

BOOLEAN
in_sequence(Descriptor descriptor;
            Descriptors sequence)
        IS IF sequence = NULL THEN FALSE
           ELSE IF FIRST(sequence) = descriptor THEN TRUE
           ELSE in_sequence(descriptor, NONFIRST(sequence));

BOOLEAN
directly_or_indirectly_contains(Descriptor container, content;
                                Reference_Table table)
        IS CONTAINS(container, content, table)
           OR (EXISTS Descriptor d:
                 CONTAINS(container, d, table)
                 AND directly_or_indirectly_contains(d, content, table));

BOOLEAN
access_for_current_step(Access_Tag access;
                        Flagged_Process process)
        IS process.flag
           AND process.data.instructions ~= NULL
           AND FIRST(process.data.instructions).tag = access;

Flagged_Process
remove_instruction(Flagged_Process process)
        IS IF process.flag
              AND process.data.instructions ~= NULL
             THEN STRUCT
                    (flag: TRUE,
                     data: STRUCT
                             (instructions:
                                NONFIRST(process.data.instructions),
                              OTHER: process.data))
           ELSE process;

BOOLEAN
valid_process(Process process;
              Security_Level sponsor_clearance)
        IS DOMINATES(sponsor_clearance,
                     process.container_clearance)
           AND DOMINATES(process.container_clearance,
                         process.data_clearance);

BOOLEAN
correct_hierarchy(Flagged_Entity container, content;
                  Descriptor content_ID)
        IS contains(container, content_ID)
           => (content.flag
               AND DOMINATES(container.data.level,
                             content.data.level));

BOOLEAN
mandatory_policy(Flagged_Process process;
                 Flagged_Entity old_entity, new_entity;
                 Reference_Table table)
        IS (FIRST(process.data.instructions).tag = CREATE
            OR DOMINATES
                 (process.data.container_clearance,
                  REFERENCE_LEVEL
                    (FIRST(process.data.instructions).data.descriptor,
                     process.data.sponsor,
                     table)))
           AND
           (IF FIRST(process.data.instructions).tag = DESTROY
              THEN TRUE
            ELSE IF FIRST(process.data.instructions).tag = READ
              THEN DOMINATES(process.data.data_clearance,
                             old_entity.data.level)
            ELSE IF FIRST(process.data.instructions).tag
                    INSET {CREATE, UPGRADE}
              THEN DOMINATES(new_entity.data.level,
                             process.data.data_clearance)
            ELSE DOMINATES(old_entity.data.level,
                           process.data.data_clearance));
```

```
BOOLEAN
discretionary_policy(Flagged_Process process;
                     Flagged_Entity old_entity, new_entity;
                     Discretionary_Access_Table table)
        IS DB_ACCESS_OK(process.data.sponsor,
                        process.data.role,
                        FIRST(process.data.instructions).data.descriptor,
                        IF FIRST(process.data.instructions).tag = CREATE
                          THEN DUMMY_DA_INDICATORS()
                        ELSE old_entity.data.DA_indicators,
                        IF FIRST(process.data.instructions).tag = DOWNGRADE
                          THEN downgrade_access(old_entity.data.level,
                                                new_entity.data.level)
                        ELSE IF FIRST(process.data.instructions).tag
                                = DA_INDICATORS
                          THEN indicators_change_access
                                 (new_entity.data.DA_indicators)
                        ELSE access(FIRST(process.data.instructions).tag),
                        table);
```

THE MLO MODEL--APPENDIX

ASSERTIONS

SECTION: PARAMETER ASSERTIONS

```
 FORALL Security_Level s1; Security_Level s2; Security_Level s3:
   DOMINATES(s1, s1)
   AND ((DOMINATES(s1, s2) AND DOMINATES(s2, s1)) => (s1 = s2))
   AND ((DOMINATES(s1, s2) AND DOMINATES(s2, s3)) => DOMINATES(s1, s3));

 FORALL Security_Level s:
   (DOMINATES(NO_CLEARANCE(), s) OR DOMINATES(s, NO_CLEARANCE()))
   => s = NO_CLEARANCE();

 FORALL Descriptor d1; Descriptor d2; Flagged_Entity e; Reference_Table t:
   ENTITY_EXISTS(d2, UPDATE_REFERENCES(d1, e, t)) =
     (IF d1 = d2 THEN e.flag ELSE ENTITY_EXISTS(d2, t));

 FORALL Descriptor d1; Descriptor d2; Flagged_Entity e; Reference_Table t:
   ENTITY_LEVEL(d2, UPDATE_REFERENCES(d1, e, t)) =
     (IF d1 = d2 AND e.flag THEN e.data.level ELSE ENTITY_LEVEL(d2, t));

 FORALL Descriptor d1; Descriptor d2; Descriptor d3; Flagged_Entity e;
        Reference_Table t:
   CONTAINS(d2, d3, UPDATE_REFERENCES(d1, e, t)) =
     (IF d1 = d2 THEN contains(e, d3) ELSE CONTAINS(d2, d3, t));

 FORALL Descriptor d1; User_ID u; Security_Level s; Reference_Table t:
   REFERENCE_LEVEL(d1, u, t) = s
   => (ENTITY_LEVEL(d1, t) = s
       OR (EXISTS Descriptor d2:
              ENTITY_EXISTS(d2, t)
              AND ENTITY_LEVEL(d2, t) = s
              AND directly_or_indirectly_contains(d2, d1, t)));
```

SECTION: INVARIANTS

```
 FORALL Descriptor d:
   entity(d).flag = ENTITY_EXISTS(d, reference_table());

 FORALL Descriptor d:
   entity(d).flag
   => entity(d).data.level = ENTITY_LEVEL(d, reference_table());

 FORALL Descriptor d1; Descriptor d2:
   contains(entity(d1), d2) = CONTAINS(d1, d2, reference_table());

 FORALL Process_ID p:
   process_table(p).flag
   => valid_process(process_table(p).data,
                    clearance_table(process_table(p).data.sponsor));

 FORALL Descriptor d1; Descriptor d2:
   correct_hierarchy(entity(d1), entity(d2), d2);
```

SECTION: TRANSITION CONSTRAINTS

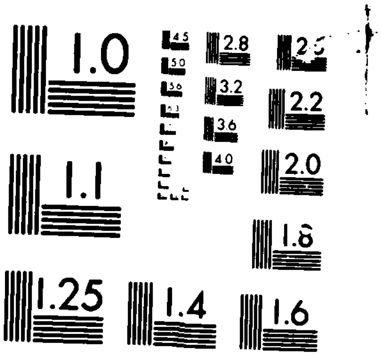
```
'latest_subject() = PROCESS
=> (process_table(current_process()).flag
    AND process_table(current_process()).data.instructions ~= NULL);

('latest_subject() = PROCESS
 AND FIRST(process_table(current_process()).data.instructions).tag
     INSET {CREATE, WRITE, UPGRADE, DOWNGRADE, DA_INDICATORS,
            READ, DESTROY})
=> mandatory_policy(process_table(current_process()),
                    entity(FIRST(process_table(current_process()).data
                      .instructions).data.descriptor),
                    'entity(FIRST(process_table(current_process()).data
                      .instructions).data.descriptor),
                    reference_table());

('latest_subject() = PROCESS
 AND FIRST(process_table(current_process()).data.instructions).tag
     INSET {CREATE, WRITE, UPGRADE, DOWNGRADE, DA_INDICATORS,
            READ, DESTROY})
=> discretionary_policy(process_table(current_process()),
                        entity(FIRST(process_table(current_process()).data
                          .instructions).data.descriptor),
                        'entity(FIRST(process_table(current_process()).data
                          .instructions).data.descriptor),
                        DA_table());
```

AD-A169 187 MULTILEVEL OBJECT SECURITY MODEL(U) SYTEK INC MOUNTAIN VIEW CA E C SULLIVAN ET AL. MAR 86 SYTEK-TR-85015 RADC-TR-86-10 F30602-85-C-0001 2/2

UNCLASSIFIED F/G 9/2 NL



MICROCOP CHART

FUNCTIONS

```
VFUN latest_subject()
        -> Subject_Class subject;
        HIDDEN;
        INITIALLY subject = SCHEDULER;

VFUN current_process()
        -> Process_ID process_ID;
        HIDDEN;
        INITIALLY TRUE;

VFUN clearance_table(User_ID user)
        -> Security_Level level;
        HIDDEN;
        INITIALLY level = (IF SECURITY_ADMINISTRATOR(user)
                             THEN INITIAL_CLEARANCE(user)
                           ELSE NO_CLEARANCE());

VFUN process_table(Process_ID process_ID)
        -> Flagged_Process process;
        HIDDEN;
        INITIALLY process = no_process();

VFUN entity(Descriptor descriptor)
        -> Flagged_Entity entity;
        HIDDEN;
        INITIALLY entity = no_entity();

VFUN reference_table()
        -> Reference_Table table;
        HIDDEN;
        INITIALLY FORALL Descriptor d1;
                         Descriptor d2:
                    ~ENTITY_EXISTS(d1, table)
                    AND ~CONTAINS(d1, d2, table);

VFUN DA_table()
        -> Discretionary_Access_Table table;
        HIDDEN;
        INITIALLY FORALL User_ID u1;
                         User_ID u2;
                         Role r1;
                         Descriptor d;
                         DA_Indicators i;
                         Access a;
                         Level_Change c;
                         DA_Segment_ID s:
                    ~DB_ACCESS_OK(u1, r1, d, i, a, table)
                    AND ~CLEARANCE_CHANGE_OK(u1, r1, u2, c, table)
                    AND ~PROCESS_KILL_OK(u1, r1, u2, r2, table)
                    AND (IF SECURITY_ADMINISTRATOR(u1)
                           THEN TABLE_CHANGE_OK(u1, r1, s, table)
```

```
              = INITIAL_ACCESS_RIGHT(ul, rl, s)
    ELSE ~TABLE_CHANGE_OK(ul, rl, s, table));
```

```
OFUN create_entity();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Descriptor
 descriptor IS step.data.descriptor;

 Entity_Tag
 new_tag IS step.data.new_tag;

 Security_Level
 new_level IS step.data.new_level;

 Flagged_Entity
 new_entity IS
    STRUCT(flag: TRUE,
           data: STRUCT(level: new_level,
                        DA_indicators: DEFAULT_DA_INDICATORS
                                        (process.sponsor, process.role),
                        tag: new_tag,
                        data: IF new_tag = CONTAINER THEN NULL
                              ELSE DEFAULT_DATA()));
ASSERTIONS

 access_for_current_step(CREATE, process_table(current_process()));

 ~entity(descriptor).flag;

 DOMINATES(new_level, process.data_clearance);

 DB_ACCESS_OK(process.sponsor, process.role, descriptor,
              DUMMY_DA_INDICATORS(), access(CREATE), DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'process_table(current_process()) =
    remove_instruction(process_table(current_process()));

 'entity(descriptor) = new_entity;

 'reference_table() = UPDATE_REFERENCES
                        (descriptor, new_entity, reference_table());
```

```
OFUN write_entity();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Descriptor
 descriptor IS step.data.descriptor;

 Edit
 edit IS step.data.edit;

 Entity
 old_entity IS entity(descriptor).data;

 Entity_Data
 new_data IS DATA_TO_WRITE(edit, old_entity.data, process.DB_info);

 Flagged_Entity
 new_entity IS STRUCT(flag: TRUE,
                      data: STRUCT(data: new_data,
                                   OTHER: old_entity));

ASSERTIONS

 access_for_current_step(WRITE, process_table(current_process()));

 entity(descriptor).flag;

 DOMINATES(process.container_clearance,
           REFERENCE_LEVEL
             (descriptor, process.sponsor, reference_table()));

 DOMINATES(old_entity.level, process.data_clearance);

 old_entity.tag = CONTAINER
 => (FORALL Descriptor d:
       in_sequence(d, new_data)
       => (entity(d).flag
           AND DOMINATES(old_entity.level,
                         entity(d).data.level)));

 DB_ACCESS_OK(process.sponsor, process.role, descriptor,
              old_entity.DA_indicators, access(WRITE), DA_table());

EFFECTS

 'latest_subject() = PROCESS;
```

```
'process_table(current_process()) =
   remove_instruction(process_table(current_process()));

'entity(descriptor) = new_entity;

'reference_table() = UPDATE_REFERENCES
                          (descriptor, new_entity, reference_table());
```

```
OFUN upgrade_entity();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Descriptor
 descriptor IS step.data.descriptor;

 Security_Level
 new_level IS step.data.new_level;

 Entity
 old_entity IS entity(descriptor).data;

 Flagged_Entity
 new_entity IS STRUCT(flag: TRUE,
                      data: STRUCT(level: new_level,
                                   OTHER: old_entity));

ASSERTIONS

 access_for_current_step(UPGRADE, process_table(current_process()));

 entity(descriptor).flag;

 DOMINATES(process.container_clearance,
           REFERENCE_LEVEL
             (descriptor, process.sponsor, reference_table()));

 DOMINATES(new_level, process.data_clearance)
 AND DOMINATES(new_level, old_entity.level)
 AND new_level ~= old_entity.level;

 FORALL Descriptor d:
   contains(entity(d), descriptor)
   => DOMINATES(entity(d).data.level, new_level);

 DB_ACCESS_OK(process.sponsor, process.role, descriptor,
              old_entity.DA_indicators, access(UPGRADE), DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'process_table(current_process()) =
    remove_instruction(process_table(current_process()));

 'entity(descriptor) = new_entity;
```

```
'reference_table() = UPDATE_REFERENCES
          (descriptor, new_entity, reference_table());
```

```
OFUN downgrade_entity();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Descriptor
 descriptor IS step.data.descriptor;

 Security_Level
 new_level IS step.data.new_level;

 Entity
 old_entity IS entity(descriptor).data;

 Flagged_Entity
 new_entity IS STRUCT(flag: TRUE,
                      data: STRUCT(level: new_level,
                                   OTHER: old_entity));

ASSERTIONS

 access_for_current_step(DOWNGRADE, process_table(current_process()));

 entity(descriptor).flag;

 DOMINATES(process.container_clearance,
           REFERENCE_LEVEL
             (descriptor, process.sponsor, reference_table()));

 DOMINATES(old_entity.level, process.data_clearance)
 AND DOMINATES(old_entity.level, new_level)
 AND old_entity.level ~= new_level;

 FORALL Descriptor d:
   contains(entity(descriptor), d)
   => DOMINATES(new_level, entity(d).data.level);

 DB_ACCESS_OK(process.sponsor,
              process.role,
              descriptor,
              old_entity.DA_indicators,
              downgrade_access(old_entity.level, new_level),
              DA_table());

EFFECTS

 'latest_subject() = PROCESS;
```

```
'process_table(current_process()) =
   remove_instruction(process_table(current_process()));

'entity(descriptor) = new_entity;

'reference_table() = UPDATE_REFERENCES
                          (descriptor, new_entity, reference_table());
```

```
OFUN write_DA_indicators_of_entity();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Descriptor
 descriptor IS step.data.descriptor;

 DA_Indicators
 new_indicators IS step.data.new_indicators;

 Entity
 old_entity IS entity(descriptor).data;

ASSERTIONS

 access_for_current_step(DA_INDICATORS, process_table(current_process()));

 entity(descriptor).flag;

 DOMINATES(process.container_clearance,
           REFERENCE_LEVEL
             (descriptor, process.sponsor, reference_table()));

 DOMINATES(old_entity.level, process.data_clearance);

 DB_ACCESS_OK(process.sponsor,
              process.role,
              descriptor,
              old_entity.DA_indicators,
              indicators_change_access(new_indicators),
              DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'process_table(current_process()) =
    remove_instruction(process_table(current_process()));

 'entity(descriptor) = STRUCT(flag: TRUE,
                              data: STRUCT(DA_indicators: new_indicators,
                                           OTHER: old_entity));
```

```
OFUN read_entity();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Descriptor
 descriptor IS step.data.descriptor;

 Entity
 entity IS entity(descriptor).data;

ASSERTIONS

 access_for_current_step(READ, process_table(current_process()));

 entity(descriptor).flag;

 DOMINATES(process.container_clearance,
           REFERENCE_LEVEL
             (descriptor, process.sponsor, reference_table()));

 DOMINATES(process.data_clearance, entity.level);

 DB_ACCESS_OK(process.sponsor, process.role, descriptor,
              entity.DA_indicators, access(READ), DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'process_table(current_process()) =
    STRUCT(flag: TRUE,
           data: STRUCT(DB_info: DATA_TO_READ(entity, process.DB_info),
                        instructions: NONFIRST(process.instructions),
                        OTHER: process));
```

```
OFUN destroy_entity();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Descriptor
 descriptor IS step.data.descriptor;

 Entity
 old_entity IS entity(descriptor).data;

ASSERTIONS

 access_for_current_step(DESTROY, process_table(current_process()));

 entity(descriptor).flag;

 DOMINATES(process.container_clearance,
           REFERENCE_LEVEL
             (descriptor, process.sponsor, reference_table()));

 FORALL Descriptor d: ~contains(entity(d), descriptor);

 DB_ACCESS_OK(process.sponsor, process.role, descriptor,
              old_entity.DA_indicators, access(DESTROY), DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'process_table(current_process()) =
    remove_instruction(process_table(current_process()));

 'entity(descriptor) = no_entity();

 'reference_table() = UPDATE_REFERENCES
                        (descriptor, no_entity(), reference_table());
```

```
OFUN change_clearance();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 User_ID
 user IS step.data.user;

 Security_Level
 new_level IS step.data.new_level;

 Level_Change
 level_change IS STRUCT(old_level: clearance_table(user),
                        new_level: new_level);

ASSERTIONS

 access_for_current_step(CLEARANCE, process_table(current_process()));

 FORALL Process_ID p:
   process_table(p).flag
   => process_table(p).data.sponsor ~= user;

 CLEARANCE_CHANGE_OK
   (process.sponsor, process.role, user, level_change, DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'clearance_table(user) = new_level;

 'process_table(current_process()) =
    remove_instruction(process_table(current_process()));
```

```
OFUN change_discretionary_access_table();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 DA_Segment_ID
 segment_ID IS step.data.segment_ID;

 DA_Segment_Update
 segment_update IS step.data.segment_update;

ASSERTIONS

 access_for_current_step(DA_TABLE, process_table(current_process()));

 TABLE_CHANGE_OK(process.sponsor, process.role, segment_ID, DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'process_table(current_process()) =
    remove_instruction(process_table(current_process()));

 'DA_table() = UPDATE_SEGMENT(segment_ID, segment_update, DA_table());
```

```
OFUN spawn_process(Process_ID process_ID;
                   Role role;
                   Security_Level container_clearance;
                   Security_Level data_clearance)
                  [User_ID user];

DEFINITIONS

 Process
 new_process IS STRUCT(sponsor: user,
                       role: role,
                       container_clearance: container_clearance,
                       data_clearance: data_clearance,
                       instructions: NULL,
                       DB_info: NO_INFORMATION());

ASSERTIONS

 ~process_table(process_ID).flag;

 DOMINATES(clearance_table(user), container_clearance);

 DOMINATES(container_clearance, data_clearance);

EFFECTS

 'latest_subject() = USER;

 'process_table(process_ID) = STRUCT(flag: TRUE,
                                     data: new_process);
```

```
OFUN kill_process();

DEFINITIONS

 Process
 process IS process_table(current_process()).data;

 Instruction
 step IS FIRST(process.instructions);

 Process_ID
 ending_process_ID IS step.data;

 Process
 ending_process IS process_table(ending_process_ID).data;

ASSERTIONS

 access_for_current_step(KILL, process_table(current_process()));

 process_table(ending_process_ID).flag;

 ending_process_ID = current_process()
 OR PROCESS_KILL_OK(process.sponsor,
                    process.role,
                    ending_process.sponsor,
                    ending_process.role,
                    DA_table());

EFFECTS

 'latest_subject() = PROCESS;

 'process_table(ending_process_ID) = no_process();

 ending_process_ID ~= current_process()
 => 'process_table(current_process())
    = remove_instruction(process_table(current_process()));
```

```
OVFUN display_to_user(Process_ID process_ID;
                       Query query)
                      [User_ID user]
                      -> Display display;

DEFINITIONS

 Process
 process IS process_table(process_ID).data;

ASSERTIONS

 process_table(process_ID).flag;

 process.sponsor = user;

EFFECTS

 'latest_subject() = USER;

 display = DISPLAY(query, process);
```

```
OFUN instruct_process(Process_ID process_ID;
                      Instructions new_instructions)
                     [User_ID user];

DEFINITIONS

 Process
 process IS process_table(process_ID).data;

ASSERTIONS

 process_table(process_ID).flag;

 process.sponsor = user;

EFFECTS

 'latest_subject() = USER;

 'process_table(process_ID) =
    STRUCT(flag: TRUE,
           data: STRUCT(instructions: CHANGE_INSTRUCTIONS
                                        (process.instructions,
                                         new_instructions),
                        OTHER: process));
```

```
OFUN schedule_process(Process_ID process_ID);
EFFECTS
 'latest_subject() = SCHEDULER;
 'current_process() = process_ID;
END_MODULE
```

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

DTIC

1-86