8-12-1985

# Access control models: Authorization mechanisms for database management systems

Diana Anglero

Rochester Institute of Technology
School of Computer Science and Technology

# Access Control Models:
# Authorization Mechanisms for Database Management Systems

by
Diana Angleró

A Thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by: ______________________________
Professor Henry A. Etlinger

______________________________
Professor Jeffrey Lasky

______________________________
Professor Peter G. Anderson

Date: August 12, 1985

## Table of Contents

## 1. Introduction

Data security is a major concern in computer systems today. Data security is a crucial issue in data management systems (DBMS), more important than in any other type of software. While the database approach brings advantages to the user, it also creates new, more intense problems in the areas of security and integrity. In the traditional approach, where each application system has its own files, a limited amount of data sharing is achieved by passing files from one system to another. Typically a database contains data of various degrees of importance and levels of sensitivity. This data is shared among a wide variety of users with different responsibilities and privileges. Therefore it becomes important to restrict database users to those portions of the total data that are necessary to their activities. Additionally, more control is needed over what changes a user can make to data because of the many ways these changes can affect other users of the database. This increased level of sharing means that access to the data must be controlled in order to ensure security and privacy.

Maintaining security of the database can be viewed as protecting the data against illegal or invalid retrieval, modification, or destruction. This may be achieved by a set of access control rules. The effectiveness of access controls rests on two premises. The first is proper user identification: no one should be able to acquire the access rights of another. This premise is met through authentication procedures at login. The second premise is that information specifying the access rights of each user or program is protected from unauthorized modification. This premise is met by controlling access to system objects as well as to user objects.

This paper presents various models to enforce access control rules. First some conceptual ideas are introduced. Then an exposition of the models used most often is made. These include access matrix, authorization list (access list), capability list, take-grant and query modification. Finally, the protection mechanisms of three systems, INGRES, DB2, and System/38 are analyzed. These three systems were chosen because they represent three different approaches. The first deals with the problem by using query-modification and access control lists; the second uses a combination of access control lists, a hierarchy and view mechanisms to provide security; abd the third uses capability list and enforces access control at a most lower level, at the hardware, microcode level.

# 2. Concepts

## 2.1. Authentication Methods

The first step in a security mechanism of any kind is to identify the person who is trying to gain access to or use the object one is guarding. In a database management system where a high level of data sharing is implied, this identification becomes essential. Various methods for verifying the identity of users, or authenticating them, will be explored.

Before discussing authentication methods, it is necessary to understand the interactions between identification, authentication, and authorization, all of which together determine what access is granted to protected resources. Identification is a unique name or number assigned to subjects (users); authentication verifies that a person or subject is who he, she or it claims to be; and authorization is whether a person or subject is legitimately entitled to a protected resource. All of these are used together to make access decisions.

### 2.1.1. Identification

In our context a user can be a person, any member of a group of persons associated with some common project or assignment, any member of a category of persons sharing some common attribute, or a program acting on behalf of a person, group or category of persons.

The identification of a user, program or other subject is the unique name or number assigned to that subject. It is only a "claim" of identity. Identification is necessary for accounting and authorization purposes, but cannot be used without additional authentication if some degree of security is desired in a system.

### 2.1.2. Authentication

Authentication verifies that a person (subject) is who he, she or it claims to be. There are several types of information which may be required before an identification is accepted as valid. While authentication is generally done once, periodic verifications may be required in high security installations, and reauthentication may be desirable after all system crashes.

There are three approaches to authentication of identity and they invoke something the user "knows", something the user "has" or something the user "is". Authentication by something the user knows is the cheapest scheme and currently the most widely used. The technique most commonly used is a password.

The password method requires the user to provide a string of characters for the computer to check. If the password matches the one that the computer already associates with a given user, access is permitted to all the information that is authorized to that user.

In a simple password scheme the user is allowed to choose a password that is easy for him to remember. The advantage of this scheme is that the user should not need to write it down where someone can see it. But care must be taken not to choose a password which is too obvious, or one which anyone who has knowledge of the user can determine with a few guesses.

A way to avoid this is to assign the user a randomly generated password. The problem here is that such a password could be difficult to memorize and the user will tend to write it down. One solution to this is to provide a generator of "pronounceable" random passwords. Multics [SALT74], for example, generates an eight-character password which has English-like characteristics, so it is both pronounceable and easy to memorize.

Another technique sometimes used is to put spaces and backspaces in the password [HOFF77]. This will prevent anyone who finds a password written in a piece of paper to use it correctly.

The password schemes mentioned above are all susceptible to wiretapping. There are three ways to overcome this: to use cryptographic techniques, one-time passwords and transformation techniques. For the one-time method, the user is given a list of N passwords; the same N passwords are stored in the computer. After using a password for login the user crosses it off the list. The next login requires use of the next password on the list. But this approach may not be feasible on highly used systems, where either there are many users or each user signs on quite often. Also, if used, the user must remember or carry the entire password list and must also keep track of the current password.

The transformation technique consists of the user remembering and performing an algebraic transformation on a string of random digits. The computer supplies the random digits. The user does the transformation and returns the answer. The computer also does the transformation and compares the answers. If they agree the authentication is successful and the user is allowed access.

Another method used to verify the identity of a user is through the question-answer method. A set of answers to "m" standard and "n" user supplied questions is stored in the computer. When the user attempts to login, some (or all) of these questions are chosen at random and asked by the system. The user must answer all the questions correctly in order to be granted access to the system.

The techniques discussed above fall under the category of something the user "knows". But as already mentioned, the authentication could also be made through something the user "has". The most common in this category are badges and cards. Cards may contain optical bar codes or a Hollerith code. Plastic cards with a magnetic strip or implanted magnetic slug may also be used [HSIA79]. These cards can be inserted into a terminal for identification. They can be designed to resist forgers and, although they can be given to others or be lost or stolen, their possession can be made mandatory and is easy to check. For example, the cards may be assigned additional functions such as operating a card-key lock to gain access to the terminal room.

The use of personal characteristics, or something the user "is", is the most expensive way of authenticating users and, to date, the least developed [WAKE77]. Some of the techniques explored are physical appearance like anatomical measures (weight, height) and hand geometry; voice recognition, where the relative energy content of each of several voice frequencies is measured (the resulting frequency profile is matched against those of persons seeking access); signature verification; and fingerprint analysis.

All these techniques involve the use of complex devices, and considerable processing time and storage space may be required. At present they are considered too costly for general applications, but future advances in hardware technology may facilitate their use.

### 2.2. Entities

All the access control models or mechanisms to be discussed try to explain the relationship between users, the data stored in the database and the privileges of access the users have over the data. As a mode of generalization, these three components will be referred to as subjects (s), objects (o), and privileges (p). Other components may be included in some of the models (e.g. flags, predicates), but these three are the most essential and are expressed in some way or another in all models. The choice of the subjects, objects, and privileges or rules of a protection system is at the discretion of the system designer and will be made to meet the protection and sharing requirements of the given system.

#### 2.2.1. Subjects

A subject is any person or other entity that can request access to a protected object. The most important subjects are the "users", particularly the end users, of the database. There may be a directory of users, which may contain user profiles that describe attributes of users. Such attributes can define access rights or default authorization characteristics. This information is also used for evaluation of requests or of content-dependent access rules.

There are two types of subjects, one is the user per se and the other the administrator. Users are the subjects that usually provide the data to be stored in the database or that request data from the database in order to make further decisions. The end user is the class of user that is of the most concern with regard to security, because the end user is usually separated from the system and therefore less subject to physical, personal, or procedural security. Some end users may be limited to specific data because of the organization they work for, or the organization from where the data originated.

On the other hand there is an administrator of the database (one person or a group of persons), who has such responsibilities as creating the databases of the system (though not for placing data in them) or determining the name, number, and size of the various files or relations to be placed in the permanent databases. The administrator is also responsible for maintaining and reorganizing the database so that it performs in the most efficient manner. He may be responsible for defining the security system, the protection data and the enforcement rules. This last responsibility may also be assigned to a different person such as a System Security Officer.

The separation of these two kinds of users is not always obvious. In a small environment there may be no such distinction. But then, further problems may arise, like the conflict of ownership (this will be discussed later in the section of Administrative policies).

Distinguishing users from administrators may make the task of security enforcement much easier. The authorizer's job may be simplified even more by the definition of "groups" or "classes" of users. When a user group is given access rights, all users belonging to the group receive those rights. Subsequently, a user who joins a group automatically receives all the rights of the group.

A user group is simply a named collection of users, each of which has previously been defined to the system. Consider the following example:

```
DEFINE USERS (SMITH, DOE, JONES)
DEFINE GROUP (PROJ1)  USERS (SMITH, DOE)
DEFINE GROUP (PROJ2)  USERS (DOE, JONES)
```

In the example, two user groups called PROJ1 and PROJ2 are defined, PROJ1 consists of the users SMITH and DOE, and PROJ2 consists of the users DOE and JONES. When the user JONES enters the system he may access any objects authorized either to JONES or to PROJ2.

The concept of user groups is extremely useful for implementing organizational concepts, such as department or project, within the security framework. The implementation of the concept of user groups should require the user to identify his group membership before the execution of his job begins. If a user who is a member of one or more groups enters a job with no group identification he should be granted access only to those objects explicitly authorized to him. In this way the user may, for example, test a program on his files without endangering files belonging to the entire group.

One can also treat transactions or application programs as subjects and control their access to data. An application is a set of transactions or programs that accomplish related functions. If users belong to more than one group or use more than one application, their rights at any moment may depend on both their identity and the group or application. One way of expressing this requirement is to use the group name or application name in the predicate of an access rule. For example SMITH may be allowed access to SALARY only when using the PAYROLL application. Another way is to make the subject a "component", such as (user, group) or (user, application)

[FERN81]. The composite's rights can be granted explicitly or can be calculated from individual and group rights.

### 2.2.2. Objects

Objects are the things in the system which have to be protected. Some of the most important concerns in authorization have to do with the choice of objects to protect. Some of the points to consider are:

- The level of the objects to be protected (external schema, conceptual schema or internal schema).
- The size of the unit of protection (file, record, field).
- Protection through view, if the concept of view exists in the system.
- Protection of data description. Some systems separate data from description of data.

(a) Level of objects

The protected objects are at all levels (external, conceptual and internal), but the access rules should be specified at the conceptual level. This is so because access rules specified for the conceptual level apply regardless of how those objects are viewed or used by different applications. If there is a need to specify external access rules these should be consistent with the rules derived from the conceptual level access rules.

(b) Granularity

A commonly used level of granularity is the file level (relation, or record type). Sometimes a larger granularity will be sufficient, and groups of files are protected as a single object. The groups can be defined in various ways such as: a subtree of a hierarchical directory structure, an explicit naming of group members, storage in the same physical area or common name portions in a multipart name.

Any number of defined files could be collected within, a named group, and a defined file could be allowed to be a member of more than one group. For example:

```
DEFINE FILES (ABC, DEF, XYZ)
DEFINE GROUP (PAYROLL)  FILES (ABC, DEF)
DEFINE GROUP (PERSONNEL)  FILES (DEF, XYZ)
```

The concept of file groups is extremely useful to accommodate the notion of departmental files, the files of an individual, and the like.

But file level control does not always suffice. As will be explained later (Policies section) there is a need in many situations for policies of field-level control and content-dependent control. Control could be specified at the record or field level as needed. One approach to field-level access control is to write rules about "field types", and to derive from them the rules about columns or fields. The problem with this approach is that the same field type may have a different meaning in different relations, and automatic derivation would be inappropiate. For example, suppose that a field type named SALARY appears in two relations: EMPLOYEE (NAME, SALARY) and LIMIT(JOBNAME, SALARY). A personnel department employee might need access to SALARY in the LIMIT relation but should not be allowed to know individual salaries. Field-level access rules therefore must have as their objects the fields of a file or relation.

(c) Views as objects

One way to provide authorization with finer granularity is to define tailored views for different users according to their needs and then control access to the views. Because the views can be used to create a subset of the rows of a relation, a subset of the columns of a relation or a subset of a combination of rows and columns, the view can be used effectively to screen parts of a table that a user should not see.

There may be some restrictions on the way a user can manipulate the data accessible to him through a view. Even if he has some rights to the views he creates these may be limited. Certain operations, for example, creation of an index, or update to a "statistical" view, may not be performed on the view. A user's rights are also limited to the ones he possesses regarding the underlying relation from which he creates new views. For example, if he has read-only access to a relation he should have read-only access to any view he defines on top of it. If the view involves more than one underlying relation, the user's privileges may be constrained by the intersection of the privileges which he holds on the underlying relations.

(d) Data Description as objects

The description of an object and the values of that object are conceptually distinct, and access to them could be separately controlled. The security control of data description should be assigned to an administrator rather than to end-users.

Transactions and programs were considered subjects, receiving access rights according to their purposes. But it is also necessary to control access to them, since they indirectly provide access to data objects. Users could be authorized RUN or EXECUTE privileges to programs or transactions.

### 2.2.3. Privileges

The third component of the protection system is the privilege, expressed as a rule, which determine the accessing of objects by subjects. The rules are the heart of the access control mechanism of the database management system. The rules must be simple, allowing users to gain an immediate understanding of their scope and use. They must be complete, not allowing a subject to gain unauthorized access to an object. They must be flexible, providing mechanisms which easily allow the desired degree of authorized sharing of objects among subjects. They must be easy to change so as to prevent the user from avoiding them or giving more access than necessary.

(a) Access types

In the database there is a set of operations which can be performed on each category of objects. These allowable operations form a set of possible access types, and access rules grant a subset of them to specific subjects. Some of the categories in which the rules can be based are:

(1) Zero-level access

The subject can do nothing with the object. Most probably it implies that his authority has been revoked.

As far as the subject is concerned that object does not exist for him any more.

(2) Execute-only access

This access rule applies to programs or transactions. It means the subject can execute the program but he will not be permitted to read the instruction sequence or to modify it in any way. It could be specified with commands like RUN or EXECUTE.

(3) Read-only access

The subject is allowed only to retrieve the object. He is not permitted to manipulate or alter data in any way. The subject could be allowed to use a command like READ, or to apply statistical operators such as AVErage.

(4) Read-write access

With this privilege the subject is allowed to update or change the information in the object, through commands like UPDATE, WRITE, INSERT.

(5) Create objects

The subject is granted the privilege to create objects and may also have the privilege to delete them (DEFINE, CREATE and DELETE or DROP). The subject may then be assigned an "owner" privilege over those objects he creates, which allows him to grant some privileges to other users over the objects he "owns".

(6) Grant and Revoke

A subject who owns an object (as mentioned above) or that was granted some privileges over a given object may have the right to grant some privileges to other users. If he does not own the object he must have been granted some privileges with the option to "grant" them. Typically a subject grants to others a set of the privileges he possesses. These may include:

READ - the ability to read the object and the ability to define views based on the table (if it is a relational system).

INSERT - ability to insert records or rows in the file or relation.

DELETE - ability to delete records from the file.

UPDATE - ability to modify existing data in the file.

GRANT - the ability to further grant privileges to others.

Depending on the way the rules are to be implemented the GRANT privilege could be specified by a "copy flag" or by a command like:

```
       ┌                         ┐
       │ ALL RIGHTS              │
GRANT <  <privileges>             > ON <object> TO <user-id>
       │ ALL BUT <privileges>    │
       └                         ┘
                                    [WITH GRANT OPTION]
```

REVOKE is a privilege which usually goes together with GRANT. It is the privilege that gives the holder the ability to revoke the authorities granted. Only the grantor may revoke access, so REVOKE implies GRANT.

The problem with revocation is its propagation. Consideration should be given to the way it is going to affect privileges of other users. For example, if a user Y revokes some privileges from user X in a lower level on the hierarchical structure, other users below X to whom X granted some privileges could loose all their privileges. Another approach is to let them keep the privileges and make Y responsible and decide which of the privileges, if any, should also be revoked.

As previously discussed, subjects may be separated into end-users and administrators. If the security structure one wants to implement calls for the specification of an administrator, then the mechanism must provide to these administrators a set of "control" or "administer" privileges to help them achieve their function.

The most common rights given to an administrator are GRANT and REVOKE. Others could be the rights to DEFINE and DELETE objects (e.g databases, relations, files). Once he creates objects, as databases, he can delegate rights to users through the GRANT right, and he then has the authority to remove these rights from them if necessary. Depending on the structure (e.g. hierarchy) and needs of the installation, administrators may not receive rights to actually manipulate the data in the databases (READ, INSERT, UPDATE). In this way his function is limited only to CONTROL or ADMINISTER objects in the database.

(b) The use of predicates for control

Finer granularity could be desired when specifying access rules. An access decision may need to be based in more than just controlling access to a record or field. It may be necessary to consider the value of the fields. The content-dependent and context-dependent policies, to be discussed later, are some of the situations which require this type of control. For content-dependent access control the authorizer defines predicates whose evaluation depends on the contents of the database or on system variables such as time of day or terminal ID. The predicate

DEPT = "MATH" AND TIME-OF-DAY > "1200"

would allow access only to the MATH department records and only after noon.

Suppose one wishes to restrict a certain set of users from accessing the field EMPNO together with SALARY (context-dependent access control). In other words, although the users have separate access to both fields, they should not be able to deduce the salaries of individual employees. One could write this constraint as

NOT (EMPNO, SALARY)

in the rule specifying access to the EMPLOYEE relation. Any access rules specified for views defined on the EMPLOYEE relation then have to be checked for consistency with this constraint. Since NAME also uniquely identifies the employee the constraint should be extended as:

NOT (EMPNO, SALARY) AND NOT (NAME, SALARY)

The language used to express these predicates usually provides for key words like WHERE. For example a predicate could test a data value, WHERE (COURSE.DEPT = 'MATH'), or test for the value of some system variable, such as WHERE (SYSTEM.USER_ID = 'JONES').

Predicates are the mechanism used on the DBMS that utilize views to enforce access rules. A predicate is used in the definition of the view to select specific records or fields to be included in the user view. For example:

```
        DEFINE MY_DEPT VIEW AS:
               SELECT EMP,  LOC
               WHERE  EMP.DEPT = LOC.DEPT
               AND    EMP.MGR = USER
```

This view is built from the join of two base relations EMP and LOC. It allows one to see the name, salary, manager, department and location of each employee who reports directly to the user of the view.

(c) Authorization time

Some of the access rules could be enforced at compilation-time. But others like the content-dependent and context-dependent rules have to be enforced at execution time. These rules require that the data be read and evaluated. It is not until the values of the data are retrieved that the decision to grant or deny access can be taken.

(d) Authorization by classification levels

In some installations such as military or government ones, information is classified and users are assigned a clearance according to their rank or position in a hierarchical organization. Typical classification levels might be top secret, secret, confidential and unclassified. Objects are grouped in a set of categories. Access is defined by a combination of categories and classification levels. There are two fundamental security rules: (1) no user can access (read) an object if the object's classification level is greater than the user's clearance level, and (2) only specially authorized users may reduce (downgrade) or remove categories from a classified object.

## 2.3. Security Principles

In order to understand any mechanism or model built to enforce access control rules, it is convenient to point out the principles behind those mechanisms. There is a series of policies which serve as guidelines for the development of security mechanisms. These policies are given by user needs, the installation environment, institutional regulations, and legal constraints. The principles to be discussed are divided into three groups: design policies, administration policies and access-control specification policies.

### 2.3.1. Design Policies

Saltzer and Schoeder identified several design principles for protection mechanisms [SALT75]:

(a) Least privilege

Every user and process of the system should have the least set of access rights necessary to complete the job.

This principle limits the damage that can result from error or malicious attack. It implies that processes should execute in "small protection domains" [DENN82], consisting of only those rights needed to complete their tasks. It also reduces the number of interactions among programs. It reduces the number of possible sources for information leaks and minimizes the possibility that the integrity of the database be violated. Thus, if a question arises related to misuse of a right, the number of programs to be audited is minimized. This principle is similar to the military security rule of need-to-know.

(b) Economy of mechanism

The design should be kept as simple and small as possible so that it can be verified and correctly implemented. Although the system must be sufficiently flexible to handle a variety of protection policies, it is better to implement a simple mechanism that meets the requirements of the system than it is to implement one with complicated features that may not be used.

(c) Complete mediation

Every access to every object must be checked for authority. This principle is the basis of the protection system. It implies that a method of identifying the source of every request must be devised.

(d) Open vs. closed design

The design should not be secret. Security should not depend on the ignorance of the intruder, but rather on the possession of specific, more easily protected keys or passwords. This separation of protection mechanisms from protection keys permit the mechanism to be examined without concern that the safeguard will be endangered.

This principle becomes important also in a decentralized system, as it may not be realistic to maintain secrecy in such a system.

(e) Least common mechanism

However, distributes security enforcement, so that no one system function has major responsibility for all forms of security enforcement.

(f) Fail safe

Base access decisions on permission rather than exclusion. That is, access should be allowed only if "explicitly authorized" instead of being allowed "unless" explicitly forbidden. So the default situation is lack of access. In this way a design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation, since it will be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure which may go unnoticed in normal use.

(g) Psychological acceptability

The mechanism must be easy to use so that it will be applied correctly. It should provide a convenient interface for the user (or DBA) to permit an easy, consistent, and efficient way of defining and maintaining the protection data.

### 2.3.2. Administration Policies

(a) Ownership vs. Administration

The concept of ownership is important when dealing with the transfer of privileges. A user can grant and deny other users access to a data object if he is the owner of the object. The owner of a database is sometimes considered the person responsible for creating the data. To become the owner of an object, the user must satisfy and agree with a set of rules. At minimum he must have been assigned the privilege to create objects. When a user becomes an owner of an object, the system assigns an "own" attribute for the object as part of that user's privileges.

However, with many shared databases, it may be difficult to identify a unique owner. Even if there is or is not a concept of ownership, there is always the need for an administrative function, whose objective is to define the data shared by the users and to control its use.

There could be then an administrator who "owns" all databases, can perform all data management operations, may authorize other users to use the databases and is subject to further administrative rules. The owners, on the other hand, own private databases, can perform all data management operations on their own databases, and may authorize other users to access their databases. There may also be some non-owner users, who do not own a database, can perform only some data management operations and may not authorize other users to use the databases.

In addition to this three-level hierarchy Hsiao identifies two other types of organizations: a multi-level sub-ownership authorization hierarchy, and two-level transfer ownership authorization hierarchy [HSIA78].

The multi-level sub-ownership hierarchy has the following structure:

(1) Each database has an owner. The owner of a database may assign a portion of the database to a user and designate him as a subowner. The subowner can further

divide his portion of the database into other objects and designate others as subowners of the portions.

(2) Sub-ownership can only be removed by the owner who originally authorized the sub-ownership.

(3) Ownership and sub-ownership can be established, replaced and removed. However co-ownership is not allowed.

(4) Owners or subowners can perform all the database management operations on their private portions of the database.

In this kind of hierarchy, at every level of data objects (e.g. records, subfiles, files, databases), there is some single user who is directly responsible for the control of the object.

The two-level transfer-ownership hierarchy enables the creator of a database to become the owner of the database, as already mentioned. The particularity of this structure is that an owner can transfer the ownership of his database to another user. Once he transfers ownership, he no longer can access the database; he is not even a user of the database. The advantage of this structure is that there are not subowners, no co-owners, thereby avoiding the situation where two co-owners attempt to cancel or interfere with each other in terms of authorization.

(b) Centralization vs. Decentralization

A fundamental administrative policy is to choose between centralized or decentralized security controls. With centralized control, a single authorizer (or group) controls all security aspects of the system. The centralized approach has the psychological drawback that the central administrator has too much "power", and that the "owner" of the data can not control it. This may be the approach taken in a small environment, but in a large or complex database it could be more realistic and efficient to have decentralized control. Decentralized control is needed to place control of security at the level that is most meaningful.

We have the situation where systems are becoming much more dynamic. A large system typically has several groups or users. Each group wants to share a central pool of data. But also it wants to easily create and maintain private data. The decentralization of authorization is independent of whether the database itself is centralized or distributed.

A model for decentralization of security functions developed by Wood and Fernandez [WOOD79] is presented. This model takes an approach similar to the multi-level sub-ownership structure described in the preceding section.

The model distinguishes between two types of rights: (1) administrative rights, are privileges to "control" other user's access to data; (2) access rights, are the ones that allow the user to access and manipulate the data in the files. The objects of delegation of authorization functions are called "classes". A data class is a set of data object occurrences. These classes could be partitioned into "subclasses".

In the model the structure of classes is described by a "class structure graph" where nodes represent classes and a directed arc from node 'i' to node 'j' indicates that class 'j' is a member of class 'i'. A user is given ADMINISTER right to a class. An administrator of class D may define subclasses of D and delegate his administration.

The various rights associated with the task of administration may be delegated separately or in groups. These include:

a1 - Right to create, delete and modify objects in D
a2 - Right to define and delete D
a3 - Right to authorize READ access to objects in D
a4 - Right to authorize DELETE access to objects in D
a5 - Right to authorize UPDATE access to objects in D
a6 - Right to authorize INSERT access to objects in D
a7 - Right to recall a delegated right for D

Rights a2 thru a6 are called CONTROL (C) rights, and a1 thru a6 ADMINISTER (A) rights. In the example the notation (s, D, t, f) is used, where 's' is the administrator (or user), 'D' is a class (or subclass), 't' the right type and 'f' indicates if the administrator has the right to further delegate (true or false).

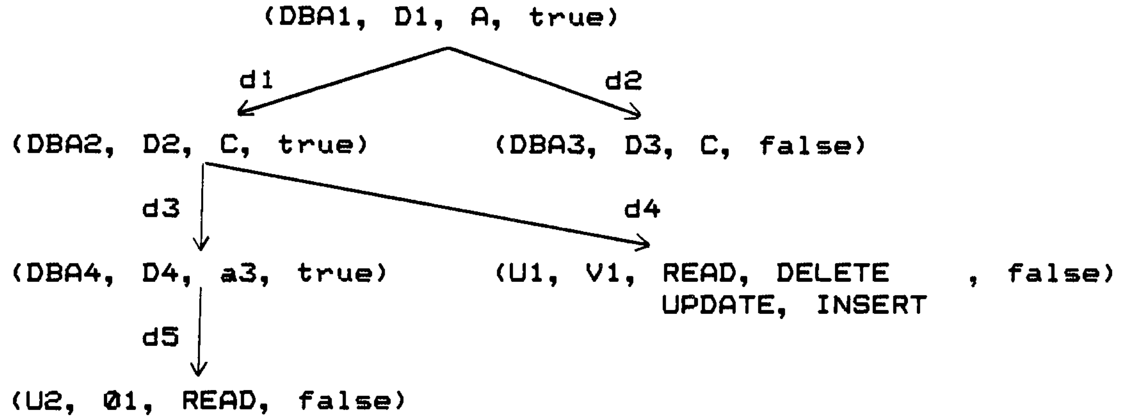


Figure 1

Figure 1 shows a sequence of delegations d1 to d5. Administrator DBA1, with ADMINISTER access to class D1 and with right to delegate, delegates to DBA2 the right to CONTROL class D2 (with right to delegate), and to DBA3 the right to CONTROL D3 (with no right to delegate). DBA2 delegates administration right a3 to DBA4 and gives user U1 a set of access rights on view V1 (an object in D2). DBA4 grants U2 read access to object Ø1 (of class D4).

As administration and database access are separate functions, a reorganization of the administration function should not mean that some users of the system can no longer access the database. Only administrative rights are revoked when a delegated class is recalled. Access rights authorized by the DBA whose administrative rights were revoked are not deleted but become the responsibility of the recalling DBA. The new structure of the graph when CONTROL right is revoked on class D from DBA2 is shown in figure 2. The situation now is logically equivalent to DBA1 having authorized all the access rules. As can be seen user U1 and U2 are still authorized to access the database.

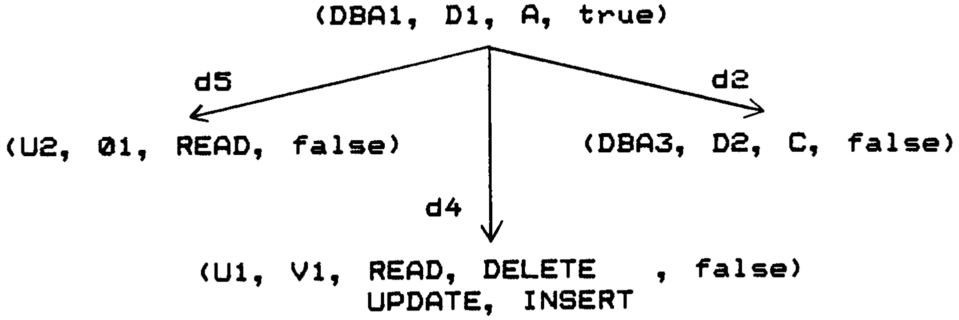


Figure 2

In this model the delegation policy allows an administrator to delegate the rights for a class to only one administrator. If it is necessary to have multiple administrators for some set of objects, then overlapping classes must be defined and separately delegated. This avoids the situation where an administrator receives administrative rights to a class from two different delegators; thus revocation is simplified.

The concept of centralization and decentralization has been referred in the literature also as discretionary and non-discretionary controls. Discretionary implies that the individual user may, at his own discretion, determine who is authorized to access the objects he creates. If the need arises to impose limits on the use of discretionary controls, then these limits are viewed as non-discretionary. A combination of both types of controls can be applied.

### 2.3.3. Access-control Specification Policies

The granularity of access control desired will determine the security measurements needed to be implemented. It is crucial for the successful implementation of a secure data management system that no user is given more information about the structure or content of a database than he is entitled to view. The existence of unviewable fields as well as their content should be hidden from users. It is much harder for a penetrator to obtain access to secret data if he does not know of its existence or of the name under which it is stored.

(a) Name-dependent access control

This is the minimum control requirement when specifying the data objects a user can access. Different levels of granularity could be specified, the lowest or finest one being field or item name (column, in a relational database). It is sometimes referred as content-independent access control because a decision on whether or not to allow a data access can be made without using data values. Security contraints that are independent of a particular data value can have their access decision made once per job execution.

(b) Content-dependent access control

Access rules could be extended by specifying control depending on the value or content of a field. For example restricting a particular user from seeing a field named SALARY in every record of a file is independent of the specific value in that field, while restricting a user from seeing values of SALARY in excess of $10,000 is a content-dependent control.

Content-dependent control must be interpreted in a general sense. The decision may depend upon the value of any datum in the system and not just the particular datum to which access is desired. For example, if a user is permitted access to salary data only between the hours of 9 and 5, then the current time is the datum in which the security decision depends. For this type of control, data values must first be retrieved from the database in order to determine whether or not the access request should be satisfied. This implies that the enforcement will be performed at execution time and that the evaluation must be repeated for each potential data element in the same class.

(c) Context-dependent access control

The policy of context-dependent control refers to using a combination of items. One aspect of this policy restricts the fields that can be accessed together. For example, in a relation containing employee names and salaries, one may want to prevent some users from finding the salaries of particular employees. One alternative would be to prevent any access at all to those relations. But to maximize sharing, one could allow separate access to names and salaries while preventing users from accessing them together. Another aspect of the policy is the requirement that certain fields appear together.

(d) History-dependent access control

It may be necessary to control not only the context in the immediate request, but also the content of previous requests, if one wants to prevent users from making certain deductions. For example, let's suppose there is an employee relation which contains a project-identifier attribute; a user could list first all names and projects and then all salaries and projects. He can then make some correlation between names and salaries. Preventing this kind of deduction requires history-dependent control, which takes into account not only the context of the immediate request but also all past requests. The current access of the user is restricted because of accesses he made in the past.

## 3. Models

Several models to express access control needs have been developed. These models intend to aid the security administrator or manager in charge of the system's security to express (or model) the relationship among subjects, objects, and access privileges the first ones have over the latter. The models to be discussed use different techniques to describe these relationships. Some use a matrix (row, columns), or a variation of it; others use directed graphs and others explicitly indicate the rules using predicates.

Some of the models have gone beyond the phase of modeling and have been implemented in more or less the same way they are used to model the access control rules.

These will be covered latter. Now an overall definition of models used most often is presented.

## 3.1. Access Matrix

The access-matrix model provides a framework for describing protection systems. The model was independently developed by researchers in both the operating system area and the database area. The model is defined in terms of state and state transitions, where the state of a protection system is represented by a matrix, and the state transitions are described as commands.

The protection system comprises three parts, which will be reflected as components of the model. The first component is a set of 'objects', O, an object being an entity to which access must be controlled. Examples of objects are files, relations, and fields. The second component is a set of 'subjects', S, a subject being an active entity whose access to objects must be controlled. The model assumes that subjects are considered to be objects, thus, $S \subseteq O$. The third component of the protection system is the set of rules which govern the accessing of objects by subjects.

All information specifying the type of access subjects have to objects is regarded as constituting a 'protection state' of the system. This protection state is represented as an 'access matrix', A, with subjects identifying the rows and objects the columns. The entry A[S,X] contains strings, called 'access attributes' specifying the access privileges held by subject S to object X. If string 'a' appears in A[S, X], then "S has 'a' access to X". For example, in Figure 3, subject $S_1$, may read $F_1$ , since 'read' appears in A[$S_1$ , $F_1$ ]; or, $S_2$ may update $F_2$ .

| | $S_1$ | $S_2$ | $S_3$ | $F_1$ | $F_2$ | $T_1$ |
|---|---|---|---|---|---|---|
| $S_1$ | control | owner | owner | read<br>write | | execute |
| $S_2$ | | control | | owner | update | owner |
| $S_3$ | | | control | delete | owner | |

Figure 3
Access Matrix

Graham and Denning [GRAH72] associate with each type of object a "monitor" through which all accesses to objects of that type must pass to be validated. Examples of monitors are the file system for files, the hardware for instructions execution and the protection system for subjects (see Figure 4). An access proceeds as follow:

(1) S initiates access to X in manner 'a'.

(2) The computer system supplies the triple (S, a, X) to the monitor of X.

(3) The monitor of X interrogates the access matrix to determine if 'a' is in A[S, X]; if it is, access is permitted, otherwise, it is denied and a protection violation occurs.

The access attributes are interpreted by object monitors at the times accesses are attempted. Figure 4 shows the organization of the protection system. The mechanisms between the dashed lines of the diagram are invisible to subjects - subjects direct their references to objects - these references are then intercepted and validated by the monitors of the system.

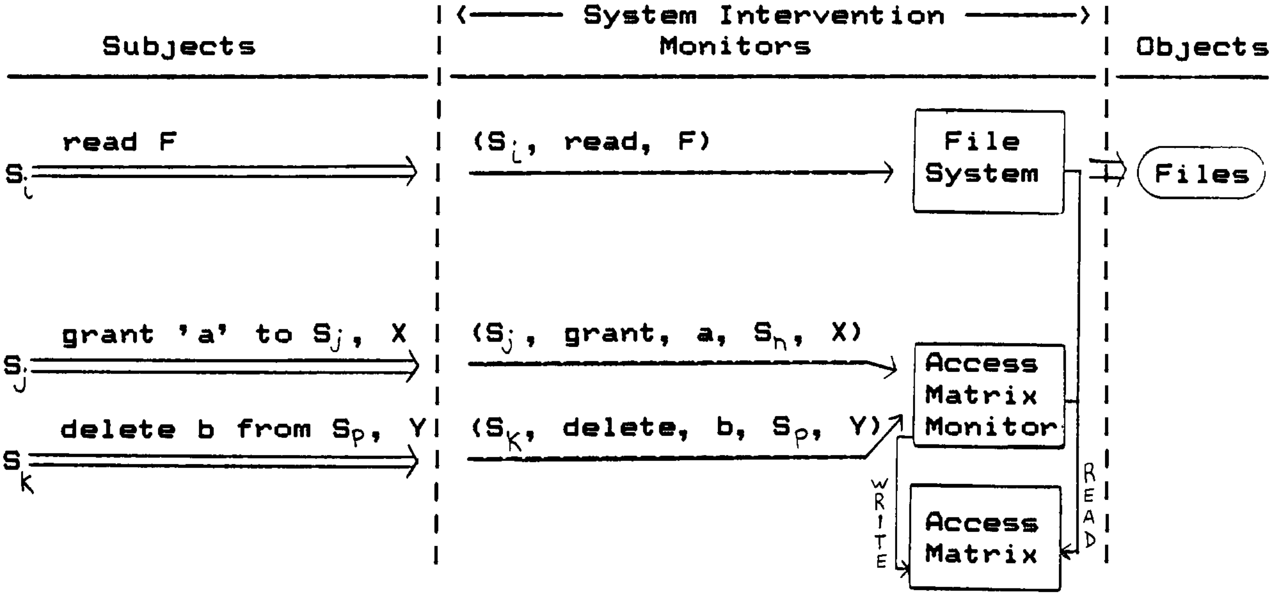


Figure 4
Organization of protection system

The foregoing rules govern the use, by monitors, of the access matrix, once it has been specified. There is also a series of rules for changing the access matrix itself. These rules will be enforced by the monitor of the access matrix. Unlike the monitors of the system's objects, the access matrix monitor may modify the access matrix. In particular, it may transfer, grant, or delete access attributes on command of subjects and only on appropiate authorization. For this purpose attributes 'owner' and 'control' are introduced, as well as the notion of a 'copy flag' (denoted by an asterisk), and the rules R1 - R3 of Table I, to be implemented by the access matrix monitor.

Rule 1 permits a subject to transfer any access attribute it holds for an object to any other subject, provided the copy flag of the attribute is set, and it may specify whether the copy flag of the transferred attribute is to be set; in Figure 3, for example, $S_1$ may place 'read*' in $A[S_2, F_1]$, but it may not transfer its ability to execute $T_1$ to any other subject. Rule 2 permits a subject to grant to any other subject access attributes to an object it owns; in Figure 3, for example, $S_1$ can grant any type of access for $S_2$ to any subject. Rule 3 permits a subject to delete any access attribute (or copy flag) from the column of an object it owns, or the row of a subject it controls; in Figure 3, for example, $S_1$ can delete any entry from column $S_2$ or $S_3$. In order to facilitate this, it is required that 'control' be in A[S, S] for every subject S and rule R4 is included, which permits a subject to read that portion of the access matrix which it owns or controls.

It should be noted that a subject may hold 'owner' access to any object, but 'control' access only to subjects. It is assumed that each subject is owned or controlled by at most one other subject.

If one wants to limit the number of outstanding access attributes to a given object, e.g. if it is required that each subject be owned by at most one other subject, or that a given object be accessible to a limited number of subjects, a 'transfer-only' copy flag could be introduced. Let's denote this flag by the symbol #. Then in R1 the command to transfer 'a' (or a#) from $S_0$ to S for object X would be authorized if a# were in A[$S_0$, X] and would cause a# to be deleted from A[$S_0$, X] and 'a' (or a#) be placed in A[S, X].

The creation of a non-subject object, e.g. a file, is straightforward, consisting of adding a new column to the access matrix. The creator subject executes a command (rule 5 of Table I) and is given 'owner' access to the newly created object. It then may grant access attributes to other subjects for the object according to rule R2. The destruction of an object, permitted only to its owner, corresponds to deleting the column from the access matrix (R6).

Creating a subject consists of creating a row and a column for the new subject in the access matrix, giving the creator 'owner' access to the new subject, and giving the new subject 'control' access to itself (rule R7). The destruction of a subject, permitted only to its owner, corresponds to deleting both the row and the column from the access matrix (rule R8).

In the access matrix model described so far, the entry 'a' in the intersection of a given subject row and object column implies that this type of access is granted automatically to the corresponding request. Fernandez, Summers and Coleman [FERN75] extend this concept in order to include arbitrary predicates which participate in the decision. In other words, the entry in the intersection of subject S and object X may contain a predicate P, that may depend on any data in the system, that must be satisfied in order for access of type 'a' to be granted. Access can then be described as

$$S \xrightarrow{P\ :\ a} X$$

Table I

| Rule Command (by $S_0$) | Authorization | Operation |
|---|---|---|
| R1 transfer $\left\{\begin{matrix}a*\\a\end{matrix}\right\}$ to S, X | a* in A[$S_0$ , X] | store $\left\{\begin{matrix}a*\\a\end{matrix}\right\}$ in A[S, X] |
| R2 grant $\left\{\begin{matrix}a*\\a\end{matrix}\right\}$ to S, X | 'owner' in A[S , X] | store $\left\{\begin{matrix}a*\\a\end{matrix}\right\}$ in A[S, X] |
| R3 delete a from S, X | 'control' in A[$S_0$, S]<br>or<br>'owner' in A[S , X] | delete a from A[S, X] |
| R4 w = read S, X | 'control' in A[$S_0$ , S]<br>or<br>'owner' in A[$S_0$ , X] | copy A[S,X] into w |
| R5 create object X | none | add column for X to A;<br>store 'owner' in<br>A[S, X] |
| R6 destroy object X | 'owner' in A[S , X] | delete column for X<br>from A |
| R7 create subject S | none | add row for S to A;<br>execute 'create object S'<br>store 'control' in A[S, S] |
| R8 destroy subject S | 'owner' in A[$S_0$, S] | delete row for S from A;<br>execute 'destroy object S' |

In actual computer systems, the access matrix would be very sparse if it were implemented as a two dimensional array. In most real world data processing situations the number of users (subjects) would be substantial, and the number of objects would be extremely large. The authorization matrix can usually be compressed to a reasonable size by some or all of the following methods:

(1) defining groups of 'virtual' users, each representing a group of users with identical security authorization.

(2) grouping the data elements (objects) into a number of data security categories.

(3) storing a list of pairs (user, permission) for each data element.

(4) storing a list of pairs ( data element, permission) for each user.

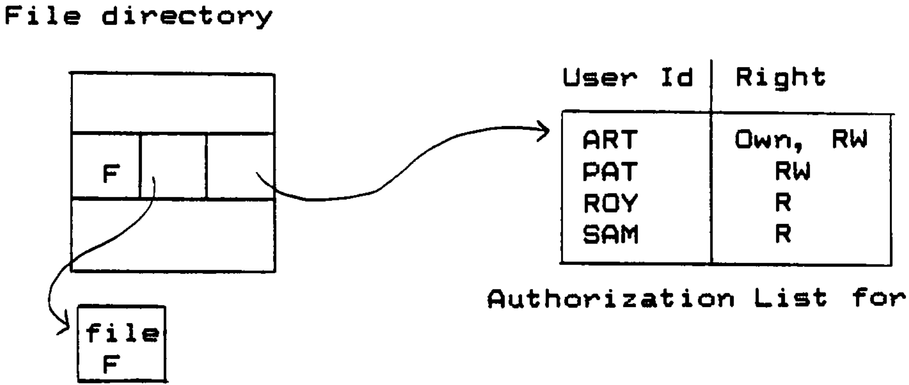
The last two are to be covered next.

### 3.1.1. Authorization List

An authorization list (also called access-control list) is a list of n subjects who are authorized to access some particular object X. The i-th entry in the list gives the name of a subject $S_i$ and the right $r_i$ in $A[S_i, X]$ of the access matrix. An authorization list, therefore represents the nonempty entries in column X of the access matrix.

Authorization lists are typically used to protect owned objects such as files. Each file has an authorization list specifying the names (or IDs) of users or user groups and the access rights permitted to each (see Figure

5). The owner of a file has the sole authority to grant access rights to the file to other users; no other user with access to the file can confer these rights on another user (the copy flag is off). The owner can revoke (or decrease) the access rights of any user simply by deleting (or modifying) the user's entry in the authorization list.



| User Id | Right |
|---|---|
| ART | Own, RW |
| PAT | RW |
| ROY | R |
| SAM | R |

Authorization List for F

Figure 5
Authorization list for a file

### 3.1.2. Capability List

The storage of access information by row is called a 'capability list' or C-list. A capability list is a pair (X, r) specifying the unique name of an object X and a set of access rights r for X (some capabilities also specify an object's type). The capability is a ticket in that possession unconditionally authorizes the holder r-access to X. Once the capability is granted, no further validation of access is required.

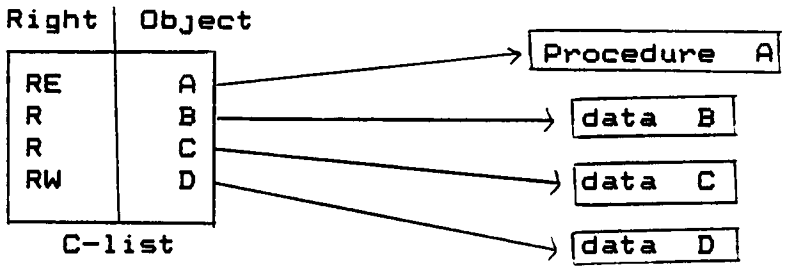


Figure 6
Capability List

The C-list for a subject S is a list of n capabilities for the objects S is permitted to access, where r gives the rights in A[S, $X_i$] of the access matrix. The C-list for S, therefore represents the nonempty entries in row S of the access matrix. Figure 6 illustrates a C-list that provides read/execute access (RE) to the procedure A, read-only-access (R) to data objects B and C, and read-write access (RW) to data object D.

## 3.2. Directed Graph Model

### 3.2.1. Take-Grant

Jones [JONE78] discussed the take-grant graph model to describe a restricted class of protection system. As in the access matrix model, a protection system is described in terms of states and state transitions. A protection state is described by a directed graph G, where the nodes of the graph represent the active subjects and passive objects of the system. Subjects are represented by closed circles as ●, and objects by open circles as ○. Objects not known to be either active or passive are notated with slashed circles, ⊘. Graphically, a right is notated by a directed edge, labeled with a name. It is interpreted to mean that the node at the tail of the edge has the named right to the node at the head of the edge. The protection state of a collection of subjects and objects is represented as a finite graph; the graph is called a 'protection graph'.

The graph in Figure 7 models a protection state in which subject A has the right to perform operation 'a' on subject B, which in turn has the right to perform 'b' on passive objects X and Y. In addition, B has 'f' right to Y.

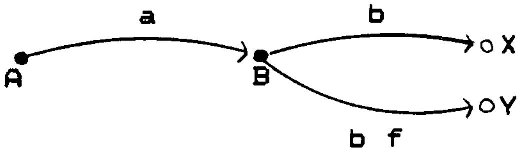


Figure 7
Protection state

Arcs can be labeled with multiple labels. If a directed arc is to be added between two objects for which an edge with the same direction already exists, a single edge labeled with the union of the existing label(s) and the new label(s) is used.

The take-grant model is intended to model the access control mechanism of a system. In a system, the protection state changes only when some subject invokes an operation that is defined as part of the protection mechanism. In the take-grant system, these operations are modeled by a set of rewriting rules for protection graph transitions. Because the protection state changes only by action of a subject, the model will refer to a subject 'exercising' its rights or privileges. Consequently, in the model any graph rewriting rules will always require at least the precence of a subject; usually a subject must have a particular right to some object as a prerequisite for a graph transition.

There are two special rights 'take', denoted by the label 't', and 'grant', denoted by the label 'g'. For the definition of the four rewrite rules, let A, X, and Y be three distinct vertices in a protection graph, such that A is a subject.

Take: Let there be an edge from A to X with at least a label 't', and an edge from X to Y with any label or set of labels 'a'. Then applying the Take rule, add an edge from A to Y having label 'a': "A takes the right to perform 'a' to Y from X" (figure 8)

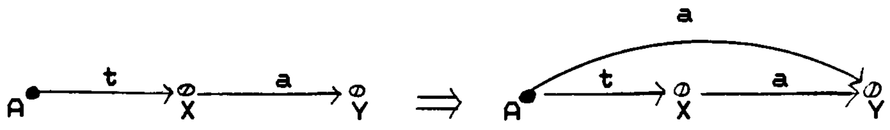


Figure 8 - Take right

Grant: Let there be an edge from A to X with at least a label 'g', and an edge from A to Y labeled 'a'. Then applying the Grant rule, add an edge from X to Y having label 'a': "A grants the right to perform 'a' over Y to X" (figure 9)

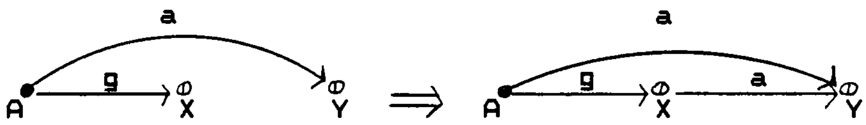


Figure 9 - Grant right

Create: Let A be a subject and 'a' a subset of rights. Applying the Create rule, add a new vertex N such that 'a' labels the edge from A to N: "A creates the subject or object N with 'a' right" (figure 10)

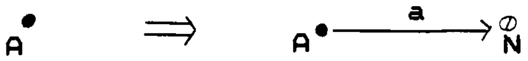


Figure 10 - Create right

Remove: Let there be an edge labeled 'f' from subject A to X. Let 'a' be any subset of rights. Applying the Remove rule causes deletion of the 'a' labels from 'f'. If f = a, then the edge itself is deleted: "A removes its right to 'a' X" (figure 11)

Figure 11 - Remove right

### 3.2.2. Grant-Revoke

Griffiths and Wade [GRIF76] used the directed graph model to define a dynamic authorization mechanism. A database user can grant or revoke privileges (such as to read, insert, or delete) on a file he has created. Furthermore, he can authorize others to grant these same privileges. The database system keeps track of a directed graph of granted privileges.

The nodes of the graph correspond to users, and the edges correspond to grants. The edges are of two types, corresponding to whether or not the recipient of the grant has been given the option to make further grants of this privilege. For each pair of A, B of nodes, there can be no more than one edge of each type from A to B.

The representation of grants is straightforward, an edge is drawn from node A to node B if node A is granting P privileges to node B. The problem arises when some of the privileges have to be removed. The decision about exactly which privileges are to be revoked is not obvious. One might expect that if the revokee possesses other grants of the revoked right, then recursive revocation should not take place. The problem is that such an algorithm does not detect cycles in the chain of grants following the revokee. The revocation algorithm must distinguish between the two cases shown in Figure 12. Effectively the correct algorithm traces the grant chain from X back to the creator of the object. If every such path passes through the revoker, then X's privileges should be revoked. However, if there exists a path back to the creator which does not pass through the revoker, then X should retain the privileges after the revoke.

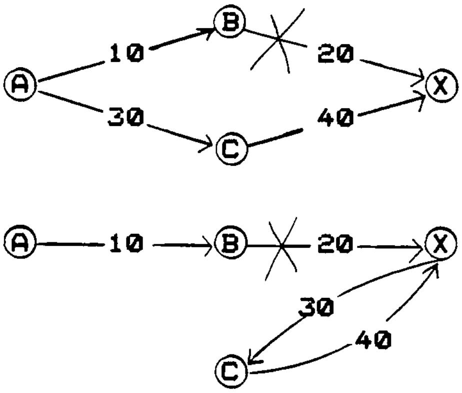


Figure 12
Effect of revocation of privileges

To decide whether to revoke recursively or not, each edge is labeled with a timestamp. The timestamp may represent real time or it may be a system maintained counter. No two grant commands could be tagged with the same timestamp. If the same privilege is granted by the same grantor to the same user on the same object, then the earlier timestamp is maintained. The duplicate later grant is not recorded in the graph. This condition was proved by Fagin [FAGI78] to be a flaw in the design since it could forbid some user from exercising or granting a privilege he "should" be allowed to exercise or grant.

Let's assume that user A is the creator of file f, and that a number of grants, each with grant option (copy flag on), of privileges P over file F take place as in Figure 13(a). For example user A grants user B privilege P at time 10, user B grants user C privilege P at time 20, and so on. Note that C grants D privilege P both at time 30 and at time 60. Under the original model the second grant from C to D is ignored, and so, after time 60 only the grants in Figure 13(b) are included in the graph.

Assume now that at time 70 user B revokes privilege P from user C. On the original model the recursive revocation will be as follows: On step 1 the grant from B to C is deleted. Then, the earliest remaining grant to C has timestamp 40. So on step 2, the revocation algorithm deletes the grant from C to D, since the timestamp (30) of this later grant is smaller that 40. Similarly, on the last step, the grant from D to E is deleted. Thus, under the original mechanism the final authorization graph will look like Figure 13(c). In particular the system no longer authorizes user D to exercise privilege P. However, user D should be allowed privilege P, since there was a grant from A to C at time 40, and from C to D at time 60.

Under the modified mechanism, on step 1 of the revocation mechanism, after the revocation by B at time 70, the grant from B to C is deleted. Then the earliest remaining grant to C has timestamp 40. So on step 2 the grant from C to D with timestamp 30 is deleted (but the grant from C to D with timestamp 60 is not deleted). On the last step, the grant from D to E is deleted. The final

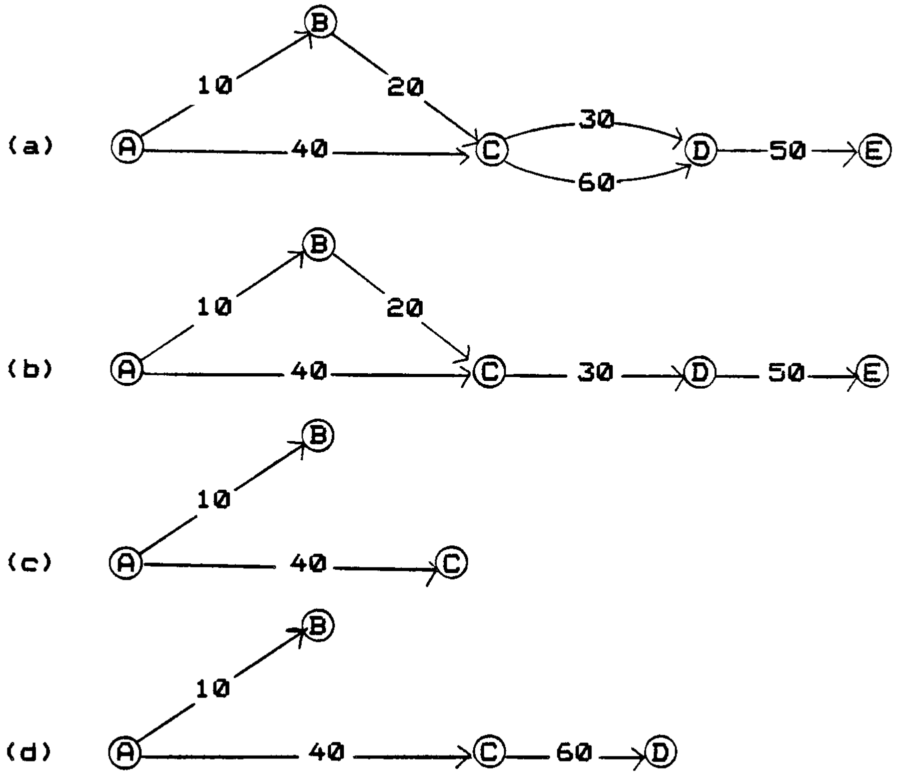


Figure 13
Recursive revocation of privileges

resulting graph contains the grants in Figure 13(d). So after time 70, user D is authorized to exercise (and to grant) privilege P, as he should.

## 3.3. Query Modification

The 'Query Modification' is a mechanism which is applicable at a high-level, user interface. The basic notion pursued is one of interaction modification. Users interact with a database through a high-level query language. Any such interaction is immediately modified into an interaction which is compiled into a sequence of simpler interactions, which are then executed without further concern for access control.

Associated with each user is a list with entries of the form (R, P), where R is the name of a relation (table), and P is a set of logical privileges or access restrictions on R. The list is similar to a capability list in that it defines a user's access rights to the database. Each access restriction is of the form (f, S), where S is an expression identifying a subset of R. It authorizes the user to perform operation f on the subset defined by S. If the user poses a query (f, R, E) the system modifies expression E according to the expression S. That is, assume a list of restriction predicates p1, p2,...pn. Then suppose the user issues a query Q. The system changes the query into "Q & (p1 | p2 | ... | pn)" such that only an authorized subset of what the user would have retrieve is actually delivered to him. See Figure 14.

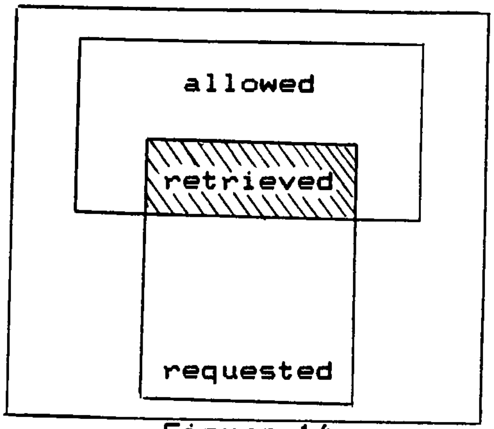


Figure 14
Query modification access control scheme

For example consider the relation EMPLOYEE, with attributes NAME, SALARY, MGR (manager), DEPT. A query to retrieve the name and salary of all employees whose manager is Jones would be:

```
Q1:   RETRIEVE EMPLOYEE (NAME, SALARY) WHERE MGR = 'JONES'
```

Query Q1 retrieves all tuples of EMPLOYEE that satisfy the qualification (predicate). Suppose there is a rule that permits the user to retrieve only the records of employees in department D1. Then the preceding request would be modified to :

```
RETRIEVE EMPLOYEE (NAME, SALARY) WHERE MGR = 'JONES'
                                 AND DEPT = 'D1'
```

## 3.4. Lattice Model

The lattice model was motivated by the controls used by the Department of Defense (DOD) and other national security agencies to regulate people's access to sensitive information. The DOD information security policy gives each document a classification level, L, and a (possibly empty) set of categories, C. The security levels are strictly ordered. The categories tend to have no ordering or precedence, but subsets of categories are ordered by set inclusion. The combination of a classification level and a set of categories is referred to as an 'access class'. Figure 15 identifies the DOD classification levels and their order, and some examples of categories.

Top Secret > Secret > Confidential > Unclassified

(a) Classification levels

Nuclear, Intelligence

(b) Example categories

Figure 15
DOD levels and categories

The classification levels and categories in the DOD system are also used to label a set of clearances that can be granted to users. There are two fundamental access rules in the lattice model:

(1) the 'simple security property' - no subject has read access to any object that has a classification greater that the clearance of the subject.

(2) the '*-property' (pronounced 'star-property') - no subject has append-access to an object whose security level is not at least the current security level of the subject; no subject has read-write access to an object whose security level is not equal to the current security level of the subject; and no subject has read access to an object whose security level is not at most the current security level of the subject.

A subject S may read an object O if

$$L_S \geq L_O \ \& \ \{C\}_S \supseteq \{C\}_O$$

(a) Simple Security Property

A subject S may write an object O if

$$L_S \leq L_O \ \& \ \{C\}_S \supseteq \{C\}_O$$

(b) *-property

Figure 16
Lattice Model Security Rules

Any attempt to apply the security lattice model to non-DOD environments will try to identify levels and categories that reflect the levels of sensitivity and organizational division in the subject environment. For example, Figure 17 shows some possible levels and categories identified in a corporation. The 'system-low' level (SL) corresponds to unclassified in the DOD system, and information at this level is readable by all users.

Security Levels: Audit-Manager (AM)
System-Low (SL)

Categories: Production-Data (PD)
Production-Code (PC)
Development (D)
System-Development (SD)
Tools (T)

Figure 17
Levels and categories

The application of the access classes to the system's subjects is outlined in Figure 18. Users, application developers and system programmers each have system-low security level and two categories. In each case the first category '[]' is that of the information they manipulate (read-write), and the second '()' is that of the programs the subjects can use. The audit and management subjects have access to information of any category, and security level of audit-manager. Finally a system control function is defined with system-low and access to each category, and 'downgrade' privilege to change categories.

| Subjects | Access Class |
|---|---|
| System management or Audit | AM; any set of categories |
| User | SL; [PD] <PC> |
| Application developers | SL; [D] <T> |
| System programmer | SL; [SD] <T> |
| System control | SL; [PC, PD, D, SD, T] plus 'downgrade' privilege |

Figure 18
Users' Access Classes

The assingment of access classes to files (objects) is given in Figure 19. The program objects (production code, tools) each has a single category and are intended to be read-only (unmodified). Objects that could be manipulated (production data, system and application programs under development) have two categories each - that of the object itself '[]' and that of the program that operates on it '<>' - so that a subject executing the program will be allowed by the *-property to write the object. Audit trial information is developed with the category(ies) of the activity being audited and is 'written up' to the higher audit-manager security level.

| Files | Access Class |
|---|---|
| Production data | SL; [PD] <PC> |
| Production code | SL; [PC] |
| Developing code/test data | SL; [D] <T> |
| Software tools | SL; [T] |
| Systems programs in modification | SL; [SD] <T> |
| System programs | SL |
| System and application Audit Trial | AM; <appropriate categories> |

Figure 19
File Access Class Assingment

The overall effect of the configuration of access classes described above is shown in Figure 20.

| Objects<br>Subjects | Prod. Data | Prod. Code | Dev. App. Prg. | Dev. Sys. Prg. | Tools | Sys Prg. | Audit Trail |
|---|---|---|---|---|---|---|---|
| System Mgt. and Audit | R | R | R | R | R | R | RW |
| Production users | RW | R | | | | R | W |
| Application Programmer | | | RW | | R | R | W |
| System Programs | | | | RW | R | R | W |
| System Control | RW | RW | RW | RW | RW | RW | W |

Figure 20
Effect of Lattice Model specifications

## 4. Systems

## 4.1. INGRES

INGRES (Interactive Graphics and Retrieval System) is a relational database system which is implemented on top of the UNIX operating system. INGRES introduced a way to implement fine grain access control using a technique called 'Query Modification'. A brief description of the INGRES system and its operational environment is first presented to provide a background for the discussion of the protection scheme.

The user interface for INGRES consists of the data sublanguage QUEL (QUEry Language). QUEL is a complete query language in that it frees the programmer from concern for how data structures are implemented and what algorithms are operating on stored data. It is a nonprocedural language which has points in common with some relational calculus based languages such as Data Language/ALPHA. It does not use calculus quantifiers and so, is considered by its designers to be a language based on functions and not on a first order predicate calculus [STON76b]. The QUEL examples in this section concern the following relations

```
EMPLOYEE (NAME, DEPT, SALARY, MANAGER, AGE)
DEPT     (DEPT, FLOOR#)
```

A QUEL interaction includes at least one RANGE statement of the form

RANGE OF variable-list IS relation-name

The purpose of this statement is to specify the relation over which each variable ranges. The interaction also includes one or more statements of the form:

Command (target-list) [WHERE qualification]

Here, 'Command' is either RETRIEVE, APPEND, REPLACE, or DELETE. For example, a query to retrieve the names and salaries of all employees whose manager is JONES would be:

```
RANGE OF E IS EMPLOYEE
RETRIEVE (E.NAME, E.SALARY)
WHERE E.MGR = 'JONES'
```

In addition to the above QUEL commands INGRES also supports a variety of utility commands. The only ones of interest in a protection context are COPY, PRINT and CREATE. The COPY command transfers a relation to or from a UNIX file. It is a 'bulk transfer' mechanism and therefore must be protected. PRINT is a simple report generator that writes a relation onto a user's terminal. CREATE sets up an empty relation and prepares it for use.

INGRES manages a collection of databases, each of which is made up of a set of relations. Each database is associated with a special user called the Data Base Administrator (DBA). A list of allowable DBAs is kept in INGRES.

Only a database's DBA may create shared relations in that database. Relations created by other users are guaranteed private by INGRES. The invoker of the CREATE command becomes the 'owner' of the relation created. A user may only destroy a relation he owns.

INGRES makes available to the DBA the following command to specify access permission for shared relations:

```
PERMIT relname TO object FOR command
          (targetlist; idlist) WHERE qualification
```

The fields of this command have the following meaning:

relname: specifies the name of the relation to be protected

object: specifies the object to be controlled. This may be a user, a teletype, or the keyword ALL, which will cause the restriction to apply to all users.

command: specifies which type of access is to be allowed. The types of access allowed are

RETRIEVE, APPEND, DESTROY, REPLACE, DELETE, COPY and PRINT, or the keyword ALL.

targetlist: a list of domains of the form tuple-variable.domain which may be accessed.

idlist: a list of domains of the form tuple-variable.domain which can be accessed but not updated. These domains may appear in a qualification clause of an update, but may not appear in the target list. This field is optional. However, only one tuple variable can be presented in the targetlist and idlist.

qualification: any valid QUEL qualification containing any number of tuple-variables. It specifies the subset of the relation which may be accessed by an interaction of the type specified by 'command'. A pound symbol (#) may appear in the qualification to specify the UNIX logon name of the user currently invoking INGRES.

The database administrator can enter an arbitrary number of protection statements governing access to his relations. Each protection statement is given a unique ID and stored in the PROTECTION relation. This relation has domains of relation name, command, object, target-list, idlist, qualification, protectionid. This relation is normally compressed and hashed on the first two domains. Removal of protection statements is accomplished by the DBA using the utility command

DENY (protection-id).

Relations owned by other users are guaranteed private by INGRES. Therefore only the DBA is permitted the use of the PERMIT command. Since system catalogs contain data about the database, their integrity must be carefully guarded. Consequently no user (including the DBA) is allowed to update system catalogs using QUEL. However, RETRIEVE permission to a portion of a catalog may be granted to others by the DBA.

In summary the DBA has the following powers not available to ordinary users:

(1) the ability to create shared relations and to specify access control for them.

(2) the ability to destroy any relation in his database (except the system catalogs).

This system allows 'one-level' sharing in that only the DBA has the above powers and he cannot delegate then to others. The designers' arguments to support this centralized control were:

(1) Additional generality would have created considerable problems, such as making revocation of privileges nontrivial.

(2) It seems appropriate to entrust to the DBA the duty (and power) to resolve the policy decision which must be made when space is exhausted and some relations must be destroyed (or archived). This policy decision becomes much harder (or impossible) if a database is not in control of one user.

(3) Someone must be entrusted with the policy decision concerning which relations to physically store and which to define as `views'. This "database design" problem is best resolved by a centralized DBA.

INGRES requests are analyzed and validated at execution time, one statement at a time. The implementation of access control has two major facets. The first is the representation of information in the PROTECTION relation (see figure 21). If data were stored in a text string, the string would have to be completely parsed for each interaction to which it applies. The overhead of this strategy is considered unreasonable.

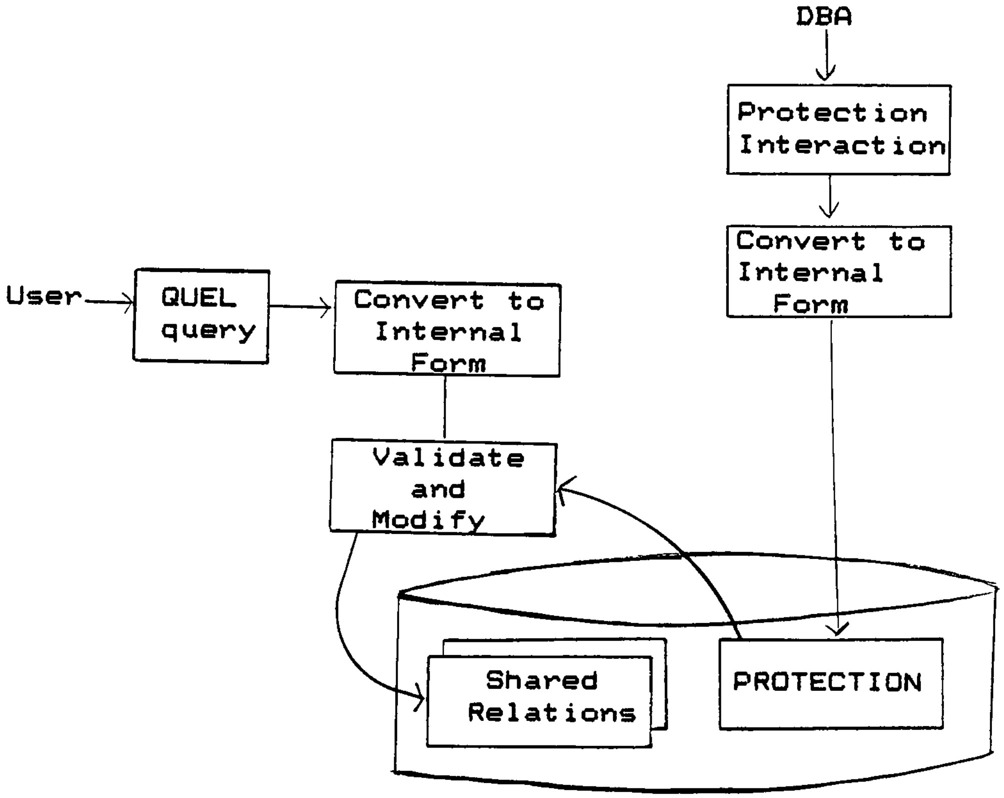


Figure 21
INGRES access control

Consequently a protection statement will be parsed when entered and stored as a tree structure in the PROTECTION relation. When an interaction is to be modified, the qualification trees of the protection interaction need only be attached to the qualification tree of the user's interaction. The ID and target lists are stored as a list of domain numbers which are then internal representation for domain names.

The second facet of the implementation is the enforcing of the protection interaction. This is done by the following algorithm:

(1) Find all attributes in the target list on the unmodified qualification statement of the interaction. Call this set S.

(2) Find all access control interactions whose command type and object match those of the user's interaction and with a target list containing all attributes in S. Denote this by T. If there is not such set T, the query is aborted due to lack of permission.

(3) Ignore any access rules in T whose target list contains the target lists of other rules in T. That is, find the smallest target lists that cover the request. Call the qualification of the remaining rules $P_1, \ldots, P_n$.

(4) Replace the request qualification $Q_r$, by

$$Q_r \text{ AND } (Q_1 \text{ OR } Q_2 \text{ OR } \ldots \text{ OR } Q_n)$$

(5) Execute the modified interaction normally.

What this algorithm means is that all rules relevant to the request are first located. It is then assumed that rules are well nested, that is, Rule A is more restrictive than Rule B if Rule B authorizes access to a subset of attributes in Rule A. Only the most restrictive rules are retrieved for the query modification.

The following example illustrates the algorithm:

```
Ex. Each manager can read salaries of employees
    who work for him

Protection interaction:
    RANGE OF E IS EMPLOYEE
    PERMIT ALL TO EMPLOYEE FOR RETRIEVE
          (E.SALARY; E.NAME)
    WHERE E.MANAGER = #

An interaction by Jones:
    RETRIEVE (E.SALARY)
    WHERE E.NAME = "SMITH"

would by modified to:
    RETRIEVE (E.SALARY)
    WHERE E.NAME = "SMITH"
      AND E.MANAGER = "JONES"
```

In INGRES the objects protected by the access control rules are always real relations. This is a different approach from other systems which provide access control to virtual relations, or views. The following example illustrates the rationale behind this design decision:

```
Consider the following two views:
    RANGE OF E IS EMPLOYEE
    DEFINE RESTRICTION-1 (E.NAME, E.SALARY, E.AGE)
    WHERE E.DEPT = "toy"

    DEFINE RESTRICTION-2 (E.NAME, E.DEPT, E.SALARY)
    WHERE E.AGE < 50

and the following two access control statements:
    RANGE OF E IS EMPLOYEE
```

```
PERMIT ALL TO EMPLOYEE FOR RETRIEVE
  (E.NAME, E.SALARY, E.AGE)
WHERE E.DEPT = "toy"

PERMIT ALL TO EMPLOYEE FOR RETRIEVE
  (E.NAME, E.SALARY, E.DEPT)
WHERE E.AGE < 50
```

Access control could be based on views, where a given user may be authorized to use views RESTRICTION-1 and RESTRICTION-2. To find the salary of Smith he might interrogate RESTRICTION-1 as follow:

```
RANGE OF R IS RESTRICTION-1
RETRIEVE (R.SALARY)
WHERE R.NAME = "Smith"
```

Failing to find Smith in RESTRICTION-1, he would have then to interrogate RESTRICTION-2. After two queries he would be returned the appropriate salary if Smith was under 50 or in the toy department. Under the INGRES scheme the user can issue:

```
RANGE OF E IS EMPLOYEE
RETRIEVE (E.SALARY)
WHERE E.NAME = "Smith"
```

which will be modified by the access control algorithm to

```
RANGE OF E IS EMPLOYEE
RETRIEVE (E.SALARY)
WHERE E.NAME = "Smith"
      AND
(E.AGE < 50 or E.DEPT = "toy")
```

In this system the user need not manually sequence through his views to obtain such data if permitted. Note clearly that the portion of EMPLOYEE to which the user has access (the union of RESTRICTION-1 and RESTRICTION-2) is not a relation and hence cannot be defined as a single view.

To summarize, access control restrictions are handled automatically by the INGRES algorithm. In a view oriented scheme, a user must sequence through his views to obtain allowed information.

## 4.2. DB2

DB2 (Data Base 2) is a subsystem of IBM's MVS operating system. It is a relational database management system (DBMS) for that operating system. Any given DB2 application, that is, any application program that accesses one or more DB2 databases, will execute under the control of exactly one of the three subsystems IMS, CICS, or TSO.

The user interface to DB2 is the SQL language (Structured Query Language). SQL is a relational nonprocedural data sublanguage with ideas both from the relational calculus and the relational algebra. However, it does not make use of calculus quantifiers or other mathematical concepts, rather it is a structured query language with English keywords. SQL can be used in both interactive and embedded environments, and it provides both data definition and data manipulation functions. The major data definition functions are:

```
CREATE TABLE          DROP TABLE
CREATE VIEW           DROP VIEW
CREATE INDEX          DROP INDEX
```

The data manipulation functions are:

```
SELECT    UPDATE    DELETE    INSERT
```

For example, a SELECT statement has the form:

```
SELECT  column-list
FROM    relname
WHERE   condition
```

where:

column-list: specifies the columns of the target result which is always a table (null, one row, or many rows). An asterisk (*) could be used to retrieve all columns of the relation.

relname: specifies the table(s) from which the target result is to be obtained.

condition: specifies the logical condition(s) or predicates on which the result is to be obtained.

Consider the following relations:

```
EMPLOYEE (NAME, DEPT, SALARY, MANAGER, AGE)
DEPT     (DEPT, FLOOR#)
```

A query to retrieve the names and salaries of all employees whose manager is Jones would be:

```
SELECT NAME, SALARY
FROM   EMPLOYEE
WHERE  MANAGER = "JONES";
```

There are two more or less independent features in the system that are involved in the provision of security in DB2: (1) the view mechanism, which can be used to hide sensitive data from unauthorized users, and (2) the authorization subsystem, which allows users having specific privileges selectively and dynamically to grant those privileges to other users, and subsequently revoke them if desired.

Once the policy decisions, as to which specific privileges should be granted to which specific users are made, DB2 enforces them in the following way:

(a) The results of those decisions are made known to the system by means of the GRANT and REVOKE statements, and are remembered by the system by saving them in the catalog.

(b) It provides a means of checking a given access request against the applicable authorization constraints (here 'access request' refers to the combination of requested operation plus target object plus requesting user). Most of the checking is done by BIND at the time the original request is bound.

(c) Provides a means to recognize the source of the request. This is done by the authorization ID assigned to each user.

In order to simplify the operation of granting many different privileges to different users, DB2 provides a hierarchy of authorization levels that can be assigned to system administrators, database administrators, application programmers, operators and others. These are called administrative authorities. Thus, in DB2 there are two kinds of authorities: individual authorities and administrative authorities. Those who are granted administrative authorities possess the individual authorities that encompass them, but they also possess other authorities that cannot be granted individually.

A brief description of each administrative authority follows:

(a) Super SYSADM (Super System Administrator) - the user designated as super system administrator during the installation process is at the head of the authorization structure in DB2. The Super SYSADM has total control over all DB2 resources. No other user can revoke SYSADM authority from the Super SYSADM.

(b) SYSADM (System Administrator) - this is the highest level of administrative authority. Persons having SYSADM authority have total control over all DB2 resources (except certain critical subsystem table spaces and indexes that are only recoverable by the Super SYSADM). They can grant and revoke any of the other levels of authority and can grant any particular individual authority to any user and revoke any authority granted by any other user.

(c) DBADM (Data Base Administrator) - users with this administrative authority have total control over a particular DB2 database and have capabilities on objects within the database. A user can have DBADM authority over more than one database.

(d) DBCTRL (Data Base Control) - users having DBCTRL have all the capabilities of the DBADM except that they cannot automatically access the data in the database.

(e) DBMAINT (Data Base Maintenance) - users with DBMAINT authority have the read-only privileges of DBCTRL. This level of authorization is meant for personnel who perform such tasks as running utilities to make image copies and obtain statistics.

(f) SYSOPR (System Operator) - This level of administrative authority is meant for system operators who issue DB2 operator commands, but have no access to databases.

These authority levels allow system administrators to separate the task of database administration on a database-by-database level. Different groups can administer their own data without affecting any other database. The distribution of authority is a policy decision. It's up to the administrators to use the capability to distribute authorization that DB2 offers, or to have it centralized.

Users of DB2 that do not hold any of the special administrative authorities can create an object if they have been granted the authority to create that type of object. Regardless of how users gain the authority to create objects (tables, table spaces, indexes, etc.), they have full access to the objects they create, even if they do not have any of the administrative authorities.

As already mentioned, all users are identified by authorization IDs. A single user can have more than one authorization ID and, conversely, several users can share one authorization ID. To use any DB2 capability a user must be granted that privilege, and the privilege is granted to the user's authorization ID. A user's authorization ID entitles the user to ownership of the data in the objects he creates plus ownership of certain privileges. In addition, any privilege on the object can be granted to another user. Also, at the creator's option, privileges can be granted such that the grantee may in turn grant privileges to other users.

Access to DB2 resources is controlled by means of the GRANT and REVOKE statements. The GRANT statement enables you to give users certain privileges. The REVOKE statement enables you to take these privileges away. Only a privilege that has been specifically granted can be revoked. Instead of granting privileges to authorization IDs, as indicated above, you can also grant privileges to PUBLIC, which means that all users are granted those privileges. When a privilege is revoked from PUBLIC, Authoriation IDs that were specifically granted that privilege will still retain that privilege.

The general form of GRANT is:

```
GRANT some-privilege ON some-resource TO some-auth-ID
      [WITH GRANT OPTION]
```

The "WITH GRANT OPTION", if indicated, enables the grantee to further grant this privilege to other users. There are five forms of the SQL GRANT statement, and five forms of the corresponding REVOKE statements. The five forms that enable you to grant and and revoke privileges, and some examples of each one, follow:

(1) GRANT or REVOKE TABLE PRIVILEGES
- Retrieve data from a table or view (SELECT)
- Insert new rows in a table or view (INSERT)
- Delete rows from a table or view (DELETE)
- Update columns of a table or view (UPDATE)
- Create indexes for columns of a table (INDEX)
- Alter a table (ALTER)

(2) GRANT or REVOKE PLAN PRIVILEGES
- Use a BIND, REBIND, or FREE subcommand
- Run programs associated with a specified application plan (EXECUTE)

(3) GRANT or REVOKE SYSTEM PRIVILEGES
- Create databases and have DBADM authorization over the database created (CREATEDBA)

(4) GRANT or REVOKE DATABASE PRIVILEGES
- DBADM, DBCTRL and DBMAINT administrative authorities
- Operator commands to start, stop, and display a database (STARTDB, STOPDB, DISPLAYDB)
- Create a table (CREATETAB)
- Drop a database (DROP)
- Use DB2 utilities to:
  - load table spaces and indexes (LOAD)
  - recover table spaces and indexes (RECOVERDB)

(5) GRANT or REVOKE USER PRIVILEGES
- Authority to use a buffer pool
- Authority to use a table space

There are certain actions a user can perform (such as dropping a table) for which no explicit authority can be granted. The appropriate authorization IDs possess these privileges implicitly. A table can only be dropped by its creator, someone who has SYSADM authority, or by someone who has DBADM authority over the database that contains the table.

In addition to the authority hierarchy described, DB2 provides another feature to control access to data: the view mechanism. Anyone who creates a table can create a view of it. Also, a user can create views based on tables created by other users, provided he has SELECT privileges over those tables. The creator of a view is granted only the capabilities on the view that he or she holds on the table or tables on which it is based.

When a view is created DB2 inserts information in the authorization table of the catalog, for each table used by the view, as to which privileges the creator has on the view. When a user's authorization to use a table changes, the system checks the authorization table to see whether the user created a view based on that table. If a privilege is lost from the table, it will also be lost from the view. If as a result of this update, the user no longer has SELECT privilege on the view, the system deletes the view.

When a view is dropped, it does not affect the table (or tables) on which the view is based. However, when a view or table is dropped, the system will also automatically drop all views that are dependent on the views or tables being dropped.

DB2's catalog contains a series of system tables which maintain information about objects. Every time an object is created, DB2 makes appropriate entries in the catalog; a description of the object is entered along with cross-reference information showing how the object relates to other objects. The tables of interest to us are the ones that keep information about authorization. Some of these are:

SYSDBAUTH - identifies the privileges held by users over databases.

SYSTABAUTH - records the privileges held by users on tables and views.

SYSCOLAUTH - records the UPDATE privileges held by users on individual columns in a table or view. Entries are made in the table only when a GRANT statement specifies a column list following UPDATE authority.

SYSPLANAUTH - records the privileges held by users on application plans.

SYSUSERAUTH - records the system privileges held by users.

In general, the relation SYSTABAUTH consists of

USERID, TNAME, TYPE, UPDATE GRANTOP, PRIV

which have the following meaning:

USERID - is the user who is authorized to perform the action recorded in PRIV on the relation TNAME.

TNAME - is the name of the table (relation) to which USERID has been granted access to.

TYPE - indicates whether TNAME refers to a base relation (TYPE = R) or to a view (TYPE = V).

UPDATE - indicates whether the user USERID may perform this action on all columns of the relation (UPDATE = ALL), on some columns (UPDATE = SOME), or on none (UPDATE = NONE).

GRANTOP - indicates whether the privileges are grantable or not.

PRIV - is a sequence of columns, one for each privilege, which indicates (Y or N) whether the corresponding privilege may be exercise or not. In addition, a time stamp is kept in order to be able to control revocation of privileges.

For each (user, table) pair there are zero, one or two entries in SYSTABAUTH. There are zero entries if there are no privileges, one if the privileges are either with or without grant-option, and two if there are some privileges with and some without grant-option, in which case there is one tuple for all privileges with grant-option and one tuple for all without. The entry in SYSCOLAUTH is needed only if UPDATE = SOME, meaning that there are some columns on which the user USERID may exercise the privileges and there are others in which he may not. SYSCOLAUTH is used to record precisely on which columns that may be done: for each updatable column a tuple of the form

(user, table, column, grantor, grantopt)

is in SYSCOLAUTH.

Every time a GRANT command is issued, the system verifies whether the grant is authorized. If it is, either a new tuple is inserted in SYSTABAUTH or an old one is modified; the action on SYSCOLAUTH occurs according to the UPDATE field of the corresponding SYSTABAUTH tuple.

## 4.3. System/38

The IBM System/38 is a general-purpose data processing system designed to provide a high-level machine interface. It supports advanced database and interactive work station applications as well as traditional batch applications. The system's interface has an object-oriented architecture. Objects are the means through which information is stored and processed on the system. An object is defined as a named entity that is described by its set of attributes; the attributes define a set of functions or operations that can be performed on the object.

There are three basic user interfaces on System/38: the Control Language, RPG III, and data description specifications. Control Language (CL) allows for the establishment and control of the processing environment and provides an interface to many system dependent functions. RPG III allows the user to write a program such that objects like database files, work station files, and external data areas are not defined in the program. I/O operations are specified in the program, but the object structures (down to the field level) are defined via a separate interface. This separate interface is the 'data description specification'. It serves only as a descriptive interface since the actual object creation can only be affected via a CL command.

One of the main characteristics of System/38 is that it is object oriented. A description of the objects that could be defined in the system follows. Then we'll analyze how these objects are related to enforce access control rules.

Objects are devided into two classes of objects: system objects and program objects.

(1) System objects

(a) Data Space: serves as the basic unit of storage for a user's data. It consists of a collection of entries, each of which may contain a given number of similarly formated data fields.

(b) Cursor: provides access to the entries residing within a data space. A cursor is the user's only interface to these entries.

(c) Context: is an object that contains addressability by name, type, and subtype to other system objects.

(d) Access group: enables a user to specify (as a group) those system objects that are used together.

(e) Data Space Index: object that is used to provide a logical ordering of the entries in a data space.

(f) Program: object that forms the basic executable unit of the machine.

(g) User Profile: object that identifies a valid user to the machine. A user profile also identifies the user's rights to use the system objects, machine resources, privileged instructions, special machine instructions, and certain machine attributes.

Each system object must be explicitly created. There are specific Create instructions (for example, Create Cursor and Create Context) for each type of system object. A system pointer provides addressability to a system object. That is, in order to reference a system object for any operation, a system pointer must be specified as an operand of the instruction or implied as a field in a template.

(2) Program objects

(a) Data Object: object that provides operational and possibly representational characteristics to byte strings in spaces for use as instruction operations. There are two classes of data objects:

- Scalar data object: provides support for operations on values in space.

- Pointer Data Object: provides addressability to both program and system objects.

(b) Constant Data Object: defines a scalar data view that remains the same throughout the existence of a program.

(c) Entry Point: references an instruction as a target of the program invocation function.

A program object is created by the execution of the Create Program instruction. This instruction causes an object to be defined in the Object Definition Table (ODT). The entry in the Object Definition table that describes a program object is called a 'view'. A view defines the type, attributes, functional location, and, possibly, a permanent value or an initial value of the object.

Once an object is created, its internal stored format is not apparent to the user. The status and values of the object may be retrieved or changed by using interface instructions, but the internal format of the object cannot be directly viewed or modified.

Although the database supports security only on the object level, differences in the authorization required to use a Create Cursor instruction allow the implementation of field level security through the use of a subset of the fields. When a cursor is created to use a data space, the user must provide another description called the 'mapping template'. The mapping template describes how a user wants the entries to appear to the user program. The user may provide two mapping templates when the transformation of data is different for retrieval than for insert or update. Cursors can be created over multiple data spaces.

Following is a discussion of the kinds of access or authorities a user can possess over the objects described.

(1) Object Authorization - Instructions that involve system objects usually require certain authorities to those objects. During the execution of a program, instructions may require one or more of the following specific object authorities, which can be granted to users in any combination:

(a) Public Authority - can be used to control access by all users to the objects in the system. Three levels are available: ALL, eliminates security checks, all users can access the object; NO, only the object owner or other users whose profile contains a specific authority for that object can access it; NORMAL, allows all users to perform normal operations associated with the object, for example, normal operation for a program would be execution but not deleting or modifying.

(b) Object Management - required to control the accessability and availability of system objects, for example: modify addressability, grant/retract authority, create cursor, modify attributes.

(c) Data-related Authorities - control reading, writing, and general usage of system objects, for example: retrieve, insert, delete, update, authorized pointer, execute-only (for programs).

(2) Special authorizations - allow use of certain implicit authorities (that are not necessarily associated with one specific instruction or object), function authorities, and machine attribute modification authority.

(a) All-object authority - provides authority to use any object in the machine without public or private object authorization being granted to the object.

(b) Load - allows the user to copy an object from a load/dump media onto the machine.

(c) Dump - allows the user to copy an object from the machine to a load/dump media.

(3) Resource authorization - controls the amount of system resources that the user can utilize.

(4) Privileged Instructions - used to restrict creation or attribute modification of certain types of system objects for example: Create User Profile, Modify User Profile, Initiate Process, Terminate Machine Processing.

(5) Owned Objects - certain authorities are implied to the owner of a system object.

A prerequisite for authority verification is the identification of the user. This is satisfied through the use of an object called the User Profile, which identifies the user and the user's authority. A user profile has a name, type, and subtype that identifies a user to the machine in one of the following manners:

- All users of the system can be identified to the system by the same user profile.
- A group of users can share a single user profile.
- Each user of the system can have a unique user profile.

Two levels of users can be defined:

(1) System Security Officer - the system security officers have ultimate control of the use of the system. They can create a user profile and grant or revoke authority of that user to access any object or system function.

(2) User - whenever a users create objects, they become the owners of those objects. They have complete control over the objects. They can perform all of the functions of the officer in granting or revoking authority over the objects they own.

Each process in the machine executes under control of a user profile. When the machine executes an instruction, references an object, or requests a resource, it is done in the name of the user. Therefore, the user profile associated with a process is checked for the authority to use that item. When a process creates a permanent system object, the user profile associated with the process is assigned ownership of and all rights to that object. Temporary objects are not owned by any user profile, instead, all object authorities are granted to the public.

In addition to the authority provided by the process user profile, a process can 'adopt' other user profiles and, therefore, use any additional authority available to these user profiles. When a program is created, the creator can specify that the program is to have the adopted user profile attribute. He can also specify that it is to have the 'propagate adopted user profile' attribute, these causes the adopted user profile to be available to other programs called by the program with these attributes.

The symbolic address of an object used for address resolution may be qualified by a minimum authority requirement. If the authorities in the user profile does not agree with this minimum requirement, no access is permitted, and the system continues the search for other objects with the proper type, subtype, name, and authority.

The right to use a system object is monitored by the machine. Object authorization is implemented in microcode below the lowest user-available interface. The objects are stored in auxiliary storage, but there is no interface available to the user for reading from, or writing to, this medium. All operations are made through well-defined, object-sensitive, microcode instruction interfaces which enforce object authorization.

In order to reference a system object for any operation, a system pointer must be specified as an operand of the instruction or implied as a field in a template. When the pointer is first referenced the machine searches the entries of one or more contexts (which contain object names and locations) in order to locate the specified name. Once found, the resulting object location is stored in the pointer, thereby, eliminating subsequent searches (see figure 22).

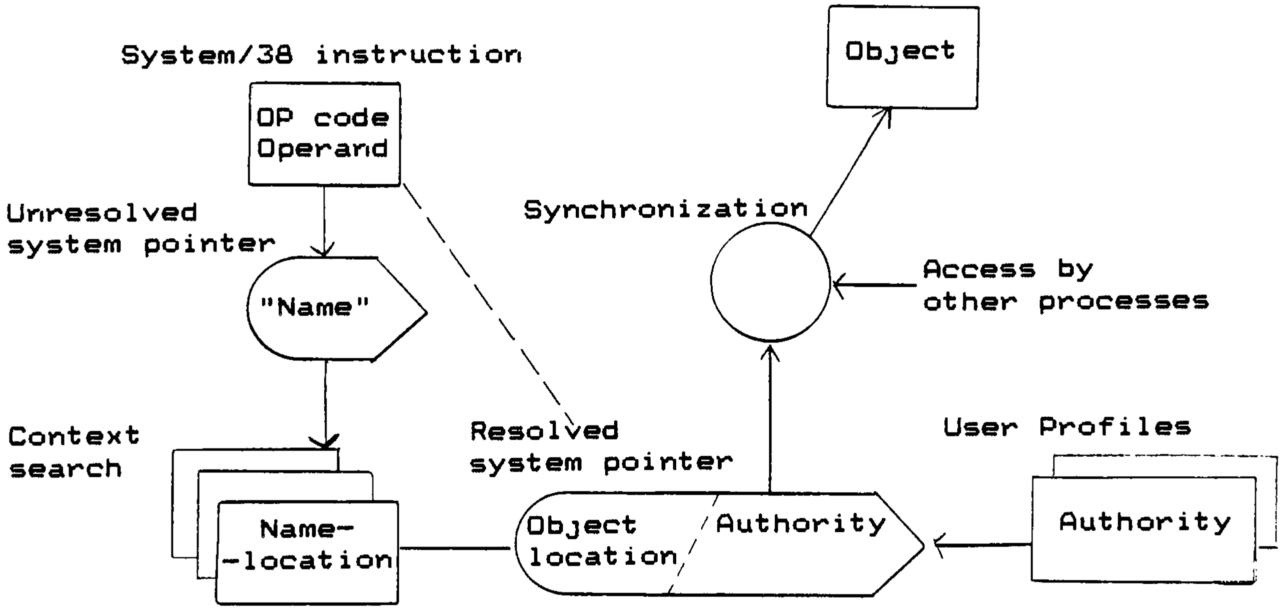


Figure 22
The object access path

For efficiency, the authority available to a user can be stored in the system pointer to the object. If the required authorization is contained in the pointer no further authorization checking is required. The user controls system pointer authority. The authority is not stored in the pointer unless the process has 'authorized pointer object' authorization to that object. Storing authority in the pointer has performance benefits, but it also has some functional disavantages. A system pointer with the authority attribute can be copied outside the process and therby cause implicit granting of authority to another process that may use the pointer or a copy of the pointer. Also, the authorization pointer can be saved indefinitely, this causes the Retract Authority instruction to be ineffective for such cases.

Pointers are protected against modification. A pointer is an object that is used only for addressing and does not permit examination or manipulation of the implied physical address. Direct modification of a pointer via a 'computational' instruction results in the pointer becoming invalid and no longer usable for addressing purposes.

## 4.4. Comparison of systems

Both INGRES and DB2 protection systems allow selective control of access to logical database subsets. Both schemes allow the designer of protection specifications to describe the logical database subset in high level, nonprocedural terms. That is, tools are provided to allow protection specifications to be written without requiring the specifier to write programs in a general purpose programming language. INGRES and DB2 use QUEL and SQL predicate expression capabilities, respectively, to facilitate this.

In INGRES, the qualification clause of a protection statement is used to identify a logical subset of the relation. QUEL allows selective access to column subsets via the specification of accessible columns. Access can be controlled in terms of the columns that are allowed to appear in the target list of an interaction as well as those that are allowed to appear in the qualification part of the interaction. QUEL protection statements are used to control access to information in a single relation only, although the subset of a relation that is accessible can be defined in terms of data in other relations.

In DB2, access control specifications for specific columns and rows of a table are described in terms of 'views', rather than in terms of base relation subsets. Each view has an associated definition that specifies how the view is constructed from the underlying relation(s). Row subsets of relations (and views) can, thus be specified. All columns of a view are readable to a user who has the appropriate privilege (read access to the view).

A subset of the columns of a view can be specified as those the user can update. In addition to the ability to define a view that is a row and/or column subset of a relation, one can define a view that summarizes the information in a relation. For example, a statistical view of the EMPLOYEE relation can be defined that has the sum of employee salaries by department (common value of Department). It is also possible to define a view that combines the information in two individual relations, e.g., the join of EMPLOYEE and DEPT on common Department.

Unlike INGRES, DB2's access privileges can grant access to information derived from base relations without necessarily granting access to the base relations themselves. In both INGRES and DB2, an interaction (query or update request) that involves more than one relation is constrained by the intersection of the relevant privileges on those relations. In INGRES, all relations involved in the interaction are base relations, whereas in DB2 one or more of them can be a view.

In supporting views, DB2 in effect handles many of the types of problems that can also be handled by limiting the operations a user can invoke. However, neither DB2, nor INGRES explicitly provide for control on the types of operations (transactions) a given user is allowed to perform. No consideration is given to the need to control what a user can do with data after it is released to him.

In System/38 indirect access control is achieved also by the definition of logical files (views). Once these files are created users are given access rights to them. These logical files are defined in a similar way as DB2 views, in that they can be formed of a subset of columns or rows. But, like INGRES, they are based only on physical files. No logical files can be defined over other logical files.

DB2 and System/38 approach to controlling access to aggregate quantities (e.g., the average employee salary) is to allow a user access to a view (or logical file) created for each relevant aggregate (e.g., one for the average employee salary, one for the salary total, etc.). INGRES handles access to aggregates in a more restrictive way: if the user does not have access to the data underlying the aggregate (e.g., access to all employee salaries, if he wants to ask for the average employee salary), then either:

(1) he can be allowed to examine aggregate information only if the aggregate is taken over a complete relation, or

(2) the aggregate is modified so that it ranges over a subset of the underlying relation to which the user has access.

The database administrator decides for each aggregate type which of the two above alternatives is to apply.

DB2, INGRES, and System/38 allow access privileges to be dynamically issued and revoked. For the revocation of access privileges to be immediate, each access to the data must be checked for authority. In DB2, revocation does not actually have an effect on a user until after the transaction he is currently executing has completed. This is done because of the desire to make the transaction the basic unit of consistency, concurrency, recovery, and authorization in DB2, and because it is desirable to minimize the number of authority checks the protection system must perform.

Because it is not transaction oriented, INGRES revokes access from a user after the current interaction is completed.

In System/38 some problems may arise if the authorization information is stored in the system pointer. Since this pointer could be indefinitely saved, the authorization could be in the system even if it has been removed from the user profile.

The ability to dynamically issue and change protection specifications in INGRES is centralized: the database administrator creates and issues all protection statements on shared relations. The ability to control protection specifications in DB2 is distributed: any user can create a relation and grant access privileges on it. A user can furthermore pass on the ability to grant a privilege to another user (the "grant option", which is related to the "copy flag").

In the INGRES scheme the database administrator can be a potential bottleneck if protection specifications are rather highly dynamic. The database administrator is given an excessive amount of power, which is not consistent with the concept of separation of privileges (i.e. distributing power). The centralized approach may, however, be appropriate in a given application environment: the database administrator can balance conflicting requirements. DB2 avoids this problem, but at the expense of added overhead (e.g., in revocation). The INGRES centralized approach has one additional advantage: it prevents the user from copying data to which he has access and subsequently granting access to it to others (who presumably should not be allowed such access).

System/38 provides also distributed authorization in that users can grant privileges to other users over objects they own. But there is a central security officer or DBA who controls all objects in the system and who has ultimately authorization over them.

The DB2 and INGRES protection schemes are based on the user oriented data selection and modification languages SQL and QUEL, respectively. The advantages and disadvantages of these two languages are to some extent inherited by the corresponding protection system. Both systems have relatively simple protection mechanisms, which is a plus with regard to the effectiveness of the user interface.

Both DB2 and INGRES protection systems are basically access control list systems, because they maintain lists of which users have which access privileges. In DB2, the user is responsible for choosing the correct view for his interaction. He can obtain different access rights if he phrases his interaction in terms of one view than he would obtain if he phrases it (equivalently) in terms of another. By contrast INGRES automatically determines which access privileges are relevant to a given interaction; the user has no choice to make. The apparent tradeoff is between the flexibility of the DB2 approach and the ability of INGRES to automatically determine all relevant access rights for a given interaction.

System/38 takes a different approach, it utilizes the capability list technique. On each user profile there is a list of the objects the user is allowed to access together with the corresponding authorities he has to each object.

In the INGRES scheme, the need for a larger number of protection statements with a large number of interactions is a problem. For example, it is required that the protection statements for a user be properly "nested": the more columns named in the protection statement (present in the target list and qualification column list), the more restrictive is the allowed access. In INGRES, it is necessary to repeat the complete specification of an access right for each user to whom it is to be given, whereas in DB2 each distinct view is defined only once.

Due to use of query modification, an INGRES user can be supplied with an answer to a question different from the one he asked; the user should normally be notified of this difference. For example, a request for the average employee salary may be modified to yield the average salary of employees in the toy department only. DB2 takes the approach of reporting all protection violations to the user, denying him the requested access.

In System/38 logical files are defined only once, but since it uses the capability list, authorization specifications for each object is entered on each user profile. One way to minimize this repetition is by defining users in groups and assigning a user profile for the group. All members of the group have the same access privileges to the specified objects.

DB2, INGRES, and System/38 have fail safe protection defaults, in that the default is to deny access if the required privileges are not explicitly present.

INGRES, DB2, and System/38 rely on the operating system on which they are built for user authentication, and reliability of operation. The operating system is also relied upon for primary memory protection (e.g., clearing storage residues), controlling the hardware for reliability, etc.

## 5. Conclusions

### 5.1. Concluding remarks

This paper has presented the techniques that has been used to model protection requirements, in particular, access control information in a database management system. These tools aid the designer to express the relationship between the components of a security system, e.g., users (subjects), objects, and the set of rules that governs who can access which data. These models should help him find the best way to enforce the security requirements at the time of implementation.

The access matrix has shown to be a good means to view all objects, subjects , and the rules that relate them. But precisely because it is a view of every component, it can be too sparse to implement. Here is where two variations of the matrix become useful. The authorization list and the capability list systems have shown to be more suitable and realistic to implement. In both cases protection data is maintained but the distinction is whether it is maintained by user (capability list) or in relation to objects (authorization list). In capability systems, each subject must keep a collection of keys for the objects of interest, keys issued by the owners of these objects with possible granting privileges. These systems require each subject to present the correct key whenever a privilege is requested.

In access-list systems a list of authorized users is kept with each object. The security system checks this list whenever access is requested and oversees that all access conditions are met. The subject only needs to provide identification/authentication information which will lead to the initialization of his profile. For this user-convenience reason, such systems could be appealing for database management systems. The authorization list systems permit the revocation of privileges without consultation with the user who is losing the privilege. They permit an auditor to determine the range of access without having the access himself.

In both authorization list and capability list schemes, what is controlled is access to the container of the data rather than access to the data itself. Access control mechanisms could not be based only on the type of access (e.g., read, write, delete). More complex controls are required in a general-purpose DBMS to ensure:

(1) container-dependent security (e.g. "user U cannot access the domain SALARY")

(2) value-dependent security (e.g. "user U can retrieve the domain SALARY if, and only if, salary value is less than $2000")

(3) context-dependent security (e.g. "user U cannot retrieve the domain SALARY together with the domain NAME")

These controls have to be stated in a "logical form" and cannot be provided by a straightforward capability or authorization list mechanism. Two basic mechanisms have been defined to implement access control at the external level of a general-purpose DBMS: views and query modification. The view mechanism offers the user the sensitive data he has the right to manipulate; other data are hidden. Such a view is built (dynamically or statically) from the global schema and the security constraints.

In the query modification mechanism, the user request is considered a "virtual" one. The actual interaction with the database takes place after appending (logical AND) the concerned security constraints to the user request.

Whatever approach is taken to enforce management security policies, the final protection system should help certain characteristics, or should have attained a series of goals. We'll summarize the goals by which database protection systems should be evaluated.

(1) The granularity of the protection specifications must be variable; it must be possible to selectively control access to arbitrary logical subsets of a database. In a database management system there are no fixed boundaries between protected objects. In a given database application environment, it may be necessary to selectively permit access to only certain records in a file, or allow a user to examine or update a subset of the fields of a record. Furthermore, a protection specification for a database can refer to information anywhere in the database (e.g., correlating information in several files).

(2) The protection specifications must facilitate granting access to derived information. It must be possible to control access to data that is calculated from

information in the database, such as statistical summaries or logical reorganization of the information.

(3) It should be possible to control the particular types of domain specific operations (operation specific to a particular application domain) that can be performed on a database. It is not always sufficient to be able to specify what data a user can access, but rather specifically what he is allowed to do with it. It is sometimes useful to be able to limit a user to certain types of database operations.

(4) Dynamic control of the protection specifications is necessary, as the needs of an application environment are not constant. It must be possible to selectively grant and revoke access privileges, as well as to immediately revoke an access privilege, if this becomes necessary.

(5) The distribution of authority to control protection specifications must provide for flexibility, e.g., by allowing users to grant access to information they "own". It is desirable to minimize the amount of trust that must be given any individual. However, since a shared database is involved, conflicting requirements must somehow be balanced.

(6) An effective user interface to the protection system must be provided. The interface must deal with administrators who issue the protection specifications, as well as with users who encounter access restrictions (and potentially violate them). The central problem is that of issuing and maintaining protection specifications. The goal is a scheme that is easy to use and easy to learn, although these requirements often conflict.

(7) The protection scheme must be flexible and generally applicable, i.e., specializable to a variety of application environments. The scheme must satisfactorily support complex protection specifications, but must also not be cumbersome when careful control is not warranted.

(8) The protection scheme and its underlying mechanism must be reliable. Users need to have confidence in it. It must have fail safe defaults, such as a default to deny access. The protection scheme and a set of protection specifications based on it must be auditable. The design of the mechanism should be open, it the sense that its success does not depend on user ignorance.

(9) The protection mechanism must be cost effective. The run time costs are particularly crucial for database management systems, as repetitive operations are commonplace. An inefficient protection mechanism can tremendously degrade the effectiveness of a database management system; if the cost of employing a protection system is too high, it may not be used despite the need for privacy controls.

Many of these requirements may conflict, e.g., flexibility, the adequacy of user interface, and the cost of use. Consequently, balances, tradeoffs, and compromises must be considered in practice.

A number of software packages provide access control and related functions. The market for these programs appears to be growing as users become more concerned about security. Examples are ACF2, RACF, and Top Secret. Typical functions provided by such packages are authenticating users, maintaining access control information, checking authorization to use files or other objects, logging, and producing reports. RACF (Resource Access Control Facility) for example, provides access control by: (1) Identifying and verifying system users, (2) Authorizing access to system resources, (3) Logging and reporting of unauthorized attempts to enter the system and to access to protected resources. RACF maintains an access control list for each object. Users may belong to groups and receive all the privileges of their group. RACF is basically an open system in that resources not defined to RACF are not protected. ACF2 is a closed system, in which resources not defined to ACF2 are protected. RACF can, however, provide closed system protection for any specified set of resources.

## 5.2. Future Trends

The presence of protection mechanisms does not guarantee security. Penetration analysis (sometimes called the "tiger team" approach) has helped locate security weaknesses. But, like program testing, it does not prove the absence of flaws. Even with advanced technology for developing and verifying systems, it is unlikely systems will be absolutely secure. Computer systems are extremely complex and vulnerable to many subtle forms of attack.

There is one architectural approach that could help in the verification of a security system. The objective of it is to isolate the access checking mechanism in a small system nucleus responsible for enforcing security. The nucleus, called a "security kernel", mediates all access requests to ensure they are permitted by the system's security policies. The security of the system is established by proving the protection policies meet the requirements of the system, and that the kernel correctly enforces the policies. If the kernel is small, the verification effort is considerably less than that required for a complete operating system.

But there are other considerations to be taken, like the trust between users of the database. The complete solution of the protection problem must find a balance between technical and nontechnical issues. More research is required in both issues, especially on methods of detecting and reporting violations and on coordinating external regulations with internal protection mechanisms.

Database security will continue to be recognized as an important goal. One can expect to see some transformations of this goal, however, for a number of reasons. First, new uses (such as the growing use of distributed systems) for databases will impose new requirements. Second, changes in hardware and software technology may eliminate some of the old problems while introducing new ones.

Office systems, offer a whole new set of problems, since their databases consist largely of text and graphic material (which may be noncoded information). Text and graphic objects have complex structures and therefore may consist of smaller objects of differing authorization characteristics.

Further research will be seen in security of commercial systems, distributed databases, inference, theory of authorization, and architecture for secure database systems, as well as in the field of database audit and control, especially as it relates to the DBMS.

## Appendix I - Glossary

Access - the ability and the means necessary to approach, to store or retrieve data.

Access Control - the process of limiting access to the resources of a computer system only to authorized users, programs and processes.

Access Control Mechanism - hardware or software features, operating procedures, management procedures and various combinations of these designed to detect and prevent unauthorized access and to permit authorized access to a computer system.

Access Type - the nature of an access to a particular resource; e.g., read, write, execute, append, delete, create.

Append Only - a class of access privilege that permits the user to add data only, not to read or write.

Authentication - the act of identifying and verifying the eligibility of an individual to access specific categories of information.

Authorization - the granting to a user, program or process the right of access.

Compartmentation - keeping the data and programs of different desired accessibility separated from each other, separating the resources available to concurrent users.

Database - (1) a single file containing information in a format applicable to any user's needs and available when needed. (2) all information required to process a set of one or more applications.

Database Administrator - an EDP manager charged with approving the design and implementation of database management systems.

Database Management System (DBMS) - the totality of all routines that provide access to data, enforce storage conventions and regulate the use of input-output devices for a specified database.

Data-dependent protection - protection of data at a level commensurate with the sensitivity level of individual data elements, rather than with the sensitivity of the entire file which includes the data element.

Data security - the protection of data from accidental or malicious modification, destruction or disclosure.

Execute Only - a type of access to a program that confers upon the user the right to run it but not to read the code nor alter it.

Hand Geometry - an identifying code derived from the length of the fingers, the shape of finger endings, and the translucency of the skin.

Identification - the process that enables, generally by the use of unique machine-readable names, recognition of users or resources as identical to those previously described to a computer system.

Login - a physical procedure in which a terminal user is identified to the computer operating system prior to any processing.

Need-to-know - a requirement for a person to receive information in order to perform his duties.

Password - a protected word or a string of characters that identify a user. Synonymous with codeword, keyword, lock-word.

Query Language - a high-level language designed for a specific set of transactions to interrogate a database.

Security - mechanisms and techniques that control who may use or modify the computer or the information stored in it.

User - individual who is accountable for some identifiable set of activities in a computer system. May refer to a natural person, an entity (program or process) possessing privileges equivalent to those of a natural person, or a group of persons (class or project).

## BIBLIOGRAPHY

[BERT78 ] Bertis, V., Truxal, C. D., and Ranweiler, J. G., "System/38 addressing and authorization", IBM System/38 Technical Developments , IBM Corporation, 1978, 51-54.

Discusses addressing, authorization, locking and synchronization in System/38.

[CARR77 ] Carroll, J. M., Computer Security. Boston: Butlerworth Publishers Inc., 1977.

A comprehensive overview of security topics such as threats, security management considerations, physical security, communication security, and systems security.

[CHAM75 ] Chamberlin, D.D., Gray, J. N. and Traiger, I. L., "Views, Authorization, and Locking in a Relational Data Base System", AFIPS Conference Proceedings. California (May 1975), 425-430.

This paper describes a mechanism that supports the concepts of views, authorization, and locking, within the context of SEQUEL language of the System R relational database system. The authors present examples of how SEQUEL is used, in particular the way in which user views are defined and how authorization and locking may be supported by including access qualifiers in the view definition.

[CHAM78 ] Chamberlin, D. D. et al., "Data Base System Authorization", Foundations of Secure Computation, R. A. DeMillo ed. New York: Academic Press Inc., 1978, 39-52.

The author gives an overview of authorization related topics such as authentication, centralized vs. decentralized authorization and revocation of privileges.

[CLAY83 ] Claybrook, B. G., "Using Views in a Multilevel Secure Database Management System", IEEE Proceedings of the 1983 Symposium on Security and Privacy. California (April, 1983), 4-13.

In this paper the author discusses the use of views in database management systems that enforce user level discretionary and nondiscretionary access control policies. This discussion involves several issues such as how should views be classified, what type of mechanism should be used to define views, mapping between views, etc.

[CONW73 ] Conway, R. W., Maxwell, W. L. and Morgan, H. D., "On the Implementation of Security Measures in Information Systems", Security and Privacy in Computer Systems. California: Melville Publishing Company, 1973, 244-266.

A "security matrix" is proposed, which is equivalent to the access matrix of the basic model. This is used as a functional model of a security system, which separates data-dependent and data-independent access decisions. It proposes compile-time access decisions.

[CONW78 ] Conway, A. J., and Harvey, D. G., "User-System/38 interface design considerations", IBM System/38 Technical Developments , IBM Corporation, 1978, 70-73.

Provides an overview of System/38 primary user interface. Describes objectives, constraints, and rationale.

[DAHL78 ] Dahley, S. H, et. al., "System/38 high-level machine", IBM System/38 Technical Developments, 1978, 47-50. Characterizes the System/38 high-level machine instruction interface. Describes microcode functions and the rationale for providing them.

[DALE74 ] Daley, R. C. and Donohue, J. P., "Security and Authorization - Semantics and Examples", Data Security and Data Processing - Study Result: MIT, Vol 4. IBM Corporation, (June, 1974), 135-150.

Result of an MIT study on security facilities of IBM's RSS (Resource Security System). Discusses need of decentralization of authorization functions, grouping of users and data, program-to-file authorization, relevant access types for this environment, and a list of suggestions on how to improve that system.

[DATE83 ] Date, C. J., An Introduction to Database Systems, Vol 2. Reading: Addison-Wesley Publishing Company, 1983.

Covers systems aspects such as recovery, integrity, concurrency, and security.

[DATE84 ] Date, C. J., A Guide to DB2. Reading: Addison-Wesley Publishing Company, 1984.

Presents a detailed description of the DB2 IBM product. It has comprehensive coverage of the SQL language and a description of the relational database model.

[DENN82 ] Denning, D. E., Cryptography and Data Security. Reading: Addison-Wesley Publishing Company, 1982.

Cryptographic techniques, information flow controls, inference controls and access controls. In relation to this last topic it describes the basic principles of mechanisms that control access by subjects to objects. Discusses the access-matrix model, authorization list and capability list.

[FAGI78 ] Fagin, R., "On an Authorization Mechanism", ACM TODS 3,3 (September, 1978), 310-319.

Makes modifications to the model by Griffiths and Wade and proves correctness for the modified mechanism.

[FERN75 ] Fernandez, E. B., Summers, R. C. and Coleman, C. D., "An Authorization Model for a Shared Database", Proceedings 1975 ACM SIGMOD International Conference, San Jose, California. (May, 1975), 23-31.

The authorization model for the LASC system. The model includes: (1) the use of applications as context for user rights, (2) the use of predicates that can depend on any data in the system as part of the access rules, and (3) the use of ordered access types. Enforcement of this model at compile time is discussed.

[FERN81 ] Fernandez, E. B., Summers, R. C. and Wood, C., Database Security and Integrity. Reading: Addison-Wesley Publishing Company, 1981.

In depth presentation of security and integrity issues of information maintained in databases.

[FRIE73 ] Friedman, T. D., "The Authorization Problem in Shared Files", Security and Privacy in Computer Systems. California: Melville Publishing Company, 1973, 159-185.

Presents the problem of authorization in shared systems. A hypothetical model of an authorization system is described.

[GAIN78 ] Gaines, R. S. and Shapiro, N. Z., "Some Security Principles and their Application to Computer Security", Foundations of Secure Computation. R. A. DeMillo ed. New York: Academic Press, 1978, 223-234.

The authors examine general ideas of security and apply them to the problem of computer security. They point out that the chief method for computer security has been the barrier, i.e., the access control mechanism.

[GOGU82 ] Goguen, J. A. and Mesequer, J., "Security Policies and Security Models", IEEE Proceedings of the 1982 Symposium on Security and Privacy. California (April, 1982), 11-20.

The paper describes an approach to security which is based on: modeling the information processing system as a capability system, defining security policies as a set of noninterference assertions and verifying that a given system satisfies a given policy.

[GRAH79 ] Graham, G. S. and Denning, P. S., "Protection - Principles and Practices", AFIPS Proceedings of the Spring Joint Computer Conference, Vol 40 (1979), 417-429.

Lampson's model is interpreted in more detail. In particular, creation and deletion of objects, granting of access to objects, and sharing by untrustworthy subsystems are covered. An argument for the correctness of the model is also given, and ways of implementing it are discussed.

[GRIF76 ] Griffiths, P.P. and Wade, B. W., "An Authorization Mechanism for a Relational Database

System", ACM TODS 1, 3 (September, 1976), 242-255.

Covers authorization system for System R. The paper discusses authorization commands, representation of the authorization information, granting and revoking rights, authorization checking, and the use of views for authorization.

[HENR78 ] Henry, G. G., "Introduction to IBM Sysyem/38 architecture", IBM System/38 Technical Developments. IBM Corporation, 1978, 3-6.

Support functions are summarized and architecture concepts are described.

[HOFF77 ] Hoffman, L. J., Modern Methods for Computer Security and Privacy. New Jersey: Prentice-Hall, Inc., 1977.

The book covers topics as authentication and identification, authorization policies, logging, traditional and modern methods of cryptography, operating systems, mathematical models, computer security (operational and physical concerns) and legal aspects of computer privacy.

[HSIA79 ] Hsiao, D. K., Kerr, D. S. and Madnick, S. E., Computer Security. New York: Academic Press, Inc., 1979.

The book addresses computer security topics as privacy, operational security, physical security, hardware security, cryptography, operating systems security and database security. On this last topic it discusses access decisions, access paths, access authorization and implementation of security features.

[IBMC80a] IBM System/38 Planning and Review Controls, IBM Corporation. November, 1980.

Report result of a study analysis to be used as a guide for planning and reviewing controls by management and auditors of the organization possessing a System/38.

[IBMC80b] IBM System/38 Functional Concepts Manual, IBM Corporation. June, 1980.

Product manual that describes the functions provided by the System/38 interface machine.

[IBMC84 ] Database 2 System Planning and Administration Guide, IBM Corporation. San Jose: Programming Publishing, 1984.

Product manual that describes how to design DB2 databases, and how to provide security, among other design issues.

[JONE78 ] Jones, A. K., "Protection Mechanism Models: Their Usefulness", Foundations of Secure Computation. R. A. DeMillo ed. New York: Academic Press, Inc., 1978, 237-252.

An extensive presentation of the Take-Grant system, as well as an evaluation of its usefulness to solve security problems.

[LAND81 ] Landwehr, C. E., "Formal Models for Computer Security", ACM Computing Surveys, 13, 3 (September 1981), 247-278.

Quite exhaustive presentation of the mostly used security models is presented, including access matrix, Bell and LaPadua, and Take-Grant among others.

[LAMP75 ] Lampson, B. W., "Protection", Protection of Information in Computer Systems. IEEE COMPCON, September 1975, 215-221.

Original access matrix model presentation which served as a base for further development of such model.

[LEIS82 ] Leiss, E. L., Principles of Data Security. New York: Plenum Press, 1982.

The author discusses, with a rather theoretical and mathematical approach, issues on security of statistical databases, authorization mechanisms, cryptosystems, and user's rights and limitations on access to certain data.

[LIPN82 ] Lipner, S. B., "Non-discretionary Controls for Commercial Applications", IEEE Proceedings of the 1982 Symposium on Security and Privacy. California (April, 1982), 2-10.

Presents a possible application of the lattice model (military security system) to commercial systems.

[McLE77 ] McLeod, D., "A Framework for Data Base Protection and its Application to the INGRES and System R Data Base Management Systems", COMPSAC77 Proceedings, Chicago, (November, 1977), 342-348.

This paper introduces a set of requirements for protection in a database environment, which include: variable granularity of access rules, control of access to derived information, limiting access to operations, and the ability to distribute authority. INGRES and SYSTEM R are then discussed in terms of these criteria.

[MIRA80 ] Miranda, S. M., "Aspects of Data Security in General-purpose Data Base Management Systems", IEEE Data Security and Privacy, California, (April, 1980), 46-58.

Presents the various aspects of data security in a general-purpose database management system (DBMS) in contrast with those of operating systems.

[PETE79 ] Petersen, H. E. and Turn, R., "System Implication of Information Privacy", Security and Privacy in Computer Systems. California: Melville Publishing Company, 1981, 76-95.

The author makes an exposition of privacy issues in the context of shared and distributed systems.

[PINN78 ] Pinnow, K. W., Ranweiler, J. G., and Miller, J. F., "System/38 object-oriented architecture", IBM System/38 Technical Developments. IBM Corporation, 1978, 55-58.

Discusses the concepts, purpose, and characteristics of System/38 machine objects.

[SALT75 ] Saltzer, J. H. and Schroeder, M. D., "The Protection of Information in Computer Systems", Protection of Information in Computer Systems. IEEE COMPCON, (September, 1975), 57-200.

The paper provides a comprehensive overview of computer protection methods. The paper describes functions, design principles, examples of elementary protection and authentication mechanisms, principles of protection architecture and the relation between capability systems and access control list systems.

[SCHA77 ] Schaefer, M., "On Certain Security Issues in Computer Systems", The ANSI/SPARC DBMS Model. D. A. Jardin ed. North-Holland Publishing Co., 1977, 131-146.

The concept of controlled data sharing is examined in the context of volatile databases. In particular, alternative data classification schemes, commercial and military, and their implementational implications are discussed.

[SHAR79 ] Sharma, K. D. and Sharma, Y. R., "An Access Control Facility for Relational Data Base Systems", Information Systems, Vol 4, no 2. Great Britain: Pergamon Press LTD, 1979, 33-39.

A set of language facilities for the specification of security requirements in a relational database is presented. An attempt is made to have a database language unified from the point of view of data storage/retrieval, computation and protection.

[STON74 ] Stonebraker, M. "Access Control in a Relational Data Base Management System by Query Modification", Proceedings 1974 ACM National Conference. San Diego, California, (November, 1974), 180-184.

Presents the access control system being implemented in INGRES. Examples using the QUEL language commands are used to present the system.

[STON76a] Stonebraker, M. and Rubinstein P., "The INGRES Protection System", Proceedings 1976 ACM National Conference. California.

The design decisions for the security system of INGRES are justified. In particular, the paper discusses the role and power of the DBA, the protection language, reasons for a centralized DBA, the policy about access violations, and the reason for not having auxiliary procedures.

[STON76b] Stonebraker, M., "The Design and Implementation of INGRES", ACM TODS, Vol 1, No. 3, September 1976, 189-222.

The INGRES database management system is described. Design decisions, support for integrity constraints, views and protection are discussed.

[SUMM84 ] Summers, R. C., "An Overview of Computer Security", IBM System Journal 23, no. 4, (1984), 309-325.

Presents an overview of computer security including concepts, techniques, and measures relating to the protection of computing systems and the information they maintain. Security strategies are considered. Security models are surveyed.

[TURN75 ] Turn, R. and Ware, W. H., "Privacy and Security in Computer Systems", Protection of Information in Computer Systems. IEEE COMPCON, (September, 1975), 49-56.

Discusses in general security concepts such as privacy, identification, computer security, access control, integrity, and protection costs.

[WALK77 ] Walker, B. J. and Blake, I. F., Computer Security and Protection Structures. Pennsylvania: Dowden, Hutchinson & Ross, Inc., 1977.

Presents security threats and countermeasures including physical safeguards and file safeguards. Discusses access control levels, storage of access control information, and enforcement schemes for access control.

[WATS78a] Watson, C. T., and Aberle, G. F., "System/38 machine database support", IBM System/38 Technical Developments. IBM Corporation, 1978, 59-62.

Describes the database machine support and discusses performance, security, integrity, and ease-of-use considerations.

[WATS78b] Watson, C. T., Benson, F. E., and Taylor, P. T., "System/38 data base concepts", IBM System/38 Technical Developments. IBM Corporation, 1978, 78-80.

Describes major database characteristics and their rationale.

[WOOD79a] Wood, C. and Fernandez, E. B., "Decentralized Authorization in a Database System", Fifth International Conference on VLDB, Brazil, 1979, 352-359.

A model of authorization for decentralized administration is developed. Mechanisms for the delegation of administration and its revocation are presented. A possible scheme for the distribution of authorization-related data through a distributed processing network is described.

[WOOD79b] Wood, C., Summers, R. C. and Fernandez, E. B., "Authorization in Multilevel Database Models", Information Systems, Vol 4, no. 2. Great Britain: Pergamon Press LTD, 1979, 155-162.

Analyzes the consistency of authorization rules written at the conceptual level and at the external level of a multilevel database architecture. The validation of access requests against authorization rules is also discussed.

[WUMS81 ] Wu, M. S., "Hierarchical Protection Systems", IEEE Proceedings of the 1981 Symposium on Security and Privacy. California, (April, 1981), 113-124.

The Take-Grant model developed by Jones, Lipton and Snyder is extended in order to present and study two hierarchical protection systems: tree systems and acyclic systems.

I, Diana Angleró, prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

1502 Farrar St.
Antonsanti
Rio Piedras, Puerto Rico 00927

Date: August 12, 1985