A Protection Model for Distributed Data Base Management Systems

by

Atika Laribi

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

APPROVED:

Dennis G. Kafura, Chairman

H. Rex Hartson

Edward A. Fox

James O. Hicks

James R. Armstrong

May, 1985

Blacksburg, Virginia

A PROTECTION MODEL FOR DISTRIBUTED DATA BASE MANAGEMENT SYSTEMS

by

Atika Laribi

Dennis G. Kafura, Chairman

Computer Science

(ABSTRACT)

Security is important for Centralized Data Base Management Systems (CDBMS) and becomes crucial for Distributed Data Base Management Systems (DDBMS) when different organizations share information. Secure cooperation can be achieved only if each participating organization is assured that the data it makes available will not be abused by other users. In this work differences between CDBMS and DDBMS that characterize the nature of the protection problem in DDBMS are identified. These differences are translated into basic protection requirements. Policies that a distributed data base management protection system should allow are described. The system proposed in this work is powerful enough to satisfy the stated requirements and allow for variations on the policies. This system is a hybrid one where both authorizations and constraints can be defined. The system is termed hybrid because it combines features of both open and closed protection systems. In addition the hybrid system, although designed to offer the flexibility of discretionary systems, incorpo-

rates the flow control of information between users, a feature found only in some nondiscretionary systems. Furthermore, the proposed system is said to be integrated because authorizations and constraints can be defined on any of the data bases supported by the system including the data bases containing the authorizations, and the constraints themselves. The hybrid system is incorporated in a general model of DDBMS protection. A modular approach is taken for the design of the model. This approach allows us to represent the different options for the model depending on the set of policy choices taken. Three levels of abstraction describing different aspects of DDBMS protection problems are defined. The conceptual level describes the protection control of the DDBMS transactions and information flows. The logical level is concerned with the interaction between the different organizations participating in the DDBMS. The physical level is involved with the architectural implementation of the logical level.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Kafura for his invaluable guidance during this research. This work would never have been completed without the long hours he spent discussing with me the research with such an untiring patience. I am also very grateful to Dr. Kafura for always being ready to discuss and solve even the smallest problem.

I would also like to thank Dr. Hartson for his valuable insights throughout this work, and for the many useful discussions we had about this research.

I also thank Dr. Fox, Dr. Hicks and Dr. Armstrong for serving on my committee.

A very special thank to my husband, , whose moral support helped me start, pursue and finish this research. His unconditional support turned out to be a key factor in the completion of this work.

My daughter deserves also to be thanked for giving me all those little joys, only a baby can bring, which brightened my days and allowed me to be more productive.

And last, but not least, I would like to thank my brother and my sister in law for inspiring me and encouraging me to pursue a higher education.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 CHAPTER 1 INTRODUCTION AND LITERATURE REVIEW

1.1 INTRODUCTION

In the 1960's and 1970's, the rapid proliferation of computing systems for business and management created the need for flexible and general purpose Data Base Management Systems (DBMS). As increasing amounts of data were made available to users in geographically dispersed locations and as the number of such users increased, the need for distributed processing became evident. The falling price of computing equipment and the improvement of computer communications during the late 1970's and early 1980's allowed more and more organizations, small as well as large, to use inter computer communications to share data resources. These systems represented extensions of centralized DBMS into Distributed Data Base Systems (DDBMS).

Beyond the advantages of a centralized DBMS, a distributed DBMS can improve reliability since the processing is shared by many computers located at different sites and is not, therefore, dependent on a single machine. The upward scaling of data base capacity is also easier when the data base is distributed, since this can be accomplished by simply adding new sites to the network supporting the DDBMS.

Security is important for a centralized DBMS and becomes crucial for a DDBMS involving different organizations. Secure cooperation can be achieved only if each participating organization is assured that the data which it holds is not abused by other users. Most of the security problems that arise in DDBMS also occur in centralized DBMS. However, these problems are much more accute in the distributed case and require solutions different from the ones used for centralized systems.

Much work has been done in the area of DDBMS. Likewise the area of Security in centralized Data Bases has been widely investigated. However very little research has been conducted in the important area of Security in DDBMS which is the converging point of the two. This work makes a contribution to bridging the gap between DDBMS and Security in centralized Data Bases by identifying the problems associated with Security in DDBMS and solving some of the most critical ones.

After listing the main issues in DDBMS, we will in this chapter review the few works in the area of DDBMS protection to put our work in perspective. Before designing a DDBMS protection model, we need to make choices about the data and protection models to be used. To justify the choices we made, we will analyze the literature for these areas. After stating the assumptions we make throughout this work, we will conclude this chapter by presenting an overview of the organization of the dissertation.

1.2 MAIN DDBMS ISSUES.

The main issues specific to DDBMS are related to the fields of architecture, query processing , data partitioning and replication, concurrency control and synchronization, and protection. All these areas with the exception of protection have been investigated extensively in the last few years. Although in this work we concentrate on the protection issues, the areas of query processing, data partitioning and replication as well as concurrency control and synchronization have to be integrated in our design. However, since these areas have been thoroughly studied, we will not investigate them further in this work.

1.3 DDBMS PROTECTION WORKS.

To place the contribution of this work in perspective, we will review four papers in the area of protection of DDBMS, which have each a different approach to the problem.

1. BUSSOLATI'S SECURITY MANAGEMENT SYSTEM FOR DDBS

   In (BUSSO80) two approaches for Distributed Data Base Design are defined : ascending and descending which correspond to bottom-up and top-down designs. An architecture is then defined for an aggregate Distributed Data Base (different local schemas). Each local data base keeps its own schema. Three levels at the local site are defined, accordingly to the ANSI model: internal, conceptual and external. Part of the local data is shared with the network. However, local users are distinct from network users. The Network Schema is the conceptual schema for the network. Logical views correspond to a Global External Schema. The proposed architecture is application oriented and a user has to be associated with a specific application at any particular time. A third level for the Network is introduced to permit the homogenization of the different local models: the Global Conceptual Schema. The Network Schema contains homogenized non-redundant data.

Each Network level is divided into two parts : a data part, and a security part which contains the network authorization rules.

At the Global External Schema Level, security rules are defined by a relation linking object, type of operation permitted, predicate condition, and application category. The Global External Schema security rules are established by a Distributed Data Base Security Administrator. The security rules at the Global External Schema level are defined for each application. To have consistency, the rules for different Global External Schemas should agree, and be a subset of the access rights in the Network Security Schema.

Modifications (grants or revokations) made at local levels are reflected at the Network level after homogenization, consistency verification and redundancy elimination. This is done at the Security Global Conceptual Schema level. This may be bypassed if the network is homogeneous.

The homogenization maps the Local Conceptual Schema to the Secure Global Conceptual Schema. The consistency/redundancy check maps the Secure Global Conceptual Schema into the Secure Network Schema.

For the Secure Global Conceptual Schema a binary model is used. It is claimed by the authors that this model allows the required uniformity (from different models) to be obtained easily. Consistency has then to be checked and two policies that could be applied, are given: taking the most restrictive rule (AND the rules) or using feedback from the different sites involved. The fact that the binary model decomposes the relations in an atomic manner makes it easier to check the consistency and redundancy.

A set of formal rules is given to ensure that the mappings Local Conceptual Schema to Secure Global Conceptual Schema and Secure Global Conceptual Schema to Secure Network Schema are done properly.

One interesting aspect of Bussolati's security management system is that a comprehensive view is given integrating the DDBMS aspects with the security ones. An architectural approach is taken which links the proposed system with the existing ANSI DDBMS model. However, in an attempt to include all aspects of DDBMS, Bussolati and al integrated in their system an homogenization to perform all types of data model mappings. Data model mapping, as we shall see later, is an area which is still under investigation and for which no satisfactory results have yet been obtained. Because the design of the Bussolati system is heavily based on model mappings between the different levels, a considerable overhead is introduced.

The ANSI model used is an application oriented model. In this model user and application become one entity at access time, which may create problems if different rights have to be given to the user and the application he/she belongs to. This scheme does not allow for protection of the user against the application or vice-versa.

The fact that global and local users are distinct does not allow the flexibility of a user using both global and local information at the same time. A user has to belong to only one application at access time, which means that rights from different applications cannot be combined. The application oriented approach implies an inflexible security system (all applications have to be known in advance), but allows for a tighter control.

The control of information flow between sites or between users, which is an important DDBMS protection issue as we shall see later, is not considered in Bussolati's work.

2. KERSTEN'S DECENTRALIZED AUTHORIZATION METHOD

The main problem addressed in (KERST81) is the determination of the subject whose authorization should be checked when an object is accessed through a series of calls of programs owned by different subjects. An example is given where user U owns program P1, the Data Base Administrator (DBA) owns program P2 and the Data Manager (DM) owns program P3, and P1 calls P2 which in turn calls P3. Three approaches are given to determine which authorizations have to be taken into account.

1) P3 knows all the chain of calls and has authorizations associated with U and DBA. The information released is then the intersection of the authorized information for U and DBA. This follows the principle of least privilege.

2) P3 bases its decision only on the authorization stored for U. The problem with this approach is that DBA could obtain information he/she was not permitted to acquire.

3) Only the authorization of the direct caller DBA is checked. It is then up to DBA to check the authorization of U on the information passed. This strategy is the one adopted by the authors. With this approach all the necessary rights for an access are given while in the second strategy P2 may have for example required more rights than U had but could not get them. This problem is refered to as the augmentation of rights problem. Users can grant or revoke rights by extending or reducing the access requirement of the programs they own. A similar problem in operating systems, referred to as "amplification of rights" is presented by Jones (JONES73).

A data base system is defined as "a collection of data objects created, used and maintained by a large collection of tools: primarily application programs". A certain number of principles are given for control decentralization. Each tool has an owner, who is not necessarily its creator, but who is responsible for it. The owner has administrative rights over the tool and can destroy it, lock it and grant or revoke rights to it. The authorization process should be carried out in as a decentralized way as possible.

The authorization scheme is extended to communicating DBMS, where the integration is minimum and the user is not given a unique view of the whole data base. An agent is introduced at each data base who takes care of establishing a secure communication channel between its data base and others.

For DDB where the user is given a single view of the whole data base, the programs are the ones which establish secure communication channels with others. The authentication can be fully replicated, in which case no communication is needed between the different sites, or it can be partially or not replicated, and the sites have to communicate to authenticate users.

This work does not contribute a great deal to the problem of security in DDBMS. Only one very specific problem is considered, without putting it in the context of a particular protection model. However the principles given for decentralized control could be a good starting point for further research.

3. VAN DE RIET MODULE DEFINITION FOR ACCESS CONTROL IN COMMUNICATING DBMSS

The driving idea behind Van de Riet and Kersten's proposed model (RIET80) is that a unified approach to the design of data base systems integrating programming languages and data base management, should be taken. They state that such an approach allows for more secure and reliable systems. To achieve this goal three different "tools" are used:

1) The use of dedicated data base computers linked to other computers. All the security and privacy information is stored in the data base computers where the enforcement is done. Users are connected to "normal computers" which request information from the closest data base machine, which in turn obtain the necessary information from other data base machines.

2) The use of a unified programming environment such as the User Software Engineering (USE) project as a basis for the developement of an integrated data base system.

3) The use of language primitives to define structure, operations, access rights, and semantic integrity constraints of a DBMS, in a uniform way. The concept of abstract data types is used to achieve this goal. The language PLAIN is used for its facilities to define abstract data types, and definition and manipulation of relational data bases.

In the proposed model access rights are not granted for simple operations separately, as is the case in capability systems for example, but rather at the transaction level. Transactions are grouped in modules and access rights to these modules are given to groups of users. Abstract data types are then used to represent the access control. Each user class has a set of abstract data types associated with it giving it access to modules.

The approach taken in Van De Riet model is not a comprehensive one. No model for DDBMS protection is proposed, but rather some principles to improve security of an existing system or to design a new one are given.

4. EXTENSION OF ACCESS CONTROL OF SYSTEM R TO R* In (WILMS81) a generalization of the authorization mechanism of system R to the distributed case is given. One of the basic policies used in system R* is that site autonomy should be preserved. This means that each site controls access to the objects it stores, and also controls the granting and revocation of access rights. The remote sites should not be involved in the removal of local authorizations or the deletion of local objects.

   In R* the authorization tables of system R are extended by recording the network location of each object and each subject. Each site holds an authorization table indicating the rights that local and remote subjects have over local objects.

   When composite objects are defined in terms of already existing objects, the object creator grants access to the components of the composite object. The authorization for the composite object is defined by "one dependency tuple per underlying object node of the underlying object, and a definition tuple at the location node of the composite object." All the grants for the composite object are stored at the node holding it.

Another problem considered in R* is the handling of users groups, when the group members are not all at the same site. Wilms and Lindsay (WILMS81) propose three different solutions to this problem:

1) The group definition is replicated at all the sites holding an object to which the group has been granted an authorization.

2) The second solution is a modified version of the first one. When a member is removed from the group, the majority of nodes have to be interconnected and working. This avoids the problem of inconsistency of rights throughout the network.

3) The authorization is checked by querying the nodes holding the definition of the group when a member of the group requests an access.

Although two main problems specific to the distributed case are addressed in R*, namely the handling of composite objects and group definition, some important issues are not considered, such as the handling of replicated data or the control of information flow between sites.

Protection in DDB is still in its infancy and very few models have been proposed. Some extensions of existing centralized protection models to the distributed case have been given, but they do not usually deal with all of the specific problems arising from the distribution of the data base. Some general principles have also been determined as the integrated system of Van de Riet and Kersten (RIET80).

In this work a number of the problems in DDB protection are examined. We present a DDBMS protection model which takes a comprehensive view integrating DDBMS and protection functions and approaches the protection issues from an architectural point of view. The protection system we propose is powerful enough to handle a large number of policy choices and protection for a centralized DBMS as well. The issues of information flow control and site autonomy are also incorporated in the model.

1.4 DATA AND PROTECTION MODELS.

Before we study the protection problems specific to DDBMS, we need to establish a working background and define the type of DDBMS for which we will investigate the protection issues. We do not make any assumptions with regard to the issues of query processing, data partitioning and replication, and concurrency control. As was mentioned before these issues are incorporated in our model without any assumption on the way they are resolved. However, decisions have to be taken in the areas of data and protection models before designing the protection model for DDBMS. The choices we make for the data and protection models are based on the state of the art in both areas. We will look at each one of the issues and give justifications for our choices.

1. DATA MODEL

To justify our choice for a unique data model for the DDBMS we will briefly survey the literature related to data mapping.

If different data models are supported by the DDBMS, a translation has to be made so that the information re-presented in one model can be expressed in another one. The different data models existing in a DDB have to be mapped either into one another or into a common one. This implies a lot of overhead if a great number of different models exist in the DDB. But most important, this may result in semantics problems which are hard to solve. For example, two different data bases may use for exam-ple, two different names for the same object, or the same name for two different objects. These problems can be worsened when the models are different and the problems become harder to identify and to correct due to the in-herent difference in representation.

Presently mappings in DDB are not done in an automatic fashion. All the mappings methods imply some kind of manual process, where a lot of discretion is left to the human translator. It seems that in the near future, we cannot hope to completely automate the mapping process, if only for the semantics problems involved.

In (BILLE79) Biller describes how a semantic logical data model and a data definition language (LDDL) could be used to specify the translation between different data models. However only an example of translation from a Codasyl schema to a relational one is given. LDDL is used to represent semantics information in terms of constraints of consistency. The method proposed is mainly manual, and LDDL is just a tool for a human translator to better express the consistency constraints.

Adiba et al (ADIBA76) propose a method for mapping a relational model into a network or a hierarchical one and vice versa, using a binary relational model as an intermediary form. However the process is not completely automatic, and requires human intervention.

Pelagatti (PELEG78) describes two methods for mapping external views to a common conceptual one.

1) Operational Mapping Definition

Each operation at the external level is translated in a set of operations at the conceptual level. The advantage of this method is its flexibility in defining new external models. The disadvantage being that translation is needed for each operation, even if a lot of them are similar.

2) Structural Mapping Definition

A general mapping is defined for a class of external models (e.g. one for relational, one for network etc.). The advantage of this method is the possibility of using optimal generalized mappings. However, flexibility is lost and the generalized mappings may be hard to create.

We can conclude from this literature review that the data model mapping problems have not been solved satisfactorily in any model and that the semantics problems require manual intervention. The current state of the art does not suggest that these problems will be solved in the near future. We therefore concluded that it was necessary to reflect the current state of the art to assume that a common data model was used in the DDBMS for which we model the protection.

The next step in our decision process was to decide which data model should be chosen. Codd (SCHMI83) argues that there are four main reasons why the relational approach lends itself to DDBMS:

a. it is easier to partition a relation and distribute the resulting relations than to decompose data bases based on the network or hierarchical models.

b. The relational operators are very powerful and can be efficiently used for recomposing distributed data dynamically.

c. The conciseness of the high level languages of relational systems allows for considerable transmission economy.

d. By using concise high level languages available for relational systems, a node can easily decompose into parts locally processed and parts that will have to be processed at other nodes.

The further study of the main DDBMS prototypes prompted us to chose the relational model as the common data model for the DDBMS for which we study the protection issues. The main protopypes we reviewed are: POLYPHEME (ADIBA82), the DDBMS developed at the Politecnico de Milano (CERI82) SDD1 (ROTHN80), and R* (HAAS82).

Most DDBMS prototypes are homogeneous and use the relational model as the common data model. Therefore, after analyzing the state of the art in the area of data model mapping, considering the reasons presented by Codd for the use of the relational model in DDBMS, and reviewing the different prototypes currently being implemented, we decided to assume that the DDBMS we consider are homogeneous and use the relational model.

2. PROTECTION MODEL

The existing protection models can be categorized in two different ways:

- Open vs. closed systems

- Discretionary vs. nondiscretionary systems.

a. Open vs. closed systems.

In closed systems users are allowed to access data only if they have been explicitly authorized to do so through authorizations. On the other hand, in open systems the users can access any data unless they have been explicitly denied access to it. In the type of applications where users are given access to a large part of the data base, closed systems are not suitable. In this case, open systems offer more flexibility than closed ones, since it is not necessary for the security officers to explicitly list all the data the users are allowed to access. However, closed systems are more secure than open ones. In a closed system the omission or deletion of an access rule although restricting the access to the data, does not jeopardize the security of the system. In an open system the absence of a necessary rule allows for unauthorized accesses to be performed.

Because of the inherent shortcomings of each type of system, we chose to design a new type of system termed hybrid, which combines features from open and closed systems. The hybrid system provides the flexibility of open systems, while keeping the security of closed systems.

b. Discretionary vs. nondiscretionary systems.

Discretionary models assume that there exists a privileged group of users, called authorizers, who can give access rights to other users. Two approaches can be taken for the definition of authorizers. Creators of data can become owners of it and have the right to give access to it. Alternatively a person such as the Data Base Administrator or the Security Administrator may be designed to define all the access rules to data. Typical discretionary models are the Access Matrix model (LAMPS74), Authorizations lists and Capabilities (GRAHA72) Formularies (HOFFM71) and Predicate Based Models (HARTS81) We will review the predicate based models because they are the ones which offer the greater power and flexibility.

PREDICATE BASED MODELS The access matrix model and models directly derived from it have the major drawback of not allowing data dependent or system dependent conditions to be defined in the authorizations.

Predicate based models can be conceptually conceived of as an extension of the access matrix model. The same concept of subjects having rights to objects is used, but is refined by adding predicates to define conditions that must be satisfied before access to the objects is allowed.

Hartson's model (HARTS81) is an example of a predicate based model. Authorizations are defined as being 5-tuples of the form p= (a,U,O,D,c) where a is the name of the authorizer granting the authorization p, U is a user group name, O are the operations users in U are allowed to perform on data D, and c is an access condition defining the condition under which the access is authorized. User groups, data and access conditions can be defined using predicates.

Three main classes of access conditions can be defined:

1) System dependent access conditions where the condition is defined according to the general system state.

2) Query dependent access conditions where the true values of the predicates are obtained directly from the user's query.

3) Data dependent access conditions which value can be determined only after a retrieval from the data base.

The Hartson's predicate based model has been integrated within the the MULTISAFE architecture which we will also briefly review since it inspired our model design.

MULTISAFE is a MULTIprocessor system for supporting Secure Authorization with Full Enforcement developed by Trueblood and Hartson (TRUEB80). The protection mechanism is based on the division of the system into three modules:

1) The User and Application Module (UAM) which is the interface with the users. This module coordinates and analyzes the user queries.

2) The data Storage and Retrieval Module (SRM) which performs the data base operations.

3) The Protection and Security Module (PSM) which performs all the security checks.

Figure 1 shows the structure of this model.

Unlike discretionary models, nondiscretionary ones assume that the data base is divided into cate-

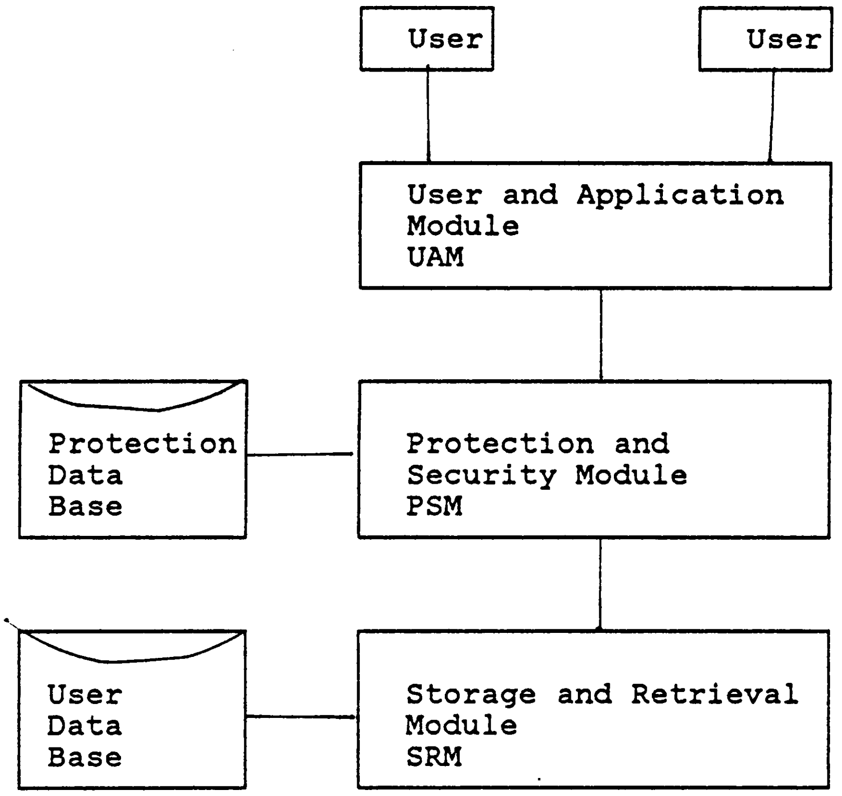

User
User
User and Application Module UAM
Protection Data Base
Protection and Security Module PSM
User Data Base
Storage and Retrieval Module SRM

Figure 1. MULTISAFE Architecture

gories and that users assigned to one category cannot access data in another one.

These models are very often used in the military environment where the security requirements establish a strict hierarchy. Examples of nondiscretionary systems include the Bell and La Padula model (ref) and Denning's Information Flow Model (DENNI82). We will only look at Denning's Information Model because it is the most general and flexible one.

DENNING'S INFORMATION FLOW MODEL In Information Flow models the main concern is the control of dissemination of information. Denning (DENNI82) developed a lattice model for Information Flow Control. In this lattice information flows only into classes at least as inclusive. The information flow is secure if and only if all explicit and implicit flows are secure.

Discretionary models allow for more flexibility in the definition of authorizations than do nondiscretionary models. This flexibility is especially true for the predicate based models which allow for system and data dependent access rules. However the flow of information is not controlled in discretionary models. In multilevel models, where the flow of information is controlled, the rigid hierarchical structure implies that certain rules cannot be represented. Fernandez et al. (FERNA81) give the example of the access rule permitting a user A to access objects O3 and O2, user B to access O2 and O1 and C to access O1 and O3 which cannot be represented in a multilevel model. However, this case can be handled by the lattice model due to the fact that the ordering is only partial in that model, but the creation of new objects implies the complete restructuring of the lattice.

Because of the restrictions of nondiscretionary systems for the type of DDBMS we consider, our choice has to be made in favor of discretionary systems. The most powerful discretionary system is the predicate based model proposed by Hartson. However, since the information flow control issue is not addressed in predicate based models, we adopted an intermediary approach. The hybrid system while possessing most properties of discretionary models, can handle information control. The main differences between the hybrid system and typical predicate based models are that the creator of data does not necessarily keep complete control over it, and that the flow of information between users is controlled in the hybrid system.

1.5 ASSUMPTIONS.

To establish a working background a certain number of assumptions have been made concerning the environment in which the protection problems are to be studied. These assumptions fall in three main categories: context, technical and solved issues.

1. Context.

   The following assumptions are made with regard to the type of environment in which the DDBMS protection issues are studied.

   a. The DDBMS under consideration serves a large and geographically dispersed community composed of individuals, corporations, government agencies, etc.

   b. There are two main types of users of the DDBMS:

      - some users contributing data to the DDB as well as querying it

      - other users only querying the DDB.

2. Technical issues.

Decisions had to be made with regard to some technical issues. These decisions are summarized below.

a. The DDB is partially redundant. This is the most general case.

b. The DDBMS is homogeneous with respect to the data model used. This assumption reflects the state of the art for DDBMS: most current DDBMS are implemented under this assumption (HAAS82, ROTHN80).

c. The common data model is the relational one. Most work done on DDB assume such a model (CERI82, CHAN82, HAAS82). It is the most mathematical model and one where semantics can be best formalized. The concepts developed in this work could be extended to other models. However, the manner in which these concepts are expressed and implemented are specific to the relational model.

d. As with the data model, the protection model is assumed to be the same for all the sites participating in the DDBMS.

e. The common protection model chosen is based on the Hartson predicate model.

3. Solved issues.

To concentrate only on the main protection issues of DDBMS we assume that in the DDBMS we consider, some problems for which solutions have been given in the literature are solved.

a. Each object is uniquely identified in the DDB. This implies in particular that the same name is not used for two different objects and that each object has only one name. We realize that, in practice, this is a demanding assumption. However, without this assumption little, if anything, can be said about protection issues.

b. The sites are assumed to function properly and to enforce the protection rules for local data without failure.

c. The communication lines are reliable, i.e. they are failure safe and they are secure (this could be achieved by encryption for example).

d. User authentication is correctly performed by the Operating System or by some function integrated in the DDBMS. Although authentication is a crucial protection issue, in this work we will concern ourselves only with access control issues. This assumption is rightly justified because the subject of authentication has already been widely studied.

The effect of these assumptions is to create a system which focuses only on the issue of protection in DDBMS without handling the problems of heterogeneity, data model mapping, reliability, recoverability and the alike problems which are covered in other works.

While focusing on the security problems, these assumptions do not diminish the difficulty or interest of the security problem itself.

1.6 DISSERTATION ORGANIZATION.

To establish a frame of reference for the study of protection in DDBMS, we first define a set of basic policies. For any given DDBMS, a subset of policy alternatives are chosen. A good protection model should support a maximum number of policy alternatives subsets.

After identifying the basic differences between centralized DBMS and DDBMS requirements, we present a system termed "hybrid" which satisfies these requirements and allows for variations on the policy choices. By combining features of open and closed as well as discretionary and nondiscretionary systems, the hybrid system provides a great flexibility and control over data derived from the DDB, and information flow between sites as well as among users. The hybrid system is the basis for the modeling of protection in DDBMS we propose. From the set of identified policies, only two have an impact on the modeling. Different models accomodating the various options for these policies are presented using a modular approach. A set of building blocks are defined and for each set of policy choices, the appropriate blocks are chosen. For each model three levels of abstraction are defined, to describe different aspects of the models.

2.0 CHAPTER 2 POLICIES

To illustrate the robust character of the security problem in DDBMS we have identified a number of policies which are related to security issues. These policies can be used to evaluate the power of given protection systems for DDBMS. For different DDBMS different sets of policy choices have to be supported. A good protection system should allow for the implementation of a maximum number of sets of policies, so as to satisfy a large community of authorizers as well as users. The flexibility of the system, as measured by the number of policy choices supported, should make the use of the DDBMS by a wide variety of networks easier.

To establish a framework of discussion for the policies we will first briefly introduce the three main categories of protection systems. The existing protection systems can be divided in two basic groups: open and close systems. In open systems users are allowed to access any data available in the system unless such access is otherwise denied by a constraint. In closed systems access to the data base is permitted only when the user possesses an explicit authorization for the attempted access. In this work we propose a third category of system where both authorizations and constraints are defined. In this hybrid system authorizations are granted to access the data stored in the DDB. Constraints are defined to control the computations a user can perform on the data as well as the flow of information in the network between users or between sites.

2.1 SPECIFICATION OF POLICIES.

For the sake of the policies' specification we assume that both authorizations and constraints are defined. For closed systems the policies for constraints should not be considered as relevant since only authorizations can be defined in such systems. Similarly for open systems the policies defined for authorizations are not to be taken into account since only constraints are defined.

Two major categories of policies are defined. The first category consists of user's rights policies which are related mainly to the enforcement mechanism of the protection system. They define the type of rights users can obtain from the DDBMS. The second category of policies delineate the system characteristics. While the first category of policies mainly concerns the system's users, the second one is of major interest to the system administrators and authorizers.

A. USER'S RIGHTS.

The policies in this category determine what properties the protection system should possess to satisfy the users' requirements. They can be further divided into three subcategories.

1. UNIFORMITY

These policies characterize the manner in which the system responds to a user's query. The policies define if the same answer should be given to the same or an equivalent query regardless of the site of the user, the site of the data, the query decomposition used, or the order in which the queries are processed.

1.1. SITE INDEPENDENCE.

1.1.1. SITE OF THE USER INDEPENDENCE.

This policy states that the response to a DDBMS query is independent of the site of the user. A user presenting the same query from two different sites should obtain the same response. This implies in particular that the protection rules applied to a user are independent of the user's site.

An alternative policy where the query response can be site dependent could also be adopted. Having such a policy may be necessary if authorizations are conditioned by system dependent predicates which can include site dependency. This would be the case if the data base administrator in a company specified that payroll records could be accessed only from the payroll department terminals.

1.1.2. SITE OF THE DATA INDEPENDENCE.

Three alternative policies can be adopted:

1. If data is replicated at different sites, the response to a query does not depend on the site where the data is retrieved. This effect can be achieved for example, by

having all sites or organizations hold the same protection information for replicated data. Alternatively, the most restrictive protection requirements of any site could be used. Information is not released to a user if even one site does not allow it to be accessed by that user.

2. The query response depends on the site from which it is retrieved. An example of a case where this policy is necessary is the following: a network is established between government agencies including the draft registration office. A constraint may be imposed by the draft office expressing that non privileged users cannot obtain information collected for draft processing such as Social Security Number or address of a draftee. However, the draft office may not object to non privileged users obtaining the same information from another source in the network.

3. A variation of the preceding policy choice is to allow the query response to be site dependent but require that the DDBMS always releases the information if it is authorized and available somewhere in the network. The information is to be released even if the query decomposition necessary for the response is not the op-

timal decomposition. This policy corresponds to the "right to know" mentioned in other works (FERNA81).

1.2. DECOMPOSITION INDEPENDENCE.

This policy states that if the DDBMS allows different decompositions of the same query depending on the state of the system (network traffic, congestion, site availability, etc.) then the same response should be given for all decompositions. This implies that equivalent protection rules are applied for all decompositions.

An alternative policy is that the query response is dependent on the decomposition. To each decomposition correspond different costs, accuracy and speed of access, depending on the state of the DDB or the system state such as the the network traffic and congestion. If the accuracy of the response is not critical to the users, the system may determine the most efficient decomposition depending on the efficiency factors most important to the users as well as on constraints on the flow of information between sites. If the organizations holding copies of the same data need to retain some autonomy over the control of the data they hold, decompositions using different copies of the same replicated data may lead to different answers since the control rules may vary from one replication to another. The degree of control the different organizations retain and the control performed over the flow of information between sites are issues that are the subject of the policies of organization trust (policies 4.6).

The same data may be managed differently depending on the organization handling it. The various organizations may decide to give separate names to the replication of the data they hold. This replication may be updated and controlled differently from the other replications. For example, data D may be named D1, D2 and D3 by organizations 1, 2 and 3. However, for a certain user U, a view may be defined expressing that whenever U requests access to D, any of D1, D2 or D3 may be used. This may be the case if U does not require very precise or updated inforrmation. The answer to a query requiring access to D is then decomposition dependent. The answer varies depending on whether D1, D2 or D3 is used.

1.3. SEQUENCE OF QUERIES INDEPENDENCE.

It may be desirable that the order in which the queries are presented to the DDBMS influence the responses. A history of the queries can be kept to determine if the information a user has already obtained from the DDBMS can be combined outside of the system with new data from the DDBMS to derive unauthorized information (COHEN77)

The other choice for this policy is one where it is required that the DDBMS respond to a query independently of the queries previously submitted. In this case the protection system assures that the user cannot obtain at any point in time data which combined with answers to future potential queries violate the security rules.

2. ACCESSIBILITY.

The accessibility policies define the conditions under which users are allowed to maintain their rights and their privacy, as well as to be knowledgeable of all their rights and constraints.

2.1. PRESERVATION.

2.1.1. RIGHTS PRESERVATION WITH NEW AUTHORIZATIONS.

When this policy is chosen, a user does not lose rights when granted new authorizations. One possible way of implementing this policy is by requiring the user to agree on any new granted right, and the DDBMS to inform the user of all implications the granting of a new right has on previously acquired rights.

An alternative policy is not to give users the assurance that old rights can be maintained if new ones are acquired. When this policy is adopted users may lose old rights when granted new ones. They may also obtain rights which they did not explicitly request, by belonging for example to a group labeled general for which implicit rights are given. This policy might be chosen for efficiency reasons, to avoid for example handling rights granting on a user's basis rather than on a group basis. Users do not exercise their rights as individuals but rather as members of groups. In this case the system does not have to notify users of all implications a new authorization has on rights previously acquired. Rights are granted to whole groups without having to analyze the consequences for the different members of the group.

2.1.2. RIGHTS PRESERVATION WITH NEW GROUP MEMBERSHIP.

This policy states that a user does not lose rights previously acquired when joining a new group.

The other policy choice corresponds to users agreeing to have their rights modified when joining a new group which rights are not fully compatible with the ones the user already holds.

2.2. DISCLOSURE.

When the disclosure policy is in effect, at any time users can obtain a list of their rights, i.e., all relevant authorizations and constraints.

However, for economical reasons or to increase security users may not be allowed to obtain a list of the constraints imposed them.

2.3. PRIVACY.

This policy defines the right to privacy a user may possess.

To obtain new rights from an authorizer a user does not have to release information about rights obtained from other authorizers.

It may be necessary to adopt the alternative policy where authorizers are allowed to access the list of rights of users to whom they grant new rights. This alternative gives authorizers more control over the rights a user may derive from the combination of rights obtained from other authorizers and the newly acquired rights.

3. USER INDEPENDENCE.

3. COLLECTIBILITY OF RIGHTS

Users have the power of rights collectibility if any user belonging to various groups can simultaneously exercise the rights associated with the different groups.

A group dependency policy could be chosen and users allowed to exercise rights of only one group at a time. Under this policy it is easier to audit a user's rights and to detect conflicting rights.

B. SYSTEM CHARACTERISTICS

The second category of policies define the characteristics the protection system should have in order to comply with the requirements of DDBMS authorizers and administrators.

4. SYSTEM OPERATION

These policies describe the conditions under which the protection system is to operate.

4.1. ACCOUNTABILITY.

If the accountability policy is implemented, at any point in time it is possible to determine who can impose constraints and grant authorizations on specific data.

Maintaining authorizers and constraint imposers lists may create a serious overhead for the protection system. If the authorizers are very independent and no central authority of any form exists in the system these lists may not be made available so that the performance of the system is not affected.

4.2. SUPREMACY.

This policy defines the relationship between authorizations and constraints.

Constraints have priority in all cases. An alternative policy is that constraints overide authorizations only when the constraint imposer has some right over the affected data.

4.3. REVERSIBILITY.

The reversibility policy determines the effect the changes of the users' rights have on the state of the protection system.

After revocation of an authorization or annulment of a constraint, the state of the protection system is the same as if that authorization or constraint had never been introduced (GRIFF76).

The set of constraints and authorizations given at a certain point in the lifetime of the DDBMS may be considered as being an important part of the history of the system which has to be preserved. The behavior of the DDBMS is then conditioned on its history. Suppose that an authorization AUT1 is granted to a user U for data D1 at time t1, and revoked at time t2. Suppose further that at time t3 an authorization AUT2 is granted to U for data D2 which if combined with D1 may lead to unauthorized information. When granting AUT2 the authorizer should therefore be aware of the fact that U obtained access in the past to D1. In this case the state of the protection system after revocation of an authorization or annulment of a constraint is different from the state in which the system would have been had the authorization or constraint never been introduced.

Another reason to chose this policy option is for efficiency reasons. When this policy option is chosen, revocation of authorizations and annulment of constraints are usually handled more easily. It is not neccesary in such a case to maintain a full history of the system states, since after the revocation of an authorization or the annulment of a constraint, the previous system state is not completely restored.

4.4. MINIMUM POWER.

The constraints that can be imposed is limited by a predefined level of minimum abilities any user should possess. Users are therefore protected from constraint imposers who could deny them access to most of the DDB, and consequently deny their basic rights as a network user.

The other choice of policy allows the authorizers and constraint imposers to have a complete control over the users activities by not restricting the actions of the protection officers in any manner.

4.5. SCOPE.

This policy defines the scope of effect for a constraint.

A constraint cannot affect any data not explicitly expressed in the constraint.

Alternatively the enforcement of the constraints may be performed by allowing the system to interpret a given constraint and make modifications to users rights not explicitly formulated in the constraint but which may reflect the most efficient way to enforce a given constraint.

4.6. ORGANIZATION TRUST.

The degree to which the different organizations participating in the DDBMS trust each other is reflected by the following policies.

4.6.1. LOCAL AUTONOMY.

Different degrees of local autonomy can be defined depending on the level of supremacy each organization contributing data to the network keeps over its own data.

1. Total autonomy.

   If organizations retain total autonomy over their data, queries submitted by users local to that organization are not controlled by any global mechanism. The organizations have complete control over their data as well as over the rules controlling access to their data. It is also the responsibility of the organizations to enforce the rules pertaining to their data, whether to local or non local users. There is however an exception to this supremacy. Global rules still need to be defined for data that is controlled by more than one organization.

2. Partial autonomy.

   If the organizations retain control over location of the rules governing their data as well as over the enforcement of these rules, but not over the location of the data itself, the type of autonomy the organizations have is labeled partial. This may be the case if the organizations accept to have their data relocated in the network for efficiency reasons. However the organizations need the assurance that they can retain control over the definition and enforcement of the rules over their data.

3. No autonomy.

   The organizations contributing data to the network may decide to rely entirely on the network for assigning the data to the sites as well as for enforcing the rules pertaining to their data. In this case the organizations totally lose their autonomy and there is only one set of rules defined on the data, and these rules are globally enforced.

4.6.2. FLOW CONTROL.

Certain sites in the network may be mistrusted by others, and the flow of information to the mistrusted sites has to be regulated.

Alternatively all sites may be trusted and the information can flow freely between sites freely as long as it is not released to non authorized users.

2.2 FORMALIZATION OF POLICIES

Using the following definitions of the authorizations, constraints queries and subqueries the user's rights policies can now be formalized.

DEFINITIONS

AUTHORIZATION	(A,O,U,D,S) indicates that authorizer A grants user U the right to perform operation O on data D under the conditions specified in the predicate S.
OPERATION CONSTRAINT	(A,O,U,D,S) indicates that constraint imposer A denies user U the right to perform operation O on data D under the conditions specified in the predicate S.

QUERY	(U,O,D,SU) indicates that user U located at site SU requests to perform operation O on data D.
SUBQUERY.	A query is decomposed into subqueries of the form (U,O,D,SU,SA,SR), where SU is the site of the user, SA the site of access, and SR the routing site. The response obtained from a subquery is routed to site SR, and it is then considered as being data stored at site SR. A query Q(i) is decomposed into Q(i)=SubQ(i,1)oSubQ(i,2)o.....SubQ(i,n).

We now present the formalization of the users' rights policies. For each policy, we only formalize the first policy alternative. The second alternative can be easily derived from the first one, since it is the negation of the first one.

1. UNIFORMITY.

1.1. SITE INDEPENDENCE.

1.1.1. SITE OF THE USER INDEPENDENCE.

The fact that a query should be independent of the site where it is submitted is formalized in the following manner:

For any Q(i) and Q(j) such that:

Q(i)=(Ui,Oi,Di,SUi)

Q(j)=(Uj,Oj,Dj,SUj)

where Ui=Uj, Oi=Oj, Di=Dj, and SUi different from SUj then R(Q(i))=R(Q(j)) where R(Q(i)) is the response to Q(i).

1.1.2. SITE OF THE DATA INDEPENDENCE.

Similarly, the fact that the same answer is given to a query whether the data is retrieved from one site or another is expressed as follows:

For any SubQ(i,j) and SubQ(k,l) such that:

SubQ(i,j)=(Ui,Oj,Dj,SUj,SAj,SRj)

SubQ(k,l)=(Uk,Ol,Dl,SUl,SAl,SRl)

where Ui=Uk, Oj=Ol,Dj=Dl and SAj different from SAl

then R(SubQ(i,j))=R(SubQ(k,l)).

1.2. DECOMPOSITION INDEPENDENCE.

If there exists two decompositions of Q(i)

Q(i)=SubQ(1,1)oSubQ(1,2)...oSubQ(1,n)

Q(i)=SubQ(k,1)oSubQ(k,2)o...oSubQ(k,m)

and

R1=R(SubQ(1,1))o...oR(SubQ(1,n)

Rk=R(SubQ(k,1))o...oR(SubQ(k,m)).

then R1=Rk.

1.3. ORDER OF QUERIES INDEPENDENCE.

The formalization of the order of queries independence policy using the query definition and introducing a time factor.

Let Q(i) and Q(j) be any two queries submitted by the same user U.

Q(i)=(U, Oi, Di,SUi)

Q(j)=(U, Oj, Dj,SUj)

Let:

R1(Q(i)) be the response to Q(i) if submitted at time t1

R2(Q(i)) be the response to Q(i) if submitted at time t2

R1(Q(j)) be the response to Q(j) if submitted at time t1

R2(Q(j)) be the response to Q(j) if submitted at time t2.

then:

R1(Q(i))=R2(Q(i)) and R1(Q(j))=R2(Q(j)).

2. ACCESSIBILITY.

2.1. PRESERVATION.

2.1.1. RIGHTS PRESERVATION WITH NEW AUTHORIZATIONS.

This policy translates into:

Let AUT1= {A1,....,An} be a set of authorizations for user U, and AUT2=(AUT1)U(AUTn+1) where AUTn+1 is another authorization for U. If DAUT1 and DAUT2 denote respectively the set of data accessible by U under authorizations AUT1 and AUT2 then DAUT1 is a subset of DAUT2.

2.1.2. RIGHTS PRESERVATION WITH NEW GROUP MEMBERSHIP.

Similarly this policy is formalized in the following manner:

For any i, let Gi represent a group of users and Gl=GloG2o...Gn be the composition of groups G1,..,Gn.
and DAUTl be the set of data accessible by a user belonging to Gl
If U is in Gl and joins Gn+1
and we define Gk=GloGn+1
then DAUTl is a subset of DAUTk
if DAUTk is the set of data accessible by a user belonging to Gk.

2.2. DISCLOSURE.

By defining authorizations and constraints sets we can formalize the disclosure policy.

U has access to
AUT(U)={AUT1,AUT2,....,AUTN}= Authorizations for U.
and CONT(U)={CONT1,CONT2,...CONTM}= Constraints for U.
where AUTi is in AUT(U) iff Ui=U
and CONTj is in CONT(U) iff Uj=U.

3. USER INDEPENDENCE.

3.1. COLLECTIBILITY OF RIGHTS.

The collectibility of rights policy is formalized by:

If U is in G1, G2, ...,Gn

then DAUT(U)=(DAUT1)U(DAUT2)U...(DAUTn).

2.3 POLICIES INTERACTION ANALYSIS

The choice of security policies for a given system has to be made by selecting a set of compatible policy alternatives. In this section we will analyze the interaction between the different policies. For each policy we will study the effect every other policy has on it. This study will then be summarized in a table of policies interaction. To analyze the policies interaction we use the numbering scheme presented in the preceding section to identify the policies.

1.1. SITE INDEPENDENCE.

Site independence is affected only by collectibility of rights, policy 3.1, and local autonomy, policy 4.6.

If the query response is site dependent then it may not be meaningful for users to collect rights from different groups, policy 3.1., if the rights are also site dependent and cannot therefore be exercised simultaneously.

If the response to a query is site independent it implies that the same rules apply to all replications of a set of data. This policy conflicts with the policy of local autonomy, policy 4.6., since with site independence sites are not allowed to maintain their own rules for the replicated data.

1.2.DECOMPOSITION INDEPENDENCE.

To implement the decomposition independence policy it is necessary to have policy 1.1.2. site of data independence in effect. The sets of subqueries for the same query given by different decompositions may retrieve the same data from distinct sites. For the responses to be identical for all the sets of subqueries, it is necessary that the same subresponses be given when the same data is retrieved from different sites.

Since the decomposition independence requires the site of the data independence, it is also incompatible with policy 4.6., local autonomy.

1.3. SEQUENCE OF QUERIES INDEPENDENCE.

When the responses are independent of the order of the queries, the rights preservation in policy 2.1. may be difficult to implement because the rights of a user are dependent on the information the user has already obtained from the system.

2.1. PRESERVATION.

As we just saw with the previous policy, policies 2.1. and 1.3. (sequence of queries independence) may be difficult to implement simultaneously.

To implement rights preservation with new group membership, policy 2.1.2. it is necessary to have rights preservation with new authorizations, policy 2.1.1.

If users are allowed to preserve previously obtained rights when acquiring new ones, authorizers may require users to disclose their rights to them before granting any new ones, which corresponds to policy 2.2. In this way the authorizers can decide if they consider the accumulation of users' old rights with new ones to be safe.

Rights preservation with new group membership, policy 2.1.2., is not meaningful if collectibility of rights, policy 3.1. is not in effect. If users can collect rights and do not lose previously acquired rights when being granted new ones, it is then impossible to implement policy 4.2. which states that constraints overide authorizations in all cases. Lets assume that a constraint is imposed on the combination of data D1 and D2 and that a user U has access to D1. If policy 4.2. is applicable and U is given access to D2, because of the constraint imposed on the combination of D1 and D2, U may lose the unlimited access previously granted.

If policy 2.1. is effective, it is always easier after the revocation of an authorization or the annulment of a constraint to return the system to its state before the introduction of the authorization or the constraint, i.e. policy 4.3. can easily be supported.

If the users rights are preserved, the number of constraints imposed on any given user are better controlled, and policy 4.4. is more easily implemented.

2.2. DISCLOSURE.

As was previously described policies 2.1. and 2.2. may be difficult to implement simultaneously.

When supremacy of the constraints, policy 4.2., is effective it can be implemented using a history keeping scheme where a user's rights are conditioned on the information the user has previously obtained from the system. In this case the list of rights a user can obtain from the system at any time does not reflect the actual rights of the user since the effective rights of the user are dependent on the user's queries and the order in which they are submitted to the DDBMS.

When the organizations cooperating in the network retain certain degrees of autonomy, that is if the choice for policy 4.6.1. is for complete or partial autonomy, producing a list of rights for any user requesting it may imply a serious overhead for the DDBMS.

2.3. PRIVACY.

The privacy policy may be in conflict with the collectibility of rights, policy 3.1., if authorizers require the users to disclose their rights before allowing them to collect their rights they acquire with different group membership.

3. COLLECTIBILITY OF RIGHTS.

When analyzing the rights preservation with new group membership, collectibility of rights, policy 3.1, rights preservation with new group membership, policy 2.1.2 and supremacy, policy 4.2, are incompatible.

Site independence, policy 1.1 and collectibility of rights, policy 3.1. cannot be implemented simultaneously as was shown when the site independence was studied.

As we discovered with the privacy policy, policies 2.3 and 3.1. may be difficult to implement simultaneously.

If a user can belong to groups located at different sites and the sites retain some form of autonomy (policy 4.6) it may be hard to implement the collectibility of rights policy.

4.1. ACCOUNTABILITY.

If the organizations retain some autonomy (policy 4.6) creating lists of authorizers and constrainers may incur a severe overhead.

4.2. SUPREMACY.

As mentioned before policies 4.2., 3.1. and 2.1. are incompatible. Policies 2.3 (privacy) and 4.2 may also be in conflict.

Policy 4.2. and policy 4.4. ,(minimum power) may be difficult to implement simulataneously since the number of constraints that can be imposed on data is reduced if the users are to keep a minimum access power.

4.3. REVERSIBILITY.

As previously seen policy 2.1., (preservation) facilitates the implementation of policy 4.3.

4.4.MINIMUM POWER.

In the same way as for policy 4.3., if policy 2.1. is in effect policy 4.4. is more easily implemented.

4.5. SCOPE.

This policy is not in conflict with any other policy, and is not affected by nor could affect any other policy. This suggests that this policy is not a fundamental one and that choices for this policy can be taken independently of any other policy.

4.6. ORGANIZATION TRUST.

As was described earlier policy 4.6. is in conflict with policies 1.1., (site independence), and 1.2., (decomposition independence).

A conflict may also arise if policy 3.1 (collectibility of rights) is implemented.

Performance of the system is reduced if either policy 4.1 (accountability) or 2.2 (disclosure) are implemented.

We will now summarize the interaction of the different policies in a conflict analysis table. For the purpose of the table the policies are represented by their affirmative option. For example when the policy of site of the user independence is listed in the table, the policy choice considered is is the choice where the reponses are site of the user dependent. The policies are represented by their number. For example, the decomposition independence is represented by the number "1.2.". The following notation is used:
The entries of the table are letters. For all the following definitions, we assume that the defined letter is entered at

the intersection of row i, representing policy k.l., and column j, representing policy m.n.

C	A C reflects that policies k.l. and m.n. are in conflict and cannot be implemented simultaneously.
CP.Q	Cp.q reflects that policies k.l., m.n., and p.q. cannot be implemented together. It is the combination of the three policies which create a conflict.
N	N is used to represent that policy k.l. cannot be implemented without implementing policy m.n.
R	R is used to express that to implement policy m.n., policy k.l. is required to be effective. R is the notation used to express the symmetric of N. Whenever an N appears in position (i,j) in the matrix an R appears in position (j,i). N is used for the requiree policy, while R is used for the required policy.
MC	MC is used to reflect that the two policies m.n. and k.l. may be in conflict under certain conditions. An example of such a type of conflict appears when users cannot lose previously acquired rights (policy 2.1.). Authorizers may require us-

	ers to disclose their rights in such a case (policy 2.2.).
H	H is used to express that policy m.n facilitates the implementation of policy k.l.
IH	IH is used to express that policy k.l facilitates the implementation of policy m.n. IH is the symmetric of H. If an H appears in position (i,j) in the matrix then an IH is present in position (j,i).
P	P is used when the simultaneous implementation of the two policies lower significantly the performance of the system.

The corresponding interaction analysis table follows.

Table 1- Policy Conflict Analysis

	1.1	1.2	1.3	2.1	2.2	2.3	3.1	4.1	4.2	4.3	4.4	4.5	4.6
1.1		R					C						C
1.2	N												C
1.3				MC									
2.1			MC		MC		C4.2		C3.1	H	H		
2.2				MC					MC				P
2.3							MC						
3.1	C			C4.2		MC			C2.1				MC
4.1													
4.2				C3.1	MC		C2.1				MC		
4.3				IH									
4.4				IH					MC				
4.5													
4.6	C					P	MC	P				P	

3.0 CHAPTER 3 HYBRID SYSTEM

3.1 INTRODUCTION.

To characterize the protection issues specific to DDBMS, we will in this chapter draw the differences between centralized and distributed systems. After translating these differences into basic requirements, we will present a system satisfying these requirements.

The list of differences between centralized and distributed systems includes: size, authority, ownership, cost, autonomy, and revocation.

These differences are translated into three basic protection requirements. Authorizers in a DDBMS need to have control over the data the user can derive from the DDB, the flow of information between users and the flow of information between sites during query processing.

To express the control over data derived from the DDB, we define a serial computation as being a series of JOIN, PROJECTION and RESTRICTION operations. Authorizers should be able to regulate the access to data not yet existing in the DDB but which the user can obtain through serial computations. Authorizers in DDBMS do not have a full list of a user's abilities. It is therefore necessary to provide a control of the accesses to computed data a user can give to others. During query processing the query is decomposed in subqueries which are processed at individual sites, but may require data computed at other sites. This transfer of information has to be controlled.

The system proposed in this chapter meets the identified requirements. In this system authorizations are defined as permission to access data in a particular manner. Constraints are used to define the abilities denied to a user.

In addition to satisfying the stated requirements the hybrid system allows for variations on the policies. The system is termed hybrid because it combines features of both open and closed protection systems. In open systems users are allowed to access any information unless otherwise specified by constraints. In closed systems users can access data only if they have been explicitly authorized to do so through authorizations. The proposed system combines both authorizations and constraints. Authorizations are defined on basic access operations such as READ, WRITE, UPDATE and DELETE as well as on single JOIN operations, and constraints are defined on serial computations. Authorizations are therefore defined on accesses to data stored in the DDB, while constraints are usually defined on data derived from the data stored in the DDB. The users are allowed to perform any relational operations such as JOINS, PROJECTIONS and RESTRICTIONS as long as these operations do not violate the constraints. In a closed system the authorizer may either list all the serial computations a user is allowed to perform or the authorizer may allow any serial computation using the authorized data. In the first case a serious overhead may be incurred while in the latter case the authorizer looses control over the data the user can derive from the authorized data. In open systems a symmetric choice of solutions exists. In the hybrid system the authorizer does not need to list all the serial computations a user is allowed to perform, which may be a lengthy process, but rather list the serial computations which the user is not allowed to perform. Similarly, if the user is given access to a very restricted set of the DDB it may be more efficient to list only the small set of authorized access rights. In the hybrid system, authorizations and constraints are two complementary types of control. It is always possible to chose the type of control which gives the shortest list of explicit access rights or denials. In addition, in the hybrid system a special type of constraints termed computational constraints can be defined to avoid a user obtaining two specific domains together in a relation derived through a serial computation. It is much more cumbersome to obtain the same effect in a system where the access control is performed solely with authorizations. In such a system the authorizer would need to determine first the list of serial computations which could be used to obtain the two domains. The authorizer would then need to define authorizations in such a way that these serial computations are not feasible. This process can be very difficult to complete and the authorizer may not be willing to incur the corresponding overhead.

Therefore, the hybrid approach allows for an increased flexibility over purely open systems while keeping the tight control of closed systems.

Furthermore, the proposed system is said to be integrated because authorizations and constraints can be defined on any of the data bases supported by the system including the data bases containing the authorizations, and the constraints themselves. In addition to handling access control the hybrid system can be used to control the flow of information between users as well as the flow of information between sites.

The format of the authorizations is based on Hartson's model (HART81) where the data on which the authorization is defined is the actual data stored in any of the data bases maintained by the system such as the main user data base, the authorizations, constraints, and the directory. Three types of constraints are defined. Operation constraints are used to control the information a user can compute from the DDB. Flow control constraints regulate the flow of information between users. Routing constraints allow authorizers to forbid routing or storage of specific data through certain sites during query processing.

One type of operation constraints: computational constraints, contributes largely to the flexibility of the hybrid system. A computational constraint is used to express that two domains appearing in any number of relations can never be obtained together by any series of queries a user is allowed to perform. However, it is possible for a user to possess authorizations which are in conflict with a computational constraint. Some algorithms to enforce the computational constraints have been identified. The first algorithm is a static one where the enforcement of a computational constraint translates in the modification of the authorizations before any user serial computation. For each user a graph is built. The relations and domains a user has access to are the nodes of the graph and there is an edge between a domain D and a relation R if the user has access to domain D in relation R. When a computational constraint expressing that the user cannot relate two domains is introduced, all the paths linking these two domains in the graph are cut and the corresponding authorizations are modified to reflect these cuts. An advantage of this algorithm is that it is possible to list all the data a user has access to and this list is independent of the order in which the queries are submitted to the DDBMS. Another advantage of this method is that the enforcement algorithm is run only once at the time of the introduction of the constraint. However, it is very difficult to predict which authorizations will be modified when a new constraint is introduced.

The second algorithm presented is dynamic. In this one the computational constraints are enforced at execution time. Whenever a computational constraint is introduced, the authorizations for data named in the constraint are tagged. The tag in the authorizations expresses that either domain of the constraint can be used in a serial computation as long as the serial computation does not involve the other one. The authorizations for the computed data are also tagged which allows for the tag to be propagated along the serial computations. The advantage of this algorithm is that the query is rejected only if it explicitely uses the domains defined in the constraint and is therefore more predictable than the preceding one. Another dynamic algorithm mentioned is Cohen's one. A history of user accesses is maintained and a graph representing the information the user has already obtained is built. Whenever the user submits a query for data which combined with the previously acquired data violates the constraints, the request is rejected. The enforcement is done at query time which classifies this algorithm as a dynamic one. While the use of one algorithm or the other depends on the tradeoffs that have to be made in the particular DDBMS used, we believe that the second algorithm presented is more efficient than Cohen's one. While both allow for a history keeping the tagging algorithm requires less information to be kept in the system.

3.2 DIFFERENCES BETWEEN CENTRALIZED AND DISTRIBUTED DATABASE SYSTEMS.

We will now describe in more detail the main differences between DDBMS and centralized DBMS. The distinctions drawn below are general characterizations of centralized versus distributed DBMS. A few very large organizations may already be experiencing these problems with centralized systems. Without denying that these problems can occur in centralized systems, the frequency and severity of these problems, nonetheless, illustrate the difference in security requirements between centralized and distributed systems.

- Size.

  The relatively small size of a centralized data base allows authorizers to have a better view of the content of the data base than is possible in a distributed system. This gives them better control of the data base, since it is not too difficult to determine which information a user can obtain from the centralized data base. However, in the distributed case the size of the data base is such that an authorizer may not be able to realistically determine what a user will be able to obtain from the data base.

  The size problem is also affected in the considerably increased number and types of users in DDBMS. These larger numbers make it more difficult to distribute authorizations for data access and to anticipate the needs of potential data users.

- Authority.

  In centralized systems, it is usually possible to have a central administration authority having control over the whole data base and who is aware of the abilities of the users. In DDBMS, this becomes much more difficult because of the number of organizations involved and the volume of the data manipulated. The tendency in DDBMS is toward having less and less centralized control. The different organizations involved in the DDBMS share the control and it is much more difficult to identify a central authority.

- Ownership.

  The concept of ownership is somewhat different in centralized and distributed systems. In centralized systems, in the last instance the data belongs to the organization as a whole. Since users are members of this organization, the information remains inside the company. In the distributed case, where many organizations are involved the concept of ownership has to be more strict, since information could easily pass from one organization to another.

- Cost of data.

  Data has a monetary value: cost to collect it, organize it etc. It may not be too critical to have valuable data divulged within an organization since the price is paid by the company and the users are part of this organization. It becomes much more important to control the transfer of costly data between organizations. An organization may for example allow another to use some valuable data provided it does not make it available publicly.

- Autonomy.

  In distributed systems, data stored at more than one site has to be protected at a system-wide level. In system R* (HAAS82) the driving policy is that the protection enforcement is made at the site of the data. This policy cannot be enforced when data is replicated or partitioned in the system. Protection rules may have to be defined involving data stored at different sites, as well as transfer of data between sites. To enforce these rules a global level can be defined, which controls the access and flow of information associated with more than one site. In centralized systems, an organization can trust its own security mechanism, while in a distributed one it needs to depend also on some kind of global mechanism. The level of local autonomy required by the different organizations contributing to the DDBMS will depend on the degree to which the global component can be trusted. The more secure the control at the global level, the higher the trust of the local organizations will be.

- Revocation.

  Within an organization, users and authorizers usually have close working relationships, either hierarchical or as colleagues. In such an environment when an authorization is revoked the authorizer is usually aware of the impact this act has upon the user of the data. But in DDBMS the user and the authorizer are more often unknown to each other, and the authorizer may not be aware of the activities of the users. Users who utilize the data obtained from the DDB for projects where they have an important investment may need the assurance that their authorizations will not be revoked at any time.

3.3 DDB REQUIREMENTS.

The differences between centralized and distributed systems outlined in the preceding section suggest that DDBMS require stronger protection mechanisms than the ones developed for centralized DBMS's. Authorizers need a system capable of aiding them to determine the information a user can obtain from the data base, as well as a mechanism to avoid users deriving unauthorized data from the DDB. The authorizers in a distributed system need to have increased control over:

1. The data the user can derive from the DDB.
2. The flow of information between users.
3. The flow of information between sites.

3.3.1 CONTROL OF DERIVED DATA.

Data derived from the DDB may be viewed as follows.

Let:

R	be the set of all relations stored in the DDB
R(A)	be the set of data authorizer A has control over.
D(R(A))	be the set of data that can be derived from R(A) through data base operations.
R(U)	be the set of data user U has access to.
D(R(U))	be the set of data that can be derived from R(U)

These sets are graphically represented in figure 2.

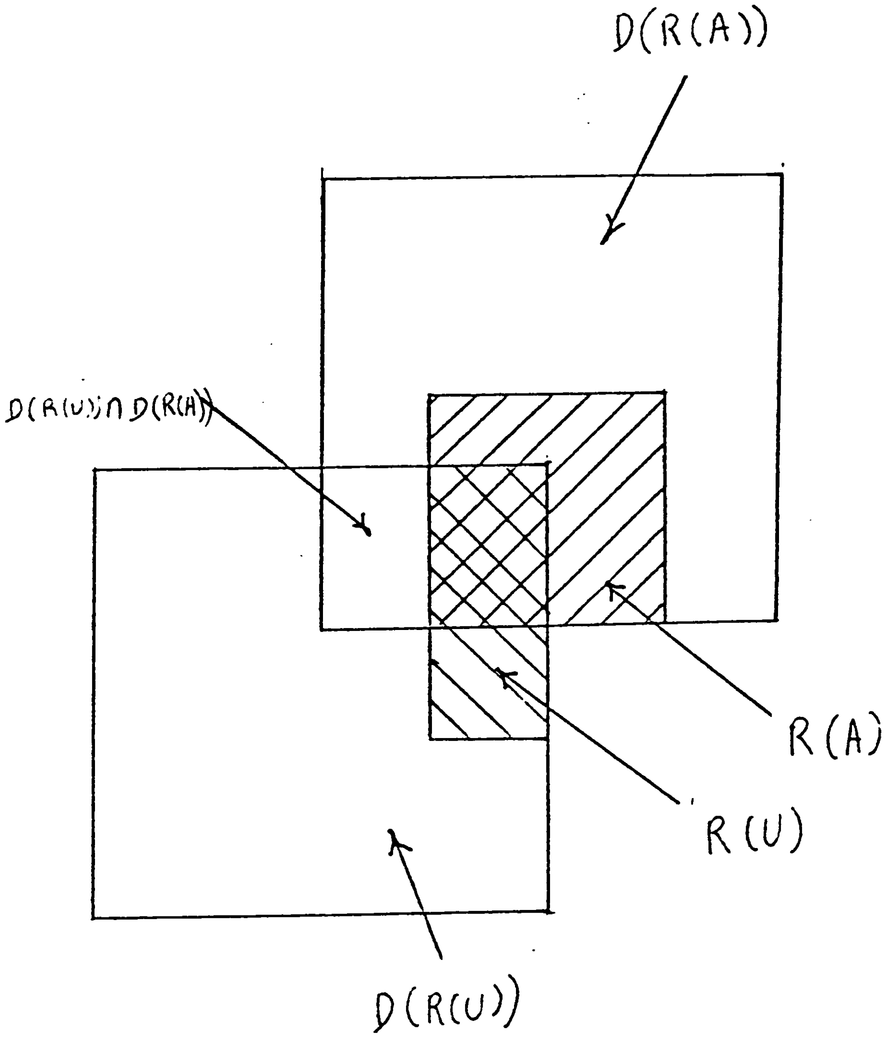


Figure 2. Control of Derived Data

cess needs to be regulated. Authorizer A needs to keep control over the information users derive from the data R(A) so as to monitor the misuse of data obtained from the system. The control of D(R(A)) may be particularly crucial if some data R_1 included in R(A) is controlled by multiple authorizers. Authorizer A may require that users do not derive specific information using R_1 independently of the rights they obtain from other authorizers.

3.3.2 CONTROL OF INFORMATION FLOW BETWEEN USERS.

A network user U1 having an authorized access to some data D could use it in some serial computation and give access to the result of the serial computation to another user U2 not having any access to the original data D. The authorizer of D should be able to control such a flow of information to user U2. This is particularly important in DDBMS where the authorizer may not have complete information on the capabilities of each user. This is a typical problem of information flow.

3.3.3 CONTROL OF INFORMATION FLOW BETWEEN SITES.

In a DDBMS queries issued at one site have to be decomposed into subqueries. Each subquery is then processable by individual sites. To process a subquery a site may require information computed at another site, and that information has to be transfered. Therefore, there is a flow of information between sites at query processing time. However, an authorizer at one site may not trust the protection mechanism of another site well enough to allow sensitive data to transit through that site. The authorizer should therefore, be allowed to control the flow of information between sites.

Current protection systems do not address fully these issues. It is the purpose of this chapter to illustrate a means of addressing these security issues in an integrated manner.

3.4 OPEN AND CLOSED SYSTEMS SOLUTIONS.

We will first look at how the typical protection systems can fulfill the requirements previously defined.

Protection systems can be divided in two major categories: open and closed systems. In open systems users are allowed to access any information unless otherwise prohibited by constraining rules. In closed systems users can access data only if they have been specifically authorized to do so.

In closed systems the authorizer has complete control over R(A). There are two ways in which an authorization mechanism can be implemented.

1. Authorizations are given over R(A) and no serial computation can be realized unless specifically authorized.

With this implementation the authorizer has complete control over the intersection of D(R(A)) and D(R(U)). However, users have to explicitly ask for rights for any data base operation they require which may translate into a considerable overhead for both user and authorizer.

2. Authorizations can be defined over R(A) but any serial computation using R(A) is allowed. In this case, the expense to the authorizer and user is lowered but the authorizer does not have any control over the intersection of D(R(A)) and D(R(U)).

In open systems a symmetric set of choices arises.

The problem of information flow between users has been addressed in flow control mechanisms. However, these mechanisms are usually based on a hierarchical class organization which does not allow discretionary access rules to be easily represented. In the lattice model proposed by Denning (DENNI82) the class lattice may have to be reorganized each time a new object is introduced in the data base. The rigid structure of the lattice makes it difficult to use predicates to qualify the system conditions under which accesses to the data base have to be granted.

Flow control mechanisms are therefore not well suited for access control. In the hybrid system presented in this chapter both access control and information flow control can be realized.

To our knowledge, based on the literature we reviewed, no system solving the problem of information flow between sites has been proposed.

3.5 HYBRID SYSTEM SOLUTION.

The system that we propose would satisfy the requirements identified in the preceding section. This system is a hybrid one because both authorizations and constraints can be defined. Authorizations are, in the usual sense, permission to access data in a particular manner. Constraints, on the other hand, describe abilities which are explicitly denied to a given user. Authorizations can be defined on data stored in the DDB and on simple JOIN operations between two relations as well, and constraints can be placed on computed data. Constraints can also be defined to control the flow of information between users and the flow of information between sites during query decomposition.

Authorizations define the data accesses a user is allowed to perform, and under which conditions these accesses are possible. On the other hand, constraints define those accesses not permitted to a user. With this combination of tools a greater flexibility than in closed systems is achieved, while a tighter control than the one available in open systems is obtained.

One illustration of the flexibility of the hybrid model is the following. Suppose that an authorizer grants to a certain group of users a set of authorizations allowing some stated access to the data which the authorizer owns and which is stored in the DDB. Also suppose that from this set of data the users can derive new information which the authorizer wishes to control. The authorizer can control this derived data by defining constraints. In this manner authorizers do not have to explicitly grant access to all the different serial computations users are allowed to make. In many cases, the succinct form of a constraint makes it much easier to express the set of data that users should not be allowed to derive than to write authorizations for all the ones they are allowed to compute.

To compare open, closed and hybrid systems, it is useful to analyze the capabilities each system has to handle different types of protection needs. The authorizations given on data may be intended to give users different abilities. Depending on the extent of the abilities given to users three types of authorizations can be defined.

1. Limited ability rights.

   In this case authorizations are limited to specific views and no serial computations are allowed using these views. This may be the case in a fairly static environment where users require access to the same information most of the time and seldom perform new serial computations. An example of such an environment is a library network serving a community of researchers which consult the bibliographic data for their specific area of interest.

Closed systems are designed to handle this type of authorizations, and therefore perform very well in such an environment. Authorizations are given to the views. The authorizers have then the assurance that users are not able to derive any information for which a right has not been explicitly granted. This is possible because the only information the user can obtain from the system are views which authorizers control directly through authorizations defined on views.

Open systems perform poorly when access has to be limited to a small number of views. To allow access exclusively to certain views a long and complex list of constraints has to be established. The authorizers have to be aware of all the possible ways a user can derive information from the DDBMS since users are allowed to access any data unless a constraint forbids it.

In the hybrid system views can be defined through the authorizations by allowing only specific JOINS, PROJECTIONS and RESTRICTIONS to be performed.

2. Extensive ability rights.

In the case of extensive ability users may be given rights to access basic relations and to use these relations in any number of operations without, or with only a few, RESTRICTIONS. This may be the case in an environment where authorized data and the data infered from it usually have the same degree of security associated with them. Extensive ability rights could also be required in an environment where users to which rights are given are trusted not to misuse them.

To allow a user to make any type of serial computation on an authorized relation in a closed system, either all serial computations are allowed for all authorized relations or a long list of authorizations has to be established. However, different types of users and of data may exist and the extensive ability authorizations may apply to only certain types of combinations of user-data. In this case allowing all serial computations for all authorized relations is not anymore a feasible solution. It is also impossible to put even a few restrictions on computed data with this solution.

Open systems handle this type of environment better because a few constraints can be defined to deny access to the small number of operations which have to be restricted. However, a list of constraints has to be established to restrict the number of relations accessible by a given user. Depending on the powers of each user this list may be a fairly short one, like in the case of a network handling public data or may be a long one if users have only access to relatively small parts of the DDB.

In the hybrid system users can be given authorizations for only the required relations and allowed to perform any serial computations on them with the exception of a few operations which are specified by a small number of constraints.

3. Average ability rights.

In this final case users may be given access to some relations and be allowed to perform only a certain number of serial computations using them. This is the most general case and represent the typical DDBMS application.

In closed systems authorizations are given to the relations and explicitly to all permissible serial computations. This may still corresponds to a relatively long list of authorizations for computed data.

Using an open system implies making a long list of constraints to define the relations to which a user does not have access.

In the hybrid system authorizations are given for the accessible relations and for some basic operations using these relations. More complex serial computations can be performed without having to explicitly grant authorizations for them. In addition constraints can be defined to control the more complex serial computations and allow an authorizer to define clearly among all the possible serial computations a user can perform the ones that are not permissible. In a typical DDBMS we expect this list of non permissible serial computations to be smaller than the list of all serial computations that should be allowed.

Open systems perform better than closed ones when users are to be given extensive rights while closed ones are more adequate if users have limited abilities. The hybrid system performs satisfactorily in all cases, and handles the average case better than either closed or open systems.

3.6 AUTHORIZATION-CONSTRAINT MECHANISM.

The driving mechanism of the hybrid system is the authorization-constraint mechanism. This mechanism is represented in figure 3 where the regulation of transactions is illustrated by control lines. Transactions to any of the data bases are regulated by the authorization-constraint mechanism. Any query or update to a data base supported by the DDBMS has to satisfy the access control rules defined by the authorizations and constraints. Authorizations and constraints are also organized in relational data bases and are therefore controlled by the same mechanism. Thus, the authorizations and constraints form a self-regulating, interdependent mechanism.

Hence, we have an integrated system where the two major elements, authorizations and constraints, control all data bases themselves included.

After describing the different types of authorizations and constraints, we will present an example to illustrate the authorizations-constraints mechanism.

3.6.1 AUTHORIZATIONS.

Authorizations can be defined on any of the data bases existing in the system. The type of authorizations defined on any given data base depends on the kind of operation for which the authorization is defined. Relational data base operations can be divided in two major categories. The first category is composed of access operations which include READ, WRITE, UPDATE, and DELETE, which are defined on single relations from the DDB. Calculation operations such as JOIN, PROJECTION and RESTRICTION compose the second category.

Since a JOIN is a binary operator, two relations are involved in the authorization for such an operation. PROJECTIONs can also be used in authorizations for JOINS to eliminate domains on which a JOIN is not allowed. PROJECTIONs can also be de-

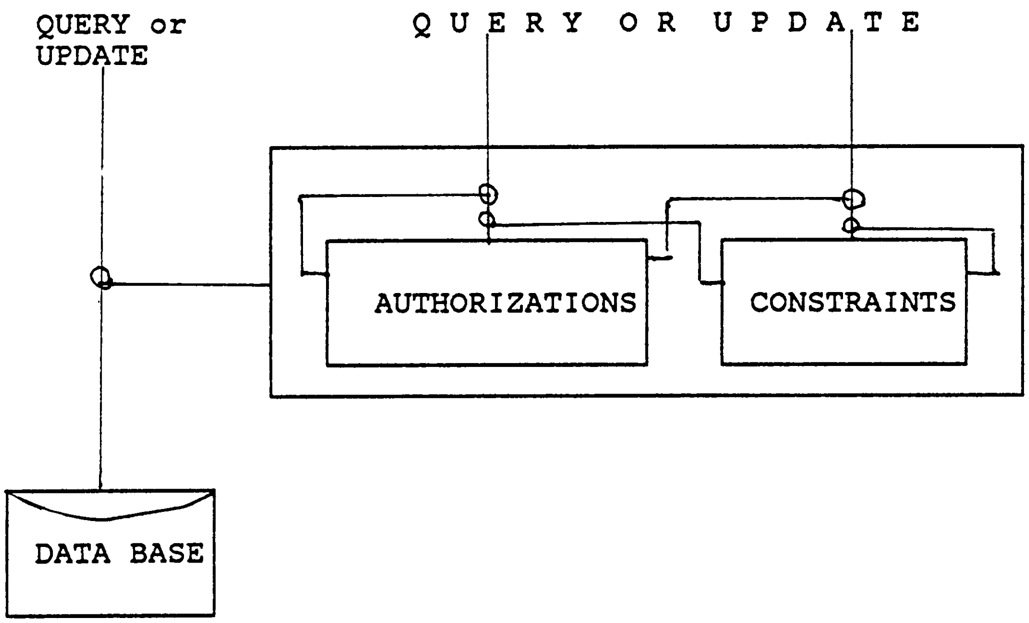


Figure 3. Authorization-Constraint Mechanism

fined in authorizations for access operations to give an access right to the PROJECTION of a relation instead of granting the right to the whole relation. RESTRICTIONS can be expressed in authorizations for access operations, by using a data dependent predicate to define the RESTRICTION. The authorizations format is based on Hartson's model (HARTH81). The authorizations are stored in a relation with the domains defined below. The following two types of authorizations are defined, depending on the type of operations authorized.

AUTA = (A,O,U,R,B,S) for access operations and

AUTC = (A,J,U,R_1,R_2,B,S) for calculation operations.

where:

A is the authorizer's name

U is the user or group name

O is the access operation U is authorized to perform on R, i.e. READ, WRITE, UPDATE or DELETE.

J is the JOIN operation U is authorized to perform between R_1 and R_2.

R	is the relation U is authorized to access. It can refer to a relation stored in any of the data bases.
R_1, R_2	are the relations U is authorized to JOIN. R_2 can be replaced by a "*" indicating that R_1 can be joined with any relation in the DDB.
B	is a bit coded field used to define PROJECTIONS on the data field. The correspondence of the bits of the field to domains of the data relation is given by the order of the domains as defined when the relation is created. A value of 1 in a bit indicates that the user has access to the corresponding domain of the relation.
S	is a predicate defining under which system conditions this authorization is in effect. S can be used to express a data dependent predicate defining a RESTRICTION.

The system conditions can also mention the user's site, the data site, or network characteristics such as the traffic status or the state of each node operating.

The authorizations can be defined on any of the data bases maintained by the system including the main users, the authorizations, the constraints, the directory, the groups, the users and authorizers profiles data bases.

A problem may arise for authorizations on joins between relations controlled by different authorizers. It may not be clear which authorizer has the authority to give such a right. In the hybrid system we solve this problem by requiring that both authorizers separately give a partial authorization. A user is then granted a JOIN right over two relations controlled by two different authorizers if and only if the two partial authorizations have been granted by the two authorizers. Lets suppose for example that authorizer A1 controls relation R_1 and authorizer A2 controls relationn R_2. Lets further suppose that U needs to be granted JOIN right over the two relations. A1 has then to introduce the partial authorization:

PAUTA1 = (A1, U, J, R_1, R_2, A2, S1)

which indicates that A1 authorizes U to JOIN R_1 with R_2 provided U obtains the same right from A2.

Similarly A2 has to grant U the following partial authorization:

PAUTA2 = (A2, U, J, R_2, R_1, A1, S2).

The two partial authorizations are then merged together in the system in the following form:

AUT = (A1 U A2, U, J, R_1, R_2, S1 S2).

The partial authorizations are not treated in the same way as the authorizations by the hybrid system. Partial authorizations are not effective until they have been merged into a complete authorization.

3.6.2 CONSTRAINTS.

To meet the three main requirements for DDBMS identified in the preceding section, three types of constraints are defined: operations, flow control and routing constraints.

OPERATIONS CONSTRAINTS

These constraints regulate the type of operations a user is allowed to perform. They can be divided in two categories.

1. Access constraints.

   A user is allowed to access a relation, (i.e. READ it, WRITE it, UPDATE it, DELETE it or make a JOIN on it), only if an authorization exists for the operation. However, it may be desirable for the authorizer to have a means of preventing a user from ever obtaining a specific right on a relation. This could be particularly important when a relation is controlled by more than one authorizer. In this case, although authorizers trust each other, they may need to be assured that another authorizer would not give away sensitive rights without consulting them.

   As for authorizations, two types of constraints can be defined depending on the type of operation.

$CONO = (A,O,U,R,S)$ for constraints on access operations

$CONJ = (A,J,U,R_1,R_2,S)$ for constraints on JOINS

where the different domains defined in the constraints are the same as the ones in the corresponding authorizations with the difference that instead of defining the granting a right, it defines the denial of one.

The problem of different authorizers for relations R_1 and R_2 mentioned previously for authorizations is solved in a similar fashion for the constraints.

As was mentioned before, operation constraints can be defined on any data base defined in the DDBMS. When operation constraints are defined on the authorization or constraint data bases, the predicate part of the constraint is used to specify the authorization or constraint on which the operation constraint is defined.

2. Computational constraints.
These constraints allow the control of data derived from the DDB by users through authorized serial computations. They are used to avoid a user obtaining two domains together in a computed relation. They have the following format:

$CONC=(A, U, D_1, D_2, S)$

where:

A	is the constraint imposer's name.
U	is the user or group name.
D_1, D_2	are the domains which cannot appear together in any serial computation.
S	is a predicate defining the condition under which the constraint has to take effect.

Although constraints on joins could be expressed using the computational constraints, we chose to keep both types of constraints for convenience. Authorizations on joins could also be eliminated allowing for any JOIN to be performed on relations to which a user has access while controlling the joins by means of computational constraints. However, some combinations of rights can be expressed more efficiently if written in terms of authorizations and constraints on joins rather than in terms of access operations authorizations and computational contraints. This is illustrated in the following example: suppose that we have two relations $R_1$and R_2 with the following domains:

R_1: D_1, D_2, D_3

R_2: D_3, D_4.

Suppose further, that a user is given the right to JOIN R_1 and R_2 but not to access domains D_1 and D_3. If we do not allow for authorizations nor constraints defined on joins the user has to be given authorizations to access domains D_2 and D_3 of relation R_1 and domains D_3 and D_4 of relation R_2. In addition, computational constraints have to be defined to avoid the user obtaining domains D_2 and D_3 or domains D_3 and D_4 together. The combination of these authorizations and constraints have the effect of allowing the user access to the JOIN of relations R_1 and R_2 with a projection over the domains D_2 and D_4. The user is therefore not given access to D_1 nor D_3.

However, if we can define authorizations on joins it is then sufficient to give the user an authorization to the restriction on D_2 and D_4 of the JOIN of the two relations.

FLOW CONTROL CONSTRAINTS

Constraints on flow control regulate the transfer of information between users. A user may perform authorized serial computations on a set of relations and then be the owner of the computed relation. As an owner the user may give access to the relation to other users and therefore transfer information obtained from the DDB to other users. This flow of information can be controlled if additional constraints are associated with the authorizations the user supplies to access data in the DDB. Sensitive data may be protected by not allowing the user to have the right to give other users access to data computed from sensitive data. Different levels of users can then be defined. Some users might have the right to access data, to have full ownership on the computed data, and to give access to it to other users. Less trusted users would have only the right to compute data but without the privilege of full ownership. This division is equivalent to the division between "administrative" and "property rights" as described in (BUSSO81). The system of authorizations-constraints is well suited for creating such different types of users, since the ability to give authorizations and the associated constraints can precisely define the type to which a user belongs, and allow for the definition of a large number of types of users.

The flow constraints have the following format:

CONF = (A, R, O, U1, U2, S)

where

A is the constraint imposer's name

R is the relation to which U1 has the access right O which should not be transfered to U2.

O is the access right on R which should not be transfered.

U1 is the user who has access right O to R.

U2 is the user who should not acquire the access right O to R.

S is the access condition under which the constraint is imposed

A "*" can replace any of the fields O, U1, U2, or S to specify that the constraint is to be effective for any value of that specific field.

ROUTING CONSTRAINTS

These constraints are defined to regulate the flow of information between sites. Two types of routing constraints are defined:

1. Temporary routing constraints.
   These constraints are defined to regulate the movement of data among sites during query processing. Certain relations may not be transfered to certain untrusted sites even if they are only temporarily stored at these sites. These constraints have the following format:

   CONR = (A, R, S1, S2, S)

where

A is the constraint imposer's name.

R: is the relation which cannot be transfered from S1 to S2.

S1 is the site where R is stored.

S2 is the site to which R cannot be transfered

S is the predicate defining the conditions under which the constraint is to take effect.

2. Storage constraints.

   These constraints are used to avoid having sensitive data stored at mistrusted sites. The storage constraints are checked when the relation resulting from a query is stored permanently at a site. These constraints also take effect when data is moved from one site to another for efficiency reasons during maintenance operations of the DDBMS. The following format is used for the storage constraints:

   CONTS = (A, R, S1, S)

where:

A is the constraint imposer's name

R is the relation that cannot be stored at S1.

S1 is the site which cannot hold the relation R.

S is the predicate defining the conditions under which the constraint is to take effect.

As for authorizations, relations are created to hold the constraints. It should be noted that the relations holding the authorizations and the ones holding the constraints are in third normal form. This is true because the keys for each one of these relations are composed of all the domains of the relation.

3.6.3 EXAMPLE.

To illustrate how the security requirements for DDBMS can be satisfied with the proposed system we will present an example of a small DDB where access control rules are defined. Consider the following DDB composed of four relations distributed among three sites.

Site	Relation	Domains
Payroll Office	Employee Department	SSN, Name, Dept#. Dept#, Name of dept, Code# of head, Address of head.
Bank	Account	Account#, Code#, Balance, Address.
Registar's Office	Course	Course name, SSN of instructor, Address of instructor.

A payroll employee U is given access to these four relations and their respective JOINS by means of eight authorizations. The following authorizations are granted to user U by authorizer DBA who in this case is the data base administrator:

AUT1 = (DBA, U, RW, Employee, 111, *)
AUT2 = (DBA, U, RW, Department, 1111, *)
AUT3 = (DBA, U, RW, Account, 1111, *)
AUT4 = (DBA, U, RW, Course, 111, *).
AUT5 = (DBA, U, J, Employee, *, 111, *).
AUT6 = (DBA, U, J, Department, *, 1111, *).
AUT7 = (DBA, U, J, Account, *, 1111, *).
AUT8 = (DBA, U, J, Course, *, 111, *).
where "*" in the last field indicates that there are no restrictions put on the authorization.

These authorizations give user U READ and WRITE access to the four relations in the DDB as well as to all the possible JOINS on the DDB. Suppose however, that the data base administrator wants to prevent U from obtaining the name and the balance of any employee. Using computational constraints this can be expressed by the following constraint:

CONC = (DBA, U, NAme, BAlance, *)

U could obtain the name and balance of an employee in at least two different ways. Joining the relations Employee and Department on the domain Dept, and Joining the resulting relation with the relation Account on domain Code, the user can associate the name of all head departments and their balance. A JOIN of Employee and Course relations on domain SSN followed by a JOIN with the relation Account on domain Address allow the user to derive the name and balance of instructors.

If a strictly closed system was used, we would have two options: either allow all possible JOINS between relations, or specify explicitly all the allowed JOINS. In the first case, it would be impossible to avoid the user obtaining name and balance of employees. In the second case, the list of authorizations would be significantly long, and the data base administrator would have to analyze carefully the effect of the combination of these rights to determine if the user can derive unauthorized information. Using the constraint mechanism allows for better control, flexibility and ease of use.

If a bank employee UB is given access to the relation Account but has an associate at the payroll department UP with accesses to the relations Employee and Department the Data base administrator may require that the rights of UB be constrained so that they cannot be passed to UP under any condition. Assuming that user UB has the authorization:

AUT20 = (DBA, UB, RW, Account, 1111, *)

The following flow constraint is then defined:

CONF = (DBA, Account, RW, UB, UP, *).

The Account relation may be considered as being sensitive data and its transfer from the bank to the registar be prohibited under any condition. This translates into the following temporary routing constraint:

CONR = (DBA, A, BANK, REGISTAR, *)

Lets suppose that the following query Q is presented at the Payroll department:

Q: Get SSN, name and code of all instructors of CS courses.

The decomposition process for such a query would be:

1. Restrict Course by CS course name. Obtaining C1 at the Registar.

2. Either:

a. Send A to Registar. Compute CR = A[AD]C1 at registar and send CR to the Payroll department.

b. Or send C1 to the Bank. Compute CR = C1[AD]A at the Bank and send CR to the Payroll department.

3. Compute R = CR[SS]E at Payroll department. Project R on (SSN, Name, Code).

To satisfy the constraint CONR the second decomposition has to be chosen in step 2.

3.6.4 APPLICATIONS.

To illustrate how the authorizations-constraints mechanism can be used to efficiently solve standard protections problems that may arise in a DDBMS we will present two examples of such problems. These problems are typically difficult to solve with systems that are either purely closed or purely open.

REVOCATION OF RIGHTS.

The process of revocation of authorizations can be considered from two different viewpoints: from the authorizer's point of view and from the user's point of view. These two aspects of the revocation process may be in conflict. Authorizers require from the system the ability to withdraw authorizations from users who are not anymore trustworthy. On the other hand users who invest in computed data want to be protected from a sudden revocation of their rights.

Because of its integrated nature, the hybrid system can easily handle this situation. Since authorizations and constraints can be defined on the authorizations and constraints data bases themselves, conditions can be put on an authorization to restrict the rights of the authorizer. Three different approaches could be taken to solve this problem.

1. Conditioned authorization.
   The authorizer is given the right to create an authorization but is not allowed to UPDATE or DELETE it. In this case the user is assured that the rights will never be revoked by the authorizer. However, a problem may arise when the authorization has to be removed for good reasons, since the authorizer loses all powers over the authorization once it is granted.

2. Constraint definition.
   An access constraint can be put on the authorization so that the authorizer is not allowed to DELETE or UPDATE the authorization. The authorizer then has the right to create such a constraint but the rights to UPDATE or DELETE this constraint are left to the user. By adding the constraint the user is not given any rights to the au-

thorization but is still able to control its revocation. The authorizer does not have complete control over the authorization, but by controlling the creation of the constraint keeps a control over the user's right to keep the authorization.

3. Contract server.
   A variation of the preceding solution is to introduce a third party called contract server controlling the rights to both authorizations and constraints. The contract server would create the authorization and the constraint, and give rights on these to the authorizer and the user. The contract server could have the ability to revoke rights of any of the two parties in case the contract was violated by one of them. The presence of a third party insures that neither the authorizer nor the user retain its powers in face of evident damage. The authorizer cannot possess all powers because the user would be unsatisfied, while giving rights to the user over authorizations and the creation of constraints may violate the security of the system.

A contract between authorizer and user may be needed in the example previously presented if user U is the economics department which obtained a grant from the government to make a study of spending patterns by zone. Supposing that the relation Account has a domain Zone in addition to the previously defined domains, U would need access to Acct, Balance, and Zone. Because of the money involved in the grant, the economics department needs the assurance that this right will not be revoked at any point during the research. The department would, therefore, pass a contract with the Data base administrator to keep the access during the whole period of the grant.

VIEW DEFINITION.

The second example will illustrate the use of the hybrid system to define different types of views of the DDB for various groups of users.

- An access right may be given to a PROJECTION of the JOIN of two relations without giving access to either of the two relations. In our example the right to obtain the list of the names of all employees as well as the name of their departments can be granted to a user U' without giving U' the right to access either the Employee relation or the relation Department. An authorization granting U' access to the RESTRICTION on domains NA and ND of the JOIN of Employee and Department is created. It should be noted that in this example no access right is given to the Joining domain. This shows that it is possible to allow users access to JOINS without necessarily having to give them access to the individual relations nor to the domain used for the PROJECTION.

- Access rights may also be given to the individual relations and not the JOIN.

- Access rights to the relations and to their JOIN may be granted, while by the use of constraints restricting the set of data that can be derived. This was illustrated in our previous example where authorizations for the relations and their JOINS were restricted by some constraints.

To allow authorizers to define authorizations on views instead of individual relations, an authorizer interface could be used. The role of the authorizer interface would be the translation of authorizations and constraints on views to authorizations and constraints on the relations composing the views.

3.7 COMPUTATIONAL CONSTRAINT ENFORCEMENT ALGORITHMS.

Of all the constraints previously defined the computational ones are the most difficult ones to implement. We therefore have identified some algorithms to enforce the computational constraints. These algorithms fall in two main categories: static and dynamic.

3.7.1 STATIC ALGORITHM.

The purpose of the algorithm presented in this section is to identify potential violations of computational constraints and to modify the authorizations so that the enforcement of the constraint is assured.

For our purpose we define a serial computation as being a set of JOIN operations between a set of relations R_i such that:

(1) $(R_{i_1} [D_{j_1}] R_{i_2}) [D_{j_2}] R_{i_3} \ldots\ldots R_{i_n}$

where the JOINS may be made in any order,

D_{j_1} is common to both R_{i_1} and $R_{i_{1+1}}$ and PROJECTIONs can be made after each JOIN on domain D_{j_k}, where at least one domain common to R_{i_k} and $R_{i_{k+1}}$ is projected.

An authorized serial computation for a user U is a serial computation of the form (1) for which U has a READ authorization for each relation R_{i_j} and a JOIN right for R_{i_j} with all other relations in the serial computation. A computational constraint CONC=(A,U,X,Y,S) can be introduced by an authorizer A expressing that a certain user U should not be allowed to create a relation with X and Y as domains by any serial computation or series of serial computations made via the Distributed Data Base System or outside of it using the information derived from the Distributed Data Base. It should be noted that we assume that the user does not have access to data outside of the Distributed Data Base which could be combined with the one of the Distributed Data Base in such a way that the constraint is violated.

<u>Definition.</u>

A violating authorized serial computation for the constraint CONC=(A,U,X,Y,S) is a serial computation of the form (1) where X is in R_{i_1}, Y is in R_{i_k} and the PROJECTIONs used are such that both X and Y are projected, with $1 \le 1,k \le n$.

<u>Example:</u>

Let Name be a domain of a relation R_1 and Salary be a domain appearing in both relations R_2 and R_3. The computational constraint CONC = (ADMIN, FAROUK, NAME, SALARY, *) would state that the authorizer ADMIN wants to deny user FAROUK the ability to relate Name and Salary together directly or indirectly under any condition. Name and Salary can be accessed individually but the user should not be able to make a link between the name and the salary of a person.

In general a link can be made between X and Y via a certain number of JOINS between relations having common domains. Let D_i be a domain common to relations R_i and R_{i+1}. Then a series of JOINS by user U of relations R_i and R_{i+1} on D_i for i=1,...,n-1 can violate the constraint CONC=(A,U,X,Y,S) if X appears in relation R_1 and Y in relation R_n. If such a sequence of JOINS exists, then a link is said to exist between X and Y. Note that the JOINS can be made in any order and that the JOINS can be combined with PROJECTIONs, as long as the common domains are projected.

More formally we can express this as :

Lemma 1

A constraint CONC=(A,U,X,Y,S) can be violated if and only if there exists a violating authorized serial computation for this constraint.

Proof:

Let CO be any violating authorized serial computation. CO is of the form (1) with X in R_{i_1} and Y in R_{i_k}. The relation resulting from CO has X and Y as domains and therefore the constraint CONC=(A,U,X,Y,S) is violated.

Since the only type of operations considered are JOINS and PROJECTIONs as we shall see later, the only type of possible series of operations is a serial computation. Since X and Y have to appear in the final relation obtained, to violate the constraint, the serial computation has to be a violating serial computation.

In general the domains named in the constraint, X and Y can appear in any number of relations. They may appear in the same relation, in which case access to the relation has to be restricted so that only one of the two domains can be accessed. They may also appear in different relations and could be obtained together by a series of relational operations. In this case the only operators that need be considered are the ones dealing with two relations since X and Y appear in distinct relations. The major primitive relational algebra operators having two relations as operands are the JOIN, union , intersection, set difference and division. The division operator can be expressed in terms of other algebraic operators and can therefore be discarded as a primitive operator. Among the set theoretic operators union, intersection and set difference only union can create a link between domains appearing in two different relations, through a domain common to both relations. This kind of link is similar to the one created by JOIN operations, and we will therefore consider only JOIN operations.

One way of enforcing the constraint CONC=(A,U,X,Y,S) is to deny the user access to X (or alternatively Y) from any relation. This solution would correspond to breaking the sequence of JOINS permitting the link between X and Y at either end of the sequence. This may be an unreasonable solution if the user has a legitimate need to access both X and Y separately. For example, a list of names may be necessary for administrative purposes, while a list of salaries may be necessary for statistical purposes, by the same user.

If the sequence cannot be broken at either end then another domain in the sequence has to be chosen.

Yet another solution is to remove all the authorizations permitting JOINS of relations containing X (alternatively Y), so that no sequence of JOINS can be built linking X and Y.

However this solution is an inadequate one since the user can still perform the JOINS outside of the system if access to domains involved in the JOINS is not denied. If access to these domains is also denied, then this solution may be too restrictive since not all the JOINS from relations containing X (alternatively Y) may be part of a sequence linking X and Y.

Since any domain appearing in more than one relation can be used in a JOIN operation, X and Y can be linked by more than one sequence of JOINS. Lets define the sequence of domains used in the sequence of JOINS as a sequence Joining domains. The first domain of this sequence will always be taken as X while the last one will always be Y. The second domain of the sequence of Joining domains can be any domain appearing in the same relation as X, and common to at least another relation. Therefore if there are more than one common domain in a relation containing X, there may be more than one sequence of JOINS linking X and Y. This will be true for all the relations containing X. Similarly the domain before the last on the sequence of Joining domains can be any common domain appearing in the same relation as Y. In general let D_i be the ith domain in the sequence of Joining domains. The ith+1 domain of the sequence can be any common domain appearing in the same relation as D_i. It is therefore evident that there may be more than one sequence of JOINS linking X and Y.

NOTATION.

Some of the initial notation used in the development of the algorithm is:

R_i $1 \le i \le N$:

is a relation of the DDB numbered i. Any numbering scheme may be used.

D_k :

is a domain appearing in the DDB.

Common Domain:

is a domain appearing in more than one relation.

D_i is a common domain iff there exist k,l $k \ne l$ such that D_i is in R_k and D_i is in R_l.

Unique Domain:

is a domain which appears in only one relation.

D_i is a unique domain iff there exists l such that D_i is in R_l and D_i is in R_k for $1 \le k \le N$ and $k \ne l$.

CD:

is the set of common domains.

UD:

is the set of unique domains.

CONC=(A,U,X,Y,S):

is a computational constraint introduced by authorizer A expressing that X and Y cannot appear together in any relation created by user U.

Database Graph G(V,E,C):

is an undirected graph whose vertex set V represent relations of the DDB and domains of relations, and the edge set E contains edges of the form D_i--R_j if and only if domain D_i is in relation R_j. C is a cost function associating to each edge D_i--R_j a cost C(i,j).

CONSTRAINT ENFORCEMENT ALGORITHM.

We define a constraint enforcement algorithm as being an algorithm which when applied for a specific constraint CONC=(A,U,X,Y,S) assures that the constraint cannot be violated by any serial computation done on behalf of user U.
A constraint CONC=(A,U,X,Y,S) can be violated by any series of JOINS linking X to Y. These JOINS are possible only between relations having common domains. The constraint enforcement algorithm presented in this section determines

all the possible series of JOINS leading to a violation of the constraint and modifies the authorizations so that each link between X and Y is broken at some point. If no link between X and Y remains then the constraint cannot be violated. To determine all the possible links between X and Y an undirected graph called the database graph is constructed. The nodes of the graph are relations and domains of relations of the DDB for which the user has an authorized access. Edges exist only between relations and domains. An edge is constructed between domain D_k and relation R_i if and only if D_k is a domain of relation R_i. If two domains D_k and D_l are in the same relation R_i, there exists a path D_k--R_i--D_l between D_k and D_l in the graph indicating that D_k and D_l can be obtained together by an unrestricted READ operation of R_i. or a PROJECTION of R_i containing both D_k and D_l. If one of them, say D_k, is common to another relation R_j then domains in R_j can be linked to D_k and D_l by a JOIN operation of R_i and R_j on domain D_k. This is reflected in the graph by paths existing between the domains in R_j and the domains in R_i. An important property of the database graph is that a path exists between two domains in the graph if and only if the domains are linked by a series of JOINS or if the domains appear together in the same relation.

This property is expressed by :

<u>Lemma 2</u>

Given a constraint, CONC=(A,U,X,Y,S), there exists a path in the graph between two domains X and Y if and only if there exists a violating authorized serial computation.

Proof:

We will first show that if there exists a path between X and Y, then there exists a violating authorized serial computation. Let the path be:

$P= (X-R_1-D_1-R_2-\ldots\ldots-R_i-Y)$.

such that D_{j-1} and D_j are both in R_j for $j<i$.

Lets us now prove by induction that there exists an authorized serial computation

$CR_n = CR_{n-1}[D_n]R_{n+1}$ for $n<i$.

where $CR_1=R_1[D_1]R_2$.

Base case n=1.

In the graph there is a path $R_1-D_1-R_2$ which implies that D_1 is in both R_1 and R_2 and that the user has access to both R_1 and R_2. So there exists an authorized serial computation $CR_1=R_1[D_1]R_2$.

Hypothesis of induction:

suppose that there exists an authorized serial computation $CR_n=CR_{n-1}[D_n]R_{n+1}$ for any n such that $n<i$.

Lets prove that it is also true for n+1 if $n+1<i$. By construction of the path D_{n+1} is in R_{n+1} and R_{n+2}. Since D_{n+1} is in R_{n+1} it is also in CR_n by hypothesis of induction. D_{n+1} being in both CR_n and R_{n+2} there exists an authorized serial computation $CR_{n+1}=CR_n[D_{n+1}]R_{n+2}$ for $n+1<i$

This completes the proof by induction.

Since there exists an authorized serial computation of the form $CR_n=CR_{n-1}[D_n]R_{n+1}$ for any n<i it is true in particular for n=i-1. There exists an authorized serial computation $CR_{i-1}=CR_{i-2}[D_{i-1}]R_i$, where $CR_1=R_1[D_1]R_2$. X is in R_1 so X is also in CR_1 and Y is in R_i so it is also in CR_i-1, and therefore X and Y are both in CR_i-1 and CR_i-1 is a violating authorized serial computation.

Lets now prove that if there exists a violating authorized serial computation then there is a path in the graph between X and Y. Let CR_m be the violating authorized serial computation and X be in the first relation of CR_m and Y in the last one. Lets prove that there is a path in the graph corresponding to the violating authorized serial computation, by induction on the number of operations in the authorized violating serial computation.

Base case: let n be the number of operations in the authorized serial computation be one. Let $CR_n=R_i[D_i]R_j$ since D_i is in both R_i and D_j, there is an arc between R_i and D_i and another one between D_i and R_j, so a path exists R_i--D_i--R_j.

Suppose now that there is a path in the graph for any authorized serial computation CR_n derived from CR_m for n<m between the first relation in CR_n and the last one. Lets add one operation from CR_m to CR_n: $CR_n+1=R_i[D_i]CR_n$. The order in which the JOINS are made is not important. The result

being the same we arbitrarily make the JOIN of R_i with CR_n which is equivalent to having JOINS made between CR_k and CR_l where if k and l represent the number of operations in CR_k and CR_l respectively then k+l=n+1. D_i is in R_i so there exists an edge R_i--D_i. D_i is also in CR_n so there exists a relation R_nj such that R_nj is part of the serial computation CR_n and D_i is in R_nj, and there is an arc D_i--R_nj. Therefore there is a path R_i--D_i--R_nj in the graph. Since the order in which the JOINS are performed is not important we can reorder the JOINS in CR_n so that R_nj is the first relation appearing in CR_n. $CR_n=R_nj[D_nj].....R_nk$. There is therefore a path from R_i to R_nk in the graph. This concludes the proof by induction.
It is particularly true for n=m and there is a path between the first and the last relation in CR_m. Since by construction of CR_m X is in the first relation and Y in the last one, there exists a path between X and Y in the graph. This completes the proof of lemma 2.

We can now, using the two lemmas, prove that the arc cutting algorithm is indeed a constraint enforcement algorithm.

<u>Theorem.</u>
The arc cutting algorithm is a constraint enforcement algorithm.
<u>Proof.</u>

By definition a constraint enforcement algorithm is an algorithm which when applied for a specific constraint CONC=(A,U,X,Y,S) assures that the constraint cannot be violated by any serial computation done on behalf of user U. By lemma 1 a constraint CONC=(A,U,X,Y,S) can be violated if and only if there exists a violating authorized serial computation for CONC=(A,U,X,Y,S).

Thus to enforce a computational constraint CONC=(A,U,X,Y,S) all authorized violating serial computations should be disabled. By lemma 2 an authorized violating serial computation is equivalent to a path in the graph. Disabling a violating authorized serial computation is therefore equivalent to cutting the corresponding path in the graph.

The arc cutting algorithm cuts all paths linking X and Y, consequently disables all the violating authorized serial computations for CONC=(A,U,X,Y,S). Therefore, the arc cutting algorithm is a constraint enforcement algorithm.

<u>Example:</u>

Let us consider an extension of the example used in the preceding section.

EMPLOYEE = E

SSN, NAME, DEPT#

DEPT =D

DEPT#, NAME OF DEPT, CODE# of head, ADDRESS of head

ACCOUNT =A

ACCOUNT#, CODE#, BALANCE, ADDRESS

COURSE =C

COURSE NAME, SSN of instructor, ADDRESS of instructor, DEPT#

LIBRARY CIRCULATION =L

BOOK#, BOOK NAME, CIRCULATION#

INVENTORY =I

BOOK NAME, AUTHOR, BOOK#, SUBJECT

```
SUBJECT =S

   SUBJECT, BOOK#
```

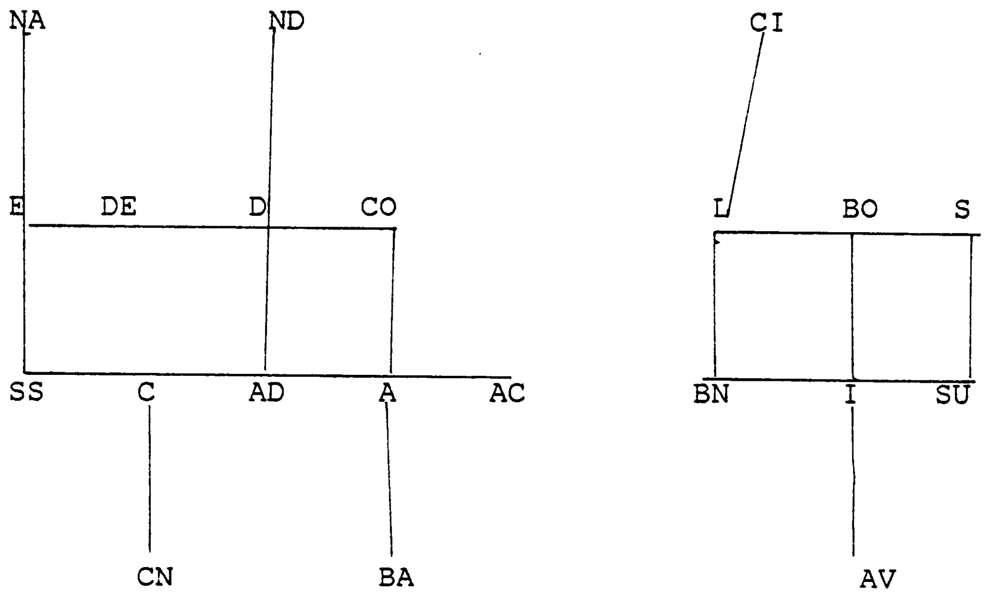
The list of common domains is given by :

CD = (SSN= SS, DEPT#= DE, CODE#= CO, ADDRESS= AD, BOOK#= BO, BOOK NAME= BN, SUBJECT= SU)

And the list of unique domains is given by:

UD = (NAME= NA, NAME OF DEPT= ND, ACCOUNT#= AC, BALANCE= BA, COURSE NAME= CN, CIRCULATION#= CI, AUTHOR= AU).

The corresponding database graph is :



In our graph there exists a path SS--E--DE--D--CO between SS and CO indicating that SS and CO can be linked by creating a relation which is the JOIN of E and D on DE.

When a constraint CONC=(A,U,X,Y,S) is introduced, all the paths between X and Y must be broken. Each path is broken by removing an edge that forms part of the path. Removing an

edge D_k--R_i from the graph corresponds to eliminating domain D_k from the user's view of R_i, since such an edge in the graph represents the fact that domain D_k appears in relation R_i. In our example removing the edge CO--AC from the graph would correspond to denying user U access to CO in AC, and disallowing any JOIN to be made between D and A on CO, since for user U CO does not appear anymore in A.
The presence of a domain D_k in a relation R_i may be of great importance to a user. The user may be willing to have two or more other domains removed from other relations rather than to have D_k removed from R_i. In our example a user may wish to keep the domain NAME in relation EMPLOYEE at any cost while being willing to lose access to CODE# of head in relation DEPT and ADDRESS of instructor in relation COURSE. One way of expressing the impact of the loss of a domain D(i) in a certain relation R_k for a certain user U is to assign a cost to the edge D(i)--R_k. In general a cost function C can be associated with the graph, which associates a cost C(i,j) to each edge D(i)--R_j.

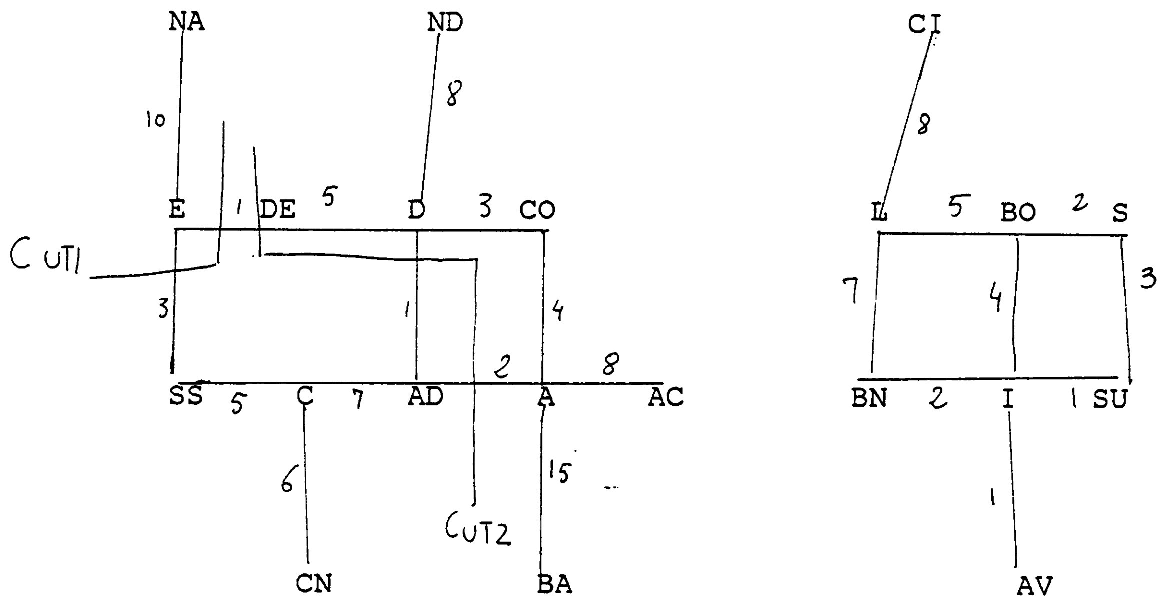
To reduce the computational constraint enforcement problem to a well known graph theory problem, we introduce the following notation. To define the cut set separating two nodes X and Y, the set of edges V of the data base graph is partitioned into two non empty sets S and S', such that X is in S and Y is in S'and S'=V-S. A cut set is then any set of edges joining S with S'. The cost of a cut set is the sum of the

costs of the edges of the set. A minimum cut set is then defined as a cut separating X and Y with minimum cost.
The problem becomes, therefore, one of finding the minimum cut set to break all paths linking X and Y given a certain cost function associated with the graph. Algorithms to solve this problem are widely available.

It should be noted that the database graph may have more than one connected component. In this case only one connected component is searched for a minimum cut set. If X and Y are not in the same component then there cannot be any path between them, and the constraint is always satisfied. Assume for our example that authorizer DBA introduces a constraint CONC=(DBA,U,NA,BA,*) for user U. Our graph contains two connected components: COMP1 containing the nodes : SS,DE,CO,AD,NA,ND,BA,CN,E,D,A, and C, and COMP2 containing the nodes: BO,BN,SU,CI,AV,L,I and S.

The only connected component searched is COMP1 since it is the one containing both NAME=NA and BALANCE=BA.

Further assume that user U has communicated to the Constraint Administrator which domains in which relations should be given priority. The Constraint Administrator in turn translates this priority list in terms of costs assigned to the edges of the graph, giving the following graph:



To isolate NA from BA the minimum cut set has to be found taking NA as the source and BA as the sink.

In this example there are two minimum cut sets :

CUT1 = {E--DE, E--SS}

CUT2 = {E--DE, D--AD, A--AD}

Each one of these cut sets divide the connected component COMP1 in two connected subcomponents.

CUT1 divides COMP1 in:

COMP3 = {NA,E} and

COMP4 = {SS, DE, CO, AD, ND, AC, BA, CN, D, A, C}

While CUT2 divides COMP1 into:

COMP5 = {SS, AD, NA, CN, E, C} and

COMP6 = {DE, CO, ND, AC, BA, D, A}

If the cut chosen is CUT1 the user will not be able to link NA to any other domain in the DDB, while if CUT2 is chosen it will not be possible to link any domain in COMP5 to any domain of COMP6. Depending on the importance to the user of the connections between the different domains one or the other cut sets would be chosen.

In general if there exists more than one minimum cut set the user should be given the connected subcomponents corresponding to the different cut sets, so that a choice can be made accordingly to the user's priorities.

The construction of the database graph is made at the time of the creation of the DDB, and updated when new relations are introduced or when domains are added to, or deleted from, an existing relation. Each time a new constraint is introduced the corresponding connected component is labeled with the costs assigned by the Constraint Administrator accordingly to the user's priorities, the minimum cut sets are found and the corresponding authorizations modified.

MODIFICATION OF THE AUTHORIZATIONS.

The minimum cut set for a constraint CONC=(A,U,X,Y,S), is defined by a set of pairs of the form D_k--R_i. For each one of these pairs all the authorizations for user U are checked to determine if they give access rights to domain D_k in relation R_i. These authorizations are modified so that right access to R_i is restricted by a PROJECTION excluding D_k. Authorizations giving access to JOINS of R_i with other relations on domain D_k are also determined and suppressed from the set of authorizations. After these modifications of the authorizations user U's view of relation R_i does not contain domain D_k. The sequences of JOINS linking X and Y involving D_k and R_i are not anymore possible.

After this process is carried out for all the pairs defining the minimum cut set, no sequence of JOINS linking X and Y is anymore possible and the constraint CONC=(A,U,X,Y,S) is satisfied.

In our example suppose we have the eight previously defined authorizations.

In our example user U has Read and Write access to relations E, D, A, C and to all of their domains, as well as all the domains defined by the JOINS of E and D on DEPT#, E and C on SSN, D and A on CODE#,D and A on ADDRESS, A and C on ADDRESS, and D and C on ADDRESS.

Lets assume in our example that the minimum cut set chosen is CUT1. Domains SS and DE have to be eliminated from E, and accesses to the JOINS of E on SS or DE have to be denied. The authorizations have then to be changed in the following way:

AUT1 = (DBA, U, RW, E, 010, *)

AUT2 = (DBA, U, RW, D, 111, *)

AUT3 = (DBA, U, RW, A, 111, *)

AUT4 = (DBA, U, RW, C, 1111, *)

AUT5 = (DBA, U, J, E, *, 010, *)

AUT6 = (DBA, U, J, D, *, 1111, *)

AUT7 = (DBA, U, J, A, *, 1111, *)

AUT8 = (DBA, U, J, C, *, 1111, *)

SSN = SS and DEPT# = DE are eliminated from the relation EMPLOYEE = E.

One of the advantages of this method is that the constraint enforcement algorithm needs to be run only at the time of definition of the constraint. The subsequent modification of the authorizations ensures that the constraints are correctly enforced at run time.

A second advantage is that a user is never able to obtain two domains mentioned in a constraint together either through the DDBMS or even outside the system by obtaining hard copy of the relations in the DDB and making the JOINS manually.

Another advantage of this method is that the rights of a user are independent of the order in which the user submits queries to the system. At any time it is possible to obtain a list of the data a user may access, provided no new constraints or authorizations are introduced. However, it may be preferable for certain users to have an access to data conditioned on previous accesses. This could be true if a user cannot forecast future queries, and cannot therefore, provide a good assignment of costs to arcs. One other drawback of this algorithm is its unpredictability. It is very hard to predict which authorizations will be modified when a new constraint is introduced.

3.7.2 DYNAMIC ALGORITHMS.

The limitations of the arc-cutting algorithm lead to more dynamic form of constraints enforcement algorithms.

TAGGING ALGORITHM

An authorization is said to be tagged when a constraint is associated with it. If a user is never allowed to access domains SS and DE together, the authorizations for SS and DE are each tagged indicating that the user is allowed to use SS directly or data derived from SS (respectively DE) as long as the serial computation does not involve DE (respectively SS). The authorizations for the computed data are also tagged, the tag is therefore propagated along the serial computations. This algorithm is dynamic because the computational constraints are enforced at execution time only.

Lets consider again the preceding example, where the data base administrator wants to avoid U obtaining the name and balance, or the SSN and account of any employee. Each authorization giving access to the domains mentioned in the constraints imposed by the data base administrator is tagged as follows:

AUT1 = (DBA, U, RW, E, 111, *, {CONC1})

AUT3 = (DBA, U, RW, A, 1111, *, {CONC2, CONC3})

AUT4 = (DBA, U, RW, C, 111, *, {CONC4})

AUT5 = (DBA, U, J, E, *, 111, *, {CONC1})

AUT7 = (DBA, U, J, A, *, 1111, *, {CONC2, CONC3})

AUT8 = (DBA, U, J, C, *, 111, *, {CONC4})

where CONC1, CONC2, CONC3 and CONC4 are pointers to the following constraints:

CONC1 = (DBA, U, NA, BA, *)

CONC2 = (DBA, U, BA, NA, *)

CONC3 = (DBA, U, SS, AC, *)

CONC4 = (DBA, U, AC, SS, *)

CONC1 specifies that balance cannot be obtained in conjunction with any domain appearing in an authorization tagged by CONC1.

Lets further suppose that U performs a JOIN of relations Employee and Course on domain SSN, obtaining the relation CE. Authorizations AUT5 and AUT8 are used to allow the JOIN to be performed. An authorization AUT9 is granted to U, and the tags of AUT5 and AUT8 are transmitted to AUT9

AUT9 = (DBA, U, RW, CE, 11111, *, {CONC1, CONC4})

If U later requests a JOIN of relations CE and Account on the domain address to be performed, the query is rejected because it violates constraint CONC1 which is tagged in the authorization for CE.

To show that the tagging algorithm enforces correctly the computational constraints we will prove the following theorem:

<u>Theorem</u>

The tagging algorithm is a constraint enforcement algorithm.

<u>Proof</u>

By definition a constraint enforcement algorithm is an algorithm which when applied for a specific constraint CONC=(A,U,X,Y,S) assures that the constraint cannot be violated by any serial computation done on behalf of user U. By lemma 1 a constraint CONC=(A,U,X,Y,*) can be violated if and only if there exists a violating authorized serial computation for CONC=(A,U,X,Y,S).

Let CR be a violating authorized serial computation for CONC=(A,U,X,Y,S).

CR = $(R_{i_1} [D_{j_1}] R_{i_2}) [D_{j_2}] R_{i_3} \ldots R_{i_n}$.

and let X be in R_{i_1} and Y in R_{i_k}.

CR being an authorized serial computation there is a set of authorizations used to authorize the serial computation. Let AUT(X,i) and AUT(Y,j) for i=1,...,m and j=1,...,m' be the subsets of authorizations used to authorize the serial computation and involving respectively X and Y. Since CR involves both X and Y AUT(X,i) and AUT(Y,j) have each at least one element. All authorizations involving X or Y are tagged,

so all authorizations in AUT(X,i) and AUT(y,j) are tagged. Since at least one of these authorizations is used to authorize CR, the tag is transmitted and the constraint checked which results in CR being disabled since it involves both X and Y.
This shows that all violating authorized constraints are disabled and the tagging is therefore a constraint enforcement algorithm.

HISTORY KEEPING

This algorithm has been proposed by Cohen (COHEN77). A history of user accesses is maintained and a graph representing the information the user has already obtained from the system is built. A user is denied access to more data if this new data combined with the knowledge already acquired violates a constraint. This is also a dynamic algorithm since the enforcement is done at query time.

It should be noted that Cohen (COHEN77) addressed the issue of computational constraint and gave solutions to the problem. However, we believe that using the tagging algorithm is more efficient because less information has to be retained in the system. Cohen addressed only the issue of computational constraints, while we have identified a larger group of constraints including the flow between users constraints and routing constraints. We propose an integrated view, i.e. all data bases, including authorizations and constraints, are controlled by a single authorization-constraint mechanism.

In Cohen's model constraints are defined only on the main users data base.

3.8 ACCESS CONTROL ENFORCEMENT ALGORITHM

We will now present the algorithm for the enforcement of access control through the authorizations constraints mechanism. The algorithm has two components: the authorization enforcement and the constraint enforcement. The authorization enforcement is made at query processing time. If the arc cutting algorithm is used, the constraint enforcement is made at the time of introduction of the constraints and the authorizations reflect the constraints at the time of query processing. If the tagging algorithm is chosen, the constraints have to be enforced at query processing time.

The authorization enforcement algorithm is based on Hartson's algorithm (HARTH81). In Hartson's model authorizations are defined on data objects that may not be physically stored in the data base but may be described by a series of operations on relations. In the hybrid system authorizations are defined on relations that are physically stored in the data base. The only operations used in the authorizations in Hartson's model are READ, WRITE, UPDATE and DELETE while in the hybrid system JOINS are also used. The first three and the last three steps of the authorization enforcement algorithm presented in this section are similar to the ones found in Hartson's algorithm. In the first steps the authorizations relevant to the user's query are fetched. The last steps of the algorithm are used to evaluate the access condition applying for the set of authorizations defined in the first steps. However, the intermediary steps which evaluate the authorizations to be applied are quite different in Hartson's algorithm and in the algorithm for the hybrid system, due to the inherent difference in the definition of the data objects and the operations involved in the authorizations. In the hybrid system authorizations are transmitted along with the serial computations. A computed relation always inherits the authorizations associated with the relations used to obtain this relation. This is true for all computed relations whether used as intermediary results or as responses to queries. The access condition of the resulting relation is a logical AND of the access conditions of the authorizations for the relations used in the derivation. The same principle is used in the tagging algorithm where computed relations inherit the constraints associated with the relations used in the serial computation. In this case the resulting access condition is a logical OR of the composing access conditions. The principle of authorizations constraints inheritance is implemented in the algorithms without explicitly creating the new authorizations or constraints.

Before the enforcement phase the query is decomposed into the following format: all JOINS are first performed, followed by all the PROJECTIONs and finally the RESTRICTIONS are made. This decomposition does not correspond necessarily to the decomposition which is used for the query evaluation. Before the query is evaluated it is decomposed according to an optimization process. However, from the protection point of view the two decompositions are equivalent and the first one allows the enforcement process to be simplified. The order in which JOINS are performed in a query is not important. We will show that PROJECTIONs in a query can always be delayed after the JOINS without modifying the result of the query.

We will use the following notation for the PROJECTION operator: $\pi_X(R)$ with $X=(D_1, D_2, \ldots, D_n)$ to express that R is projected over the domains $D_1, D_2, \ldots, D_n$.
We therefore want to show that

Theorem.

$(\pi_X(R_i))[D_k]R_j = \pi_{XUY}(R_i[D_k]R_j)$.
with

$R_i= (D_{i1}, \ldots., D_{in})$.

$R_j= (D_{j1}, \ldots\ldots, D_{jm})$.

$X= (D_{ix1}, \ldots.,D_{ixl})$ and D_k is in X.

$Y= (D_{j1}, \ldots\ldots., D_{jm})$.

we have:

$\pi_X(R_i) = (D_{ix1}, \ldots,D_{ixl})$

so we can write:

$(\pi_X(R_i))[D_k]R_j = (D_{ix1}, \ldots,D_{ixl},D_{j1}, \ldots,D_{jm})$

replacing the JOIN $R_i[D_k]R_j$ by its value:

$R_i[D_k]R_j = (D_{i1},...,D_{in},D_{j1},...,D_{jm})$

and projecting on XUY we have:

$\pi_{XUY}(R_i[D_k]R_j) = (D_{ix1},...,D_{ixl},D_{j1},...,D_{jm})$

which is what we had for the left hand side of the equality and therefore we have:

$(\pi_X(R_i))[D_k]R_j = \pi_{XUY}(R_i[D_k]R_j)$

So this shows that π can be factorized. This result can be generalized to any number of PROJECTIONS and JOINS since all combinations of PROJECTION an JOIN as expressed in the left hand side of the equality proven in the theorem can be replaced by the right hand side of the equality and the process repeated if the right hand side is joined with another relation. Therefore, the PROJECTIONS can be performed after all JOINS are executed.

3.8.1 AUTHORIZATION ENFORCEMENT ALGORITHM.

This algorithm is executed each time a query q is submitted to the system on behalf of a user U

The following steps are taken for the authorization enforcement algorithm.

(1) Determine the groups to which user U belongs.

$G(U) = \{G_i \ / \ U \text{ is in } G_i\}$.

(2) Find all authorizations for user U. This is called the franchise of user U and is denoted F(U).

$F(U) = \{AUT_i \ / \ U(q) = U(AUT_i)\}$.

(3) Find in F(U) the authorizations that have relations appearing in the query and are for the access operation specified in the query: O(q).

$F(U,AO,q) = \{AUT_i \ / \ AUT_i$ in $F(U)$, $O(AUT_i) = O(q)$ and $D(AUT_i) = R_j$ with R_j contained in $R(q)\}$.

(4) Find in F(U) the authorizations that have relations appearing in the query and are for JOIN operations.

$F(U,CO,q) = \{AUT_i \ / \ AUT_i$ in $F(U)$, $O(AUT_i)$=JOIN and $R_1(AUT_i) = R_j$ or $R_2(AUT_i) = R_j$ for R_j contained in $R(q)\}$.

(5) Find in the query all projected domains

$DP = \{D_i \ / \ D_i$ is in $\pi(q)\}$.

(6) Check that all domains in DP are accessible through an authorization from F(U,AO,q).

Evaluate {For all D_i in DP there is an AUT_j in F(U,AO,q) such that $D(AUT_j) = D_i\}$.

(7) Check that for all relations of the query involved in a JOIN there are authorizations in F(U,CO,q) to allow the JOINS with all other relations involved in the JOIN part of the query.

Let the set of all relations in the query involved in a JOIN be $J(q) = \{R_1, R_2, \ldots, R_m\}$.

Evaluate {for all R_i in J(q) there is AUT_{j_k} in F(U,CO, q) such that $R_1(AUT_{j_k}) = R_i$ and $R_2(AUT_{j_k}) = R_k$ for $k = 1, \ldots, m$ and $k \neq i\}$.

(8) Partition F(U,AO,q) into equivalence classes by having two authorizations be in the same equivalence class if they authorize access to the same relation

$F(U,AO,q)_i$ = {AUT_j in F(U,AO,q) and $R(AUT_j) = R_i$}.

(9) Partition F(U,CO,q) into equivalence classes by having two authorizations be in the same equivalence class if they authorize a JOIN to be made on the same relation.

$F(U,CO,q)_k$ = {AUT_j in F(U,CO,q) and $R_1(AUT_j) = R_k$}.

(10) Evaluate the effective access condition for the query. It is computed by first evaluating the access condition for each equivalence class defined in the two preceding steps, and then making a logical AND of all equivalence classes access conditions. The access condition for an equivalence class is found by doing a logical OR of the access conditions of its authorizations. The query is rejected if does not satisfy the effective access condition.

$EAC_i = U_j S(AUT_j)$ with j covering all the authorizations in the class i and i covering all equivalence classes of F(U,AO,q). $EAC_k = U_j S(AUT_j)$ with j covering all the authorizations in class k and i covering all equivalence classes in F(U,CO,q).

$EAC = (\cap_i EAC_i) \cap (\cap_k EAC_k)$.

3.8.2 COMPUTATIONAL CONSTRAINT ENFORCEMENT ALGORITHM.

When the arc cutting algorithm is chosen for constraints enforcement the min cut algorithm is invoked each time an authorization or constraint is introduced in the system.

If the tagging algorithm is chosen to enforce the constraints the following algorithm is performed after the authorization enforcement algorithm each time a query is submitted to the DDBMS.

(11) Find all constraints labeled in the franchise of U with respect to the query q.

$CONC = \{CONC_i \; / \; CONC(AUT_j) = CONC_i$ for all AUT_j in $(F(U,AO,q))U(F(U,CO,q))\}$.

(12) Evaluate the access condition of each constraint in CONC and eliminate the ones which do not satisfy their access condition.

$ECON = \{CONC_i \; / \; CONC_i$ is in CONC and $p(CONC_i) = true\}$.

(13) For each constraint evaluate if the two domains specified in the constraint appear together in the PROJECTIONs. If it is true for at least one constraint the query is rejected.

For all $CONC_i$ in ECON do

If $\{D_1(CONC_i)$ is in $\pi(q)$ and $D_2(CONC_i)$ is in $\pi(q)\}$

then reject the query.

3.8.3 EXAMPLE.

To illustrate the two algorithms the authorizations of our previous example are slightly modified to introduce two user groups G1 and G2 to which user U belongs. Access conditions are also added to the different authorizations.
The following authorizations and computational constraints are defined:

AUT1 = (DBA, G1, RW, Employee,111, P1, {CONC1})

AUT2 = (DBA, G2, RW, Employee,011, P2, {CONC1})

AUT3 = (DBA, G1, R, Department, 1101, P3)

AUT4 = (DBA, G2, RW, Department, 1101, P4)

AUT5 = (DBA, G2, R, Account, 1111, P5, {CONC2, CONC3})

AUT6 = (DBA, G1, RW, Course, 111, *, P6, {CONC4})

AUT7 = (DBA, G1, J, Employee,*, 111, P7, {CONC1})

AUT8 = (DBA, G2, J, Employee,*, 111, P8, {CONC1})

AUT9 = (DBA, G1, J, Department, *, 1111, P9)

AUT10 = (DBA, G2, J, Department, *, 1111, P10)

AUT11 = (DBA, G2, J, Account, *, 1111, P11, {CONC2, CONC3})

AUT12 = (DBA, G2, J, Course, *, 111, P11, {CONC4})

CONC1 = (DBA, NAme, BAlance, P21)

CONC2 = (DBA, BAlance, NAme, P21)

CONC3 = (DBA, SSN, ACcount, P22)

CONC4 = (DBA, ACcount, SSN, P22)

This set of authorizations and constraints gives members of group G1 read access to the relations Employee, Department and Course, and write access to the relations Employee and Course. These rights are granted provided members of group G1 do not obtain the Name and Balance nor the Account number and Social Security Number of any employee. Members of group G1 are also given the rights to JOIN relations Employee and Department with any other relation, provided they do not obtain the Name and Balance of any employee. Members of group G2 are granted read and write rights on relations Department and Employee, and only read right on relation Account. They are also given the right to JOIN any relation with any other one in the DDB. These rights are granted provided the members of group G2 do not obtain the Name and Balance nor the Social Security Number and Account number of any employee. The authorizations and constraints are to take effect when their respective access conditions are satisfied.

We will not spell out all the access conditions to avoid unecessary details for the example. As an illustration we will give predicates P1 and P2. P1 expresses that the terminal used should be a terminal at the Payroll office. P2 specifies that the authorization is to take effect only if the relation Employee, is accessed by a member of group G2 between 8:00 a.m. and 5:00 p.m.

At time t user U submits query Q1 requesting a list of SSN, name, balance and address of all instructors.

$Q1 = \pi_{SS,NA,BA,AD}(E[SS]C)[AD]A$

At time t the following access conditions are true P1, P4, P5, P6, P7, P10, P11 and P21, while all others are false.

Lets further assume that the tagging algorithm is chosen for computational constraint enforcement. The query is evaluated by:

(1) determine that U belongs to G1 and G2 by consulting a user profile

(2) the franchise of U consists of all 12 authorizations

(3) F(U,AO,Q1) = {AUT1, AUT2, AUT5, AUT6}

(4) F(U,CO,Q1) = {AUT7, AUT8, AUT11, AUT12}

(5) DP = {SS, NA, BA, AD}

(6) all domains in DP are accessible through authorizations in F(U, AO, Q1).

(7) AUT7, AUT8, AUT11 and AUT12 give rights to JOIN Employee, Account and Course with any other relation

(8) F(U,AO,Q1) = { (AUT1, AUT2), (AUT5), (AUT6)}

(9) F(U,CO,Q1) = { (AUT7, AUT8), (AUT11), (AUT12)}

(10) evaluate the effective access condition:

$(P1 \cup P2) \cap (P5) \cap (P6) \cap (P7 \cup P8) \cap (P11) \cap (P12) = \text{TRUE}$

(11) CONC = { CONC1, CONC2, CONC3, CONC4 }

(12) ECON = { CONC1, CONC2}

(13) NA and BA appear together in $\pi(Q1)$ which violates CONC1 the query is therefore rejected.

Suppose now that a later time t' user U requests a list of names, department name and course taught for all instructors.

$Q2 = \pi_{NA,ND,CO}(C[SS]E)[DN]D$.

Suppose that at t' the access conditions have the same value as at time t.

Q2 is then processed as follows:

(1) it is determined that U belongs to G1 and G2

(2) the franchise of U is composed of all twelve authorizations

(5) DP = {NA, BA, DE, CO}

(6) all domains in DP are accessible through authorizations in F(U, AO, Q2)

(7) AUT7, AUT8, AUT9, AUT10 authorize JOINS of Course, E and D to be made with any other relation

(8) F(U,AO,Q1) = { (AUT1, AUT2), (AUT3, AUT4), (AUT6) }

(9) F(U,CO,Q2) = { (AUT7, AUT8), (AUT9, AUT10), (AUT12) }

(10) evaluate effective access condition:

$((P1 \cup P3) \cap (P3 \cup P4) \cap P6 \cap (P7 \cup P8) \cap (P9 \cup P10) \cap P12) = TRUE$

(11) CONC = { CONC1, CONC4)

(12) ECONC = { CONC1 }

(13) Since NA and BA do not appear together in $\pi(Q2)$ CONC1 is not violated and the query is accepted.

The result of Q2 may be stored in the DDB for later use. If U then asked the JOIN of R2 and the relation Account to be made, the constraint would be violated. We will show in the section on authorizations and constraints inheritance how this JOIN will not be allowed.

3.9 COMPUTATIONAL CONSTRAINT EVALUATION.

It should be noted that step 13, which corresponds to the computational constraints evaluation, has been purposefully left abstract. This was done to avoid introducing unecessary implementation details to the algorithm presentation and to the example. We will now study in a little more detail the implementation of step 13 of the algorithm.

To allow for an efficient implementation of the computational constraint evaluation step, the format of the computational constraints is modified before the constraints are stored in the system. Whenever a computational constraint CONC=(A, U, D_1, D_2, S) is imposed, a unique integer N is assigned to the constraint by the system. Two internal constraints are then derived by the system:

CONC1=(A, U, N, S) and
CONC2=(A, U, -N, S).

All authorizations giving access to D_1 are then tagged with CONC1 and all authorizations giving access to D_2 are tagged with CONC2. The system maintains a table of correspondence between the integers assigned and the domains defined in the constraints. A constraint imposer does not need to know the integer assigned to the computational constraint. The table of correspondences acts as an interface between the constraint imposer and the system.

Step 13 of the algorithm can then be replaced by the following three steps:

(13) Partition ECON in two disjoint sets, by having two constraints in the same set if they both have the same sign of tag.

ECONP = $\{CON_i \,/\, TAG(CON_i)>0\}$

ECONN = $\{CON_i \,/\, TAG(CON_i)<0\}$

(14) Order ECONP and ECONN.

(15) Check for all pairs (CON_i, CON_j) with CON_i in ECONP and CON_j in ECONN, if the absolute values of the tags for both constraints are the same. If it is true for at least one pair, then reject the query.

If there are CON_i in ECONP and CON_j in ECONN such that: $|TAG(CON_i)| = |TAG(CON_j)|$ then reject the query.

We can now look at the way the previous example is implemented following the modified constraints enforcement algorithm. The four computational constraints are translated to the following constraints:

CONC1 = (DBA, 1, P21)

CONC2 = (DBA, -1, P21)

CONC3 = (DBA, 2, P22)

CONC4 = (DBA, -2, P22)

The following table of correspondence is also created:

1	Name	Balance
2	SSN	Account

When query Q1 is submitted step 13 of the preceding algorithm is replaced by the following steps:

(13) ECONP = {CONC1} and ECONN = {CONC2}

(14) Both ECONP and ECONN are already ordered.

(15) |TAG(CONC1)| = |TAG(CONC2)| so the query is rejected.

Query Q2 is still accepted since ECONN is empty in this case. The results of query Q2 can be stored in the system for later use. The type of control which needs to be performed over this stored result has to be defined. The next section presents the mechanisms used in the hybrid system to perform this control.

3.9.1 AUTHORIZATIONS AND CONSTRAINTS INHERITANCE

The relation CR resulting from a user's query can be stored in the system for later processing by the user. CR then becomes part of the DDB and new authorizations and constraints have to be associated with CR. The type of control that needs to be performed on CR depends on the control that is effective on the relations used when deriving the CR. The process by which the authorizations and constraints for the relation CR are derived is refered to as the inheritance process. The policy used for the inheritance process is that a user U deriving authorized data with the intent of storing it in the DDBMS obtains a conditioned ownership over the derived data. The user obtains the rights to read, write, update and delete the new relation CR. However, such ownership is conditioned because all the constraints associated with the data used in the derivation of CR are inherited and will take effect whenever CR is used in further computations. User U is also authorized to grant other users READ, WRITE, UPDATE and DELETE rights over CR provided no constraint restricting this transfer was inherited. User U also inherits its rights to JOIN CR depending on the JOIN rights U has for the relations used in the derivation of CR. A JOIN right of relations CR and R_i is granted to U if and only if U has JOIN rights for R_i with all relations used to compute CR. The inheritance process is formalized below, followed by an example.

Let CR = $f(R_1, R_2, \ldots., R_n)$ express that CR is derived through a computation involving relations R_i's for $i=1,\ldots,n$. Then the following authorizations giving U READ, WRITE, UPDATE and DELETE access to CR are created:

$AUT_R(CR)$ = (DBA, U, READ, CR, *, CONSET(CR))

$AUT_W(CR)$ = (DBA, U, WRITE, CR, *, CONSET(CR))

$AUT_U(CR)$ = (DBA, U, UPDATE, CR, *, CONSET(CR))

$AUT_D(CR)$ = (DBA, U, DELETE, CR, *, CONSET(CR)).

The system state predicate in the preceding authorizations is set to "*" to indicate that no state conditions are imposed on the basic access rights U obtains on CR. CONSET(CR) represents the set of constraints applicable to these authorizations. All constraints associated with all R_i's are included in the set CONSET(CR). This includes access, flow and computational constraints. To express formally CONSET(CR), we first have to define the sets of authorizations and constraints associated with the R_i's. The computational and flow constraints are obtained from the tagged authorizations for the R_i's.

Let:

$AAUT(R_i)$ = {$AUT_j(R_i)$/ $AUT_j(R_i)$=($A_j(R_i)$, U, O_j, R_i, S_j, $CCONSET_j(R_i)$) }

be the set of access authorizations for R_i.

$JAUT(R_i)$ = {$AUT_j(R_{i.})$/ $AUT_j(R_i)$=($A_j(R_i)$, U, J, R_i, R_j, S_j, $JCONSET_j(R_i)$) }

be the set of JOIN authorizations for R_i.

$UCCONSET(R_i) = U_j(CCONSET_j(R_i))$

be the set of all computational as well as flow constraints associated with the access authorizations for R_i

$UJCONSET(R_i) = U_j(JCONSET_j(R_i))$

be the set of computational and flow constraints associated with the JOIN authorizations for R_i

$UCFCONSET(R_i) = (UCCONSET(R_i)) \; U \; (UJCONSET(R_i))$

be the set of all computational and flow constraints imposed on U for the use of R_i.

To complete the list of constraints imposed on U for the use of R_i we need to include all access constraints. We define $UACONSET(R_i)$ as being the set of access constraints imposed on U for the use of R_i. This set includes all constraints on access operations as well as on joins. We now define the set of all constraints imposed on U for the use of R_i as being:

$$UCONSET(R_i) = (UCFCONSET(R_i)) \; U \; (UACONSET(R_i))$$

The set of constraints which are then imposed on CR is the union of all the constraints imposed on the various R_i's.

$$CONSET(CR) = U_i(UCONSET(R_i)).$$

The inheritance mechanism is flexible enough to allow the user some freedom in using computed relations, while giving the authorizer the assurance that the user connot gain unauthorized access to the derived data nor transfer the data to other users without the agreement of the original authorizer. The control of the derived data is performed through the constraints. When writing control rules on some data D stored in the DDB, the authorizer has two options: use authorizations or constraints. If the system predicates expressed in the control rules for D do not have to be transmitted to the data derived from D, the authorizer can use either authorizations or constraints since they are complementary. In this case, writing an access authorization or its complementary access constraint achieves the same results for the data being protected. On the other hand, if the system predicates of the control rules for D have to reflected in the rules for the derived data, the authorizer should use constraints. The constraint is inherited in further computations, while the authorization is not. Therefore if an authorizer A needs to impose conditions on the use of the derived data as expressed in a system state predicate, A needs to express the access rights in terms of the constraints. The conditions are then necessarily transmitted along with the computations.

The JOIN authorizations created for relation CR are authorizations derived from the JOIN authorizations which exist for the relations R_i's. A join authorization for the join of CR with relation R_k is created if U has JOIN rights for R_k and R_i for all R_i's. Let:

$UJAUT(R_i) = U_i(JAUT(R_i))$

be the set of JOIN authorizations for U with respect to all R_i's.

It should be noted that the order in which relations R_i and R_j are expressed in $AUT_j(R_i)$ is unimportant. Therefore, we assume that all authorizations are written so that R_i appears as the first relation in the JOIN authorization.

If there is an R_k such that for all R_i there is an authorization AUT_k

$AUT_k(R_i) = (A_k(R_i), U, J, R_i, R_k, S_{ki}, JCONSET_k(R_i))$

create a JOIN authorization for CR of the form:

$AUT(CR) = ((A_j(R_i) \cup (A_u(R_v)), U, J, CR, R_k, AND_i(S_{ki}), CONSET(CR))$.

We will now look at the inheritance process on a small example.

To illustrate the inheritance process we modify the previous example by creating an additional constraint: CONF1 controlling the transfer of information between users in group G2 and users in a third group: G3.

CONF1 = (DBA, Employee, RW, G2, G3, P31)

Authorizations AUT2 and AUT8 are also modified to reflect the introduction of this new flow constraint.

AUT2 = (DBA, G2, RW, Employee, 111, P2, {CONC1, CONF1})

AUT8 = (DBA, G2, J, Employee, *, 111, P8, {CONC1, CONF1})

Suppose that the result of query Q2 is to be stored in the system for further use by U with the name CR1. The following authorizations are created for CR1:

AUT21 = (DBA, U, RWUD, CR1, *, 111, CONSET)

AUT22 = (DBA, U, J, CR1, *, 111, CONSET)

AUT21 gives U the right to read, write, update and delete relation CR1. We condensed the authorizations for the four basic rights in a single authorization. AUT22 gives U the right to JOIN CR1 with every relation in the DDB. This authorization is created because U has the right to JOIN every relation used to derive CR1, i.e. Employee, Course and Department, with every other one in the DDB. The constraints associated with all the authorizations for the relations used in the derivation of CR1 are inherited for all authorizations given to U on CR1

CONSET = {CONC1, CONC4, CONF1}.

Lets suppose further that the user submits a new query Q3 requesting the JOIN of CR1 with the relation Account to derive all the instructors balances. This query is rejected by the system in the same way as query Q1 was rejected because CONC1 which has been inherited would be violated.

Using the inheritance process and the implementation of the computational constraint evaluation previously presented, the authorizers are guaranteed that the limitations they put on the use of their data are respected even when the data is used in further computations. An important property of the implementation of the constraint evaluation is that the constraints cannot be lost during computations since they are not dependent on the names of the relations involved in the constraints. It is not necessary to have the derivation history of computed relations, i.e. the names of all the relations used to derive them, to enforce the constraints. This would have been necessary if the constraints were stored in terms of the names of the relations. It should be noted that although the internal constraints used by the system have to be name independent, the constraints expressed by the authorizers can be name dependent.

The inheritance process is powerful enough to provide authorizers with a tool for an efficient control of the data the user derives from the DDB and stores for future use. In addition, it is flexible enough to allow for different policies to be easily represented. As an illustration of this flexibility we will look at the manner in which different types of policies for the control performed on data supervised by multiple authorizers, can be implemented in the hybrid system.

Suppose that a relation R is under the control of authorizers A1, A2 and A3. There are two ways in which the authorizations and constraints defined by the different authorizers can be interpreted

1. With the first option, all authorizations and constraints defined by the different authorizers on the data apply at any time. This implies that the same response is given to the user for queries on the same data. If A1 grants user U an access O1 to R and A2 grants U access O2 on R, then whenever a constraint on the use of R is imposed by either A1, A2 or A3, it has to be reflected on all authorizations granted to U for R by either authorizer. This is done by tagging all authorizations for the access of R by U with all the constraints imposed by any of the authorizers.

2. The second option is to allow for some inconsistency in the responses given by the DDBMS. This may be necessary for example if R is replicated and controlled by different agencies and one agency does not allow U to obtain R from its data base but does not object to U obtaining it from another source in the network. In this case each authorizer performs a different control over the data. Lets suppose that authorizer A1 grants authorization AUT1 to U over R, and authorizer A2 grants authorization AUT2 to U over R, and that these are the only authorizations granted to U. Lets further suppose that A1 imposes a constraint CON1 on the use of R by U. This constraint will take effect only if U uses AUT1 to access R. Using the

tagging algorithm, only AUT1 will be tagged by the constraint CONT1. In this way the constraints which apply to an access to the data depends on the authorizations used for the access, and therefore on the authorizer which granted the access.

3.10 DECIDABILITY OF THE SAFETY PROBLEM.

The set of tools offered to the authorizer by the hybrid system provide for a solution to the decidability of the safety problem. This problem was introduced by Harrison in (HARRI76) where it was proven that "it is undecidable whether a given configuration of a given protection system is safe for a given generic right". A system is said to be safe if it enables the authorizers to keep their data "under control" in Harrison's words. The safety problem for all protection systems remains an undecidable problem. However, in the hybrid system it is possible to decide if the system is safe or not. By the use of authorizations, tagged constraints and the inheritance process, authorizers can control the dissemination of rights on the data stored in the DDBMS. A user U cannot obtain a certain right O over data D if the authorizer of D did not intend U to obtain the right either explicitly or implicitly. Users are still allowed to transfer rights to each other for data obtained from the system. However, through the constraints and the inheritance mechanism the authorizer controls the types of rights the user will obtain. Lets assume for example, that authorizer A grants a right to a user U1 over data D, but does not want to allow U1 to use D in computations and then transfer some rights over the computed data to user U2. To do so A has only to write a flow constraint expressing that U1 cannot transfer any right over D under any form to U2. The inheritance process assures that the constraint will be transmitted along with any computations performed on D.

It should be noted that the hybrid system is not the first system where the safety decidability problem can be solved. This problem has been solved in the case of non-discretionary systems by using the rigid structure of either a multilevel design or a lattice design (DENNI82). In this case no right can be transfered from one level of the structure to another one. However, the structure remains inflexible in that the rights have to be defined in a rigid framework.

The fact that the safety problem can be satisfactorily decided in the hybrid system is of theoretical as well as practical interest. From the theoretical point of view it is an important result which shows that the safety problem can be easily solved in this specific case. From the practical point of view it is of great interest to the authorizers to know that they can keep a complete control over their data. However, the control over the transfer of derived data between users being mainly controlled by constraints, the authorizer have to anticipate the type of transfers they do not want to allow.

4.0 CHAPTER 4 MODEL

4.1 INTRODUCTION.

We will now integrate the hybrid system proposed in the preceding chapter into a general model for DDBMS protection. To ensure flexibility of design, ease of verification and to accomodate a maximum number of set of policy choices, a modular approach is taken. Building blocks representing basic functions of the components of the model are defined. These building blocks can be organized in different ways depending on the set of policy choices chosen for the DDBMS to be implemented.

We first identify the major components necessary for the modeling of protection in DDBMS. We then define three modeling levels which describe different aspects of the model. The design of the first level model is done following three main design principles. After presenting the building blocks for the first level, we illustrate the construction of the first level by giving the model for a specific set of policy choices. The building blocks of the first level are then translated to another set of building blocks for the second level following two main rules. The example presented for the first level is used to illustrate the two next levels as well. For completeness the models at the different levels for the other sets of policy choices are given in appendix A. We close the chapter by giving the conclusions drawn from the analysis of the models presented.

To model protection in DDBMS, four main components have to be represented: the user interface, the security, the query decomposition and optimization, and the retrieval components.

USER INTERFACE COMPONENT.

This component is responsible for accepting users queries, analyzing and translating them into a suitable form for the security component analysis.

SECURITY COMPONENT.

The security component corresponds to the modelling of the hybrid system presented in the preceding chapter. Two main types of protection rules used to maintain the security of the DDBMS have been identified: authorizations and constraints.

These protection rules can be defined at two different levels: local or global.

1. Local rules.

   These are rules pertaining to data under the jurisdiction of a single organization and not involving operations on two or more relations controlled by authorizers associated with different organizations.

2. Global rules.

   These are rules for regulating the operations on relations owned by authorizers associated with different organizations, or pertaining to replicated data or relations placed under the control of the network Data base administrator.

The protection is ensured by three functions: definition, storage and enforcement of the protection rules.

The security component is further divided into two subcomponents.

1. The access control subcomponent which allows for the definition, storage and enforcement of the protection rules pertaining to the access control of the DDBMS. This includes all the authorizations and constraints defined to control the access (read, write, update) to the data as well as computations performed on the data.

2. The flow control subcomponent allows for the definition, maintenance and enforcement of authorizations and constraints controlling the flow of information between users as well as the flow of information between sites.

QUERY DECOMPOSITION AND OPTIMIZATION.

The query is decomposed into subqueries to be applied at individual sites and the necessary moves of data from site to site are determined according to an optimized strategy.

RETRIEVAL

In response to a query the data is retrieved from the DDB, and the subresponses to the subqueries are obtained from the different sites. This component is also responsible for relocating data in the network for efficiency reasons. If, for example, data D located at site S1 is most often used by users issuing queries at site S2, D may be moved to site S2.

To model the protection in a DDBMS the single site MULTISAFE model has been chosen as a basis. One of the main advantages of the MULTISAFE model is the clear separation between the security function, the user functions and the retrieval function. In MULTISAFE a user cannot submit a query to the DBMS without the security module checking on the user's request. Therefore, one of the main objectives of our model is to separate as much as it is feasible the security function from the other functions and design the model so that no query to the DDBMS can bypass the security function. We assume the communications to be safe, for example by means of encryption. Thus, we are only concerned in this work with the possible security violations which may occur within the DDBMS, namely by queries issued by authenticated users. Our main objective is to separate the user and retrieval functions by all the necessary security functions.

To capture the different aspects of the protection in DDBMS three levels of modeling are defined: conceptual, logical and physical.

1. Conceptual level:

   At this level the security components of the model and the control they perform on each other as well as the logical links between these components are presented. This level of the model allows a user to obtain a conceptual image of the processing of a query by the DDBMS. To the user the distribution of the data in the network is transparent. For the user the DDB appears to be a single centralized data base.

2. Logical level

   The notion of distribution of the data is introduced at this level. The organization of the DDB in a network becomes apparent and the different sites are logically represented. The relationships between the conceptual components of the different sites are defined.

3. Physical level

   The network topology and the manner in which the different components of the logical level are implemented, are shown at this level. The logical paths of communication of the preceding level are mapped into the physical topology of the network. The physical communications constraints between the logical components imposed by the network topology are also visible at this level.

After analyzing the effect of the different policy options on the design of the model, we concluded that the protection system design is influenced by policy decisions taken in only two major areas: local autonomy and flow control.

1. Local autonomy.

   Different levels of local autonomy can be defined, and to each level corresponds a different model design. Individual sites may not have any local autonomy, which may be the case if the organizations participating in the network are completely trusting of each other. An example of such a network is one where departments of an organization share their resources and do not handle very sensitive data. The next degree of local autonomy would be one where the authorizers have control over the enforcement and the location of the control rules but not over the site of the data controlled by these rules. This may be the case if each organization wants to keep some control over the data it owns but is willing to allow the physical movement of the data for efficiency reasons, as long as the owning organization can control the enforcement of the control rules it defines on its own data. The highest degree of autonomy is achieved if all organizations have complete control over the enforcement and location of the protection rules as well as the location of their own data. This degree of autonomy may be required if the organizations sharing data handle very sensitive data and do not trust each other. However, there is a minimum level of trust required from the different organizations if they are to cooperate in a network. It is necessary to define some global rules on data obtained from sets of data under the control of different organizations. A cooperation of the different sites is then necessary or a third entity may be trusted to ensure the definition and enforcement of such rules.

2. Flow of data control.

   Two different choices are possible for this policy: no flow control or total flow control.

a. No flow control.

If the different organizations participating in the network trust each other in addition to trusting the network as an entity, as in the case of no local autonomy previously defined, the overhead incured by data flow control may be avoided. In the cases of partial and total autonomy the organizations may also opt for a no flow control policy. The tradeoff between flow control overhead but tighter security, versus efficiency but lesser protection is in this case resolved in favor of efficiency. The organizations may consider sufficient the control they retain over their data through local rules.

b. Total flow control.

Independently of the amount of autonomy supported by the system, some sites may be mistrusted by others. In this case routing and storage constraints can be defined to control the flow of information in the network.

We will now present the different elements composing the three levels of modeling. We will also illustrate the design criteria used for the different levels by describing a model correspoding to one set of policy choices.

4.2 CONCEPTUAL LEVEL.

Two types of components have to be represented in the conceptual model:

1. security components including the global and local protection rules enforcement as well as the flow control rules enforcement,

2. non security components which are the user functions, the query decomposition and optimization, and the storage and retrieval.

A transaction issued by a DDBMS user is first analyzed by the user functions and transmitted to the query decomposition and optimization function which in turn requests the data from the storage and retrieval function. The transmission of the transaction between the different functions is controlled by the security rules at each level. The global rules control the transfer of the transaction from the user functions to the query decomposition and optimization function. The flow of information between the query decomposition and optimization and the storage and retrieval functions initiated on behalf of the transaction are controlled by local and flow control rules. The local rules or the flow control rules may be present in the model or absent from it depending on the policy choices made. The general conceptual model is given in figure 4. In this figure the controls that the different protection rules perform are represented by small circles. The security components of the model are represented using a double box.

Three basic properties are used for the design of the conceptual model:

1. Self containment of the security components.

   The security components of the system should be complete and should not have to communicate with any other non secure module to obtain information to render a security judgement.

2. Minimal security kernel.

   The security components should contain only security functions. The size of the security component should be reduced to the minimum so as to ensure the reliability and security of the protection kernel. In this case only the security functions have to be trusted and proven correct.

3. Layers of abstraction.

   Two layers of abstraction are defined in the model. At the higher level the physical distribution of the

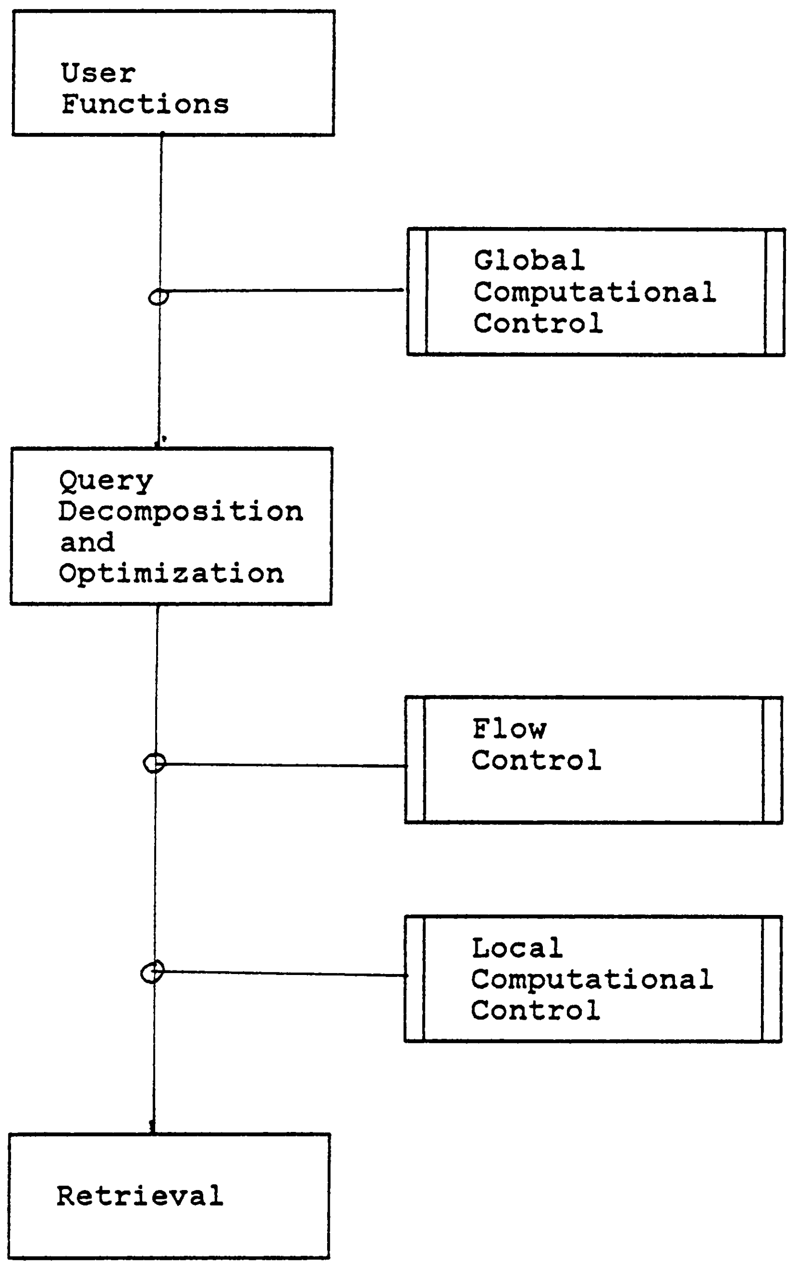

User Functions
Global Computational Control
Query Decomposition and Optimization
Flow Control
Local Computational Control
Retrieval

Figure 4. General Conceptual Model

data is unknown while at the lower level the data is known to be distributed. The highest layer would include the user functions as well as the global rules, while the lowest one would include the query decomposition and optimization, the flow control and storage and retrieval functions.

These three properties cannot always be satisfied simultaneously. Depending on the policy choices made, one of the properties may have to be compromised so as to satisfy another one.

When no flow control is performed and the organizations are allowed to retain a total autonomy, the three properties can be satisfied. The local autonomy policy does not affect the interaction of the first two properties. However, if flow control is introduced the security component cannot be both minimal and self contained at the same time. The query decomposition and optimization functions are closely related to the flow control functions. In MULTISAFE the security function can be clearly separated from any other function and the two properties can always be satisfied simultaneously. However, because of the nature of the new functions introduced in the distributed case, it is not possible anymore to maintain this clear division of functions. In MULTISAFE the query is either rejected or accepted by the security module. In the distributed case a particular decomposition of the query may be rejected because it does not satisfy the flow control constraints but the query itself is not rejected if it can be decomposed in another manner. The query decomposition and optimization has therefore to be made in accordance with the flow control constraints. If the security component has to be self contained, the query decomposition and optimization functions as well as the Retrieval Optimizer function have to be associated with the flow control functions. In this case the enforcement of the flow control constraints is made at the same time as the query decomposition and optimization is done or the data moves are initiated. If on the other hand it is preferable to have a minimal kernel, the query decomposition and optimization and Retrieval Optimizer functions have to be separate from the flow control functions, and the enforcement has to be done after the query decomposition and optimization and after the data moves are initiated.

The two last properties cannot be concurrently satisfied if flow control is performed and the organizations do not have any autonomy or only partial autonomy. In these models the Retrieval Optimizer (ROM) can either be integrated with the flow control functions or it can be placed at the level of the user functions and considered as a user of the DDBMS submitting requests to access information and to move data. If the Retrieval Optimizer is integrated the layers of abstraction property is satisfied since the distribution of the data does not have to be known before the lowest layer at the level of the flow control. However, the security kernel is not maintained minimal since the Retrieval Optimizer is an integrated part of the flow control functions. If the Retrieval Optimizer is placed at the user functions level, the security kernel is kept minimal, but the distribution of the data has to be known at the highest layer.

Building blocks are defined to facilitate the construction and comparison of the models for the different policy choices. Each building block includes modules used to implement the different functions of the secure DDBMS. These blocks are graphically represented in figures 5 to 11. Mnemonics are used to represent the blocks using abbreviations for the modules composing the blocks. A table of correspondence between these mnemonics and the full name of the modules and blocks is given in appendix B.

BLOCK C(UAM-QTM).

This block contains the User Access Module (UAM) and the Query Translation Module (QTM). The User Access Module is similar to the User Access Module defined in Multisafe. The Query Translation Module translates the queries from the query language to a relational algebra form. As we have seen in the previous chapter to simplify the enforcement process the query is translated so that all joins are first expressed, followed by all projections and restrictions.

BLOCK C(SRM-DDB).

This block contains the Storage and Retrieval Module (SRM) and the Distributed Data Base (DDB). The Storage and Retrieval Module retrieves or updates data from the DDB in response to user's queries.

BLOCK C(QDOM).

The Query Decomposition and Optimization Module (QDOM) decomposes the query into subqueries to be processed at different sites. To make the routing decisions the Query Decomposition and Optimization module module uses information derived from the Directory of the DDB and from the Network Topology and status data base. The query decomposition and optimization can be made so as to minimize the communications costs, access costs as well as computation costs. In R*, for example the critical resource is assumed to be the communication lines and the optimization process is performed so that the communications between sites are reduced to a minimum.

BLOCK C(GRM).

The global rules compose this block. The authorizations constraints mechanism defined in the hybrid system chapter is used for the global rules. Authorizations and constraints are self regulating in addition to controlling the transactions to all other data bases.

BLOCK C(LRM)

This block is identical to the preceding one but is used for local rules.

BLOCK C(QDOM-TRCM-QH).

This block contains the same module and data bases as block C(QDOM) but in addition has Temporary Routing Constraints rules (TRC) and a Query History (QH) data base. The Temporary Routing Constraints rules control the information flow between sites. The decomposition performed by the Query Decomposition and Optimization module is subject to the Temporary Routing Constraints rules. The query history data base is used to determine the pattern of use of the data to move the data near to the sites where it is mostly used for efficiency reasons. All the data bases used in this block can be updated by a transaction issued from block C(UAM-QTM) after having been subject to the control of the appropriate rules.

BLOCK C(SCM)

This block consists of only the Storage Constraints Module rules (SCM). The Storage Constraints Module rules control the permanent storage of data at certain sites.

BLOCK C(ROM).

The Retrieval Optimizer module (ROM) is the module responsible for the moves of the data between the different sites for efficiency reasons. This block is used in the case where the Retrieval Optimizer is placed in the model in the same position as a UAM.

<u>BLOCK C(ROM-SRM-SCM).</u>

The integrated Retrieval Optimizer and its interaction with the Storage and Retrieval Module are presented in this block. The Query History data base is consulted by the Retrieval Optimizer to make decisions about data moves. The Retrieval Optimizer requests information from the Storage and Retrieval Module and initiate data moves by accessing directly the data bases at the different sites. These moves are controlled by the Storage Constraints rules.

<u>BLOCK C(PP-GRM-LRM).</u>

This block is used for the construction of the partial local autonomy model. A transaction issued from the User Access Module is first preprocessed to determine if it is a local or global transaction. If it involves only data that is considered to be local, i.e. controlled by the organization of the user, it is then subject only to the local rules. The local rules do not have to be physically stored at the site of the user. If the query requests some global data, it is first controlled by global rules. It is then controlled by the respective local rules associated with the data in the query pertaining to different organizations.

<u>BLOCK C(SRM-QH).</u>

In this block, in addition to the module and data base defined in block C(SRM-DDB), the Query History Data Base is represented. It is queried by the Storage and Retrieval Module to respond to requests from the Retrieval Optimizer, when the Retrieval Optimizer is placed in the model as a user.

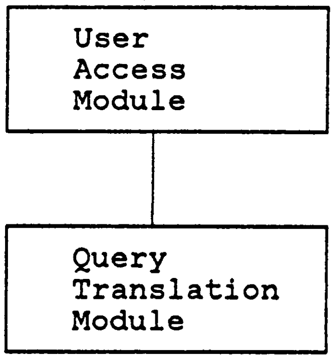
<u>BLOCK C(ROM-SRM).</u>

Block C(ROM-SRM) is similar to block C(ROM-SRM-SCM) but is used when no flow control is performed and does not therefore contain any Storage Constraints module.
1

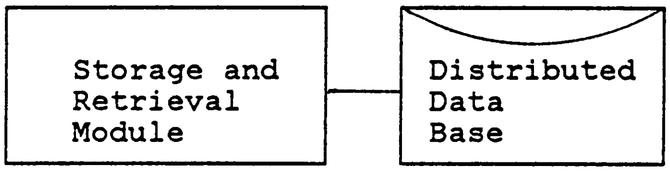
The conceptual blocks can be used to build the models corresponding to the different sets of policy choices. By combining the different choices for the policies of trust between sites we obtain six cases:

1. No autonomy and no flow control.
2. No autonomy and flow control.
3. Total autonomy and flow control.
4. Total autonomy and no flow control.
5. Partial autonomy and no flow control.
6. Partial autonomy and flow control.

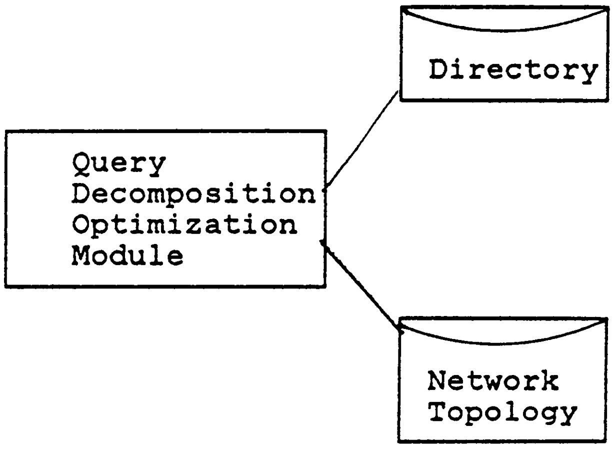
To illustrate the use of the conceptual blocks in the design of the conceptual models we will present the case of partial autonomy and flow control. The corresponding conceptual model is given in figures 12-13.



Block C(UAM-QTM)

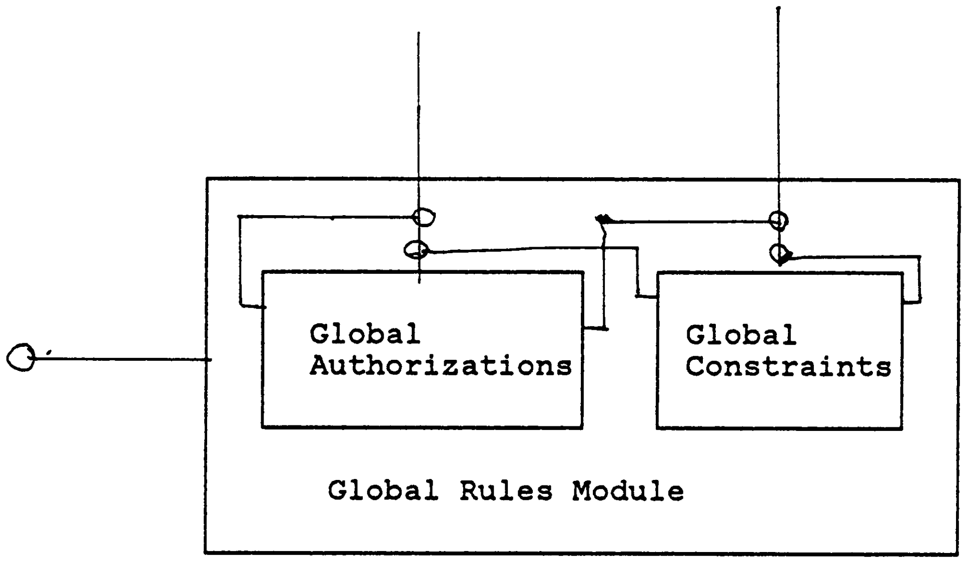


Block C(SRM-DDB)

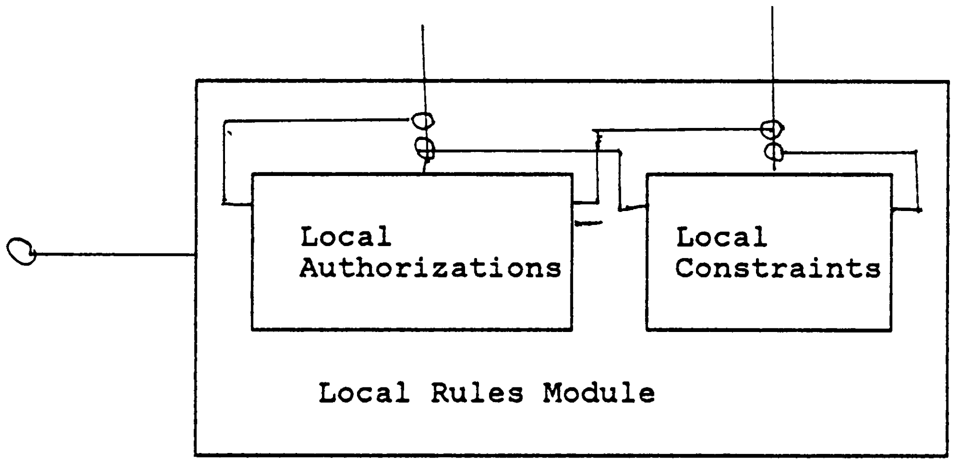


Block C(QDOM)

Figure 5. Conceptual Blocks 1.



Block C(GRM)



Block C(LRM)

Figure 6. Conceptual Blocks 2

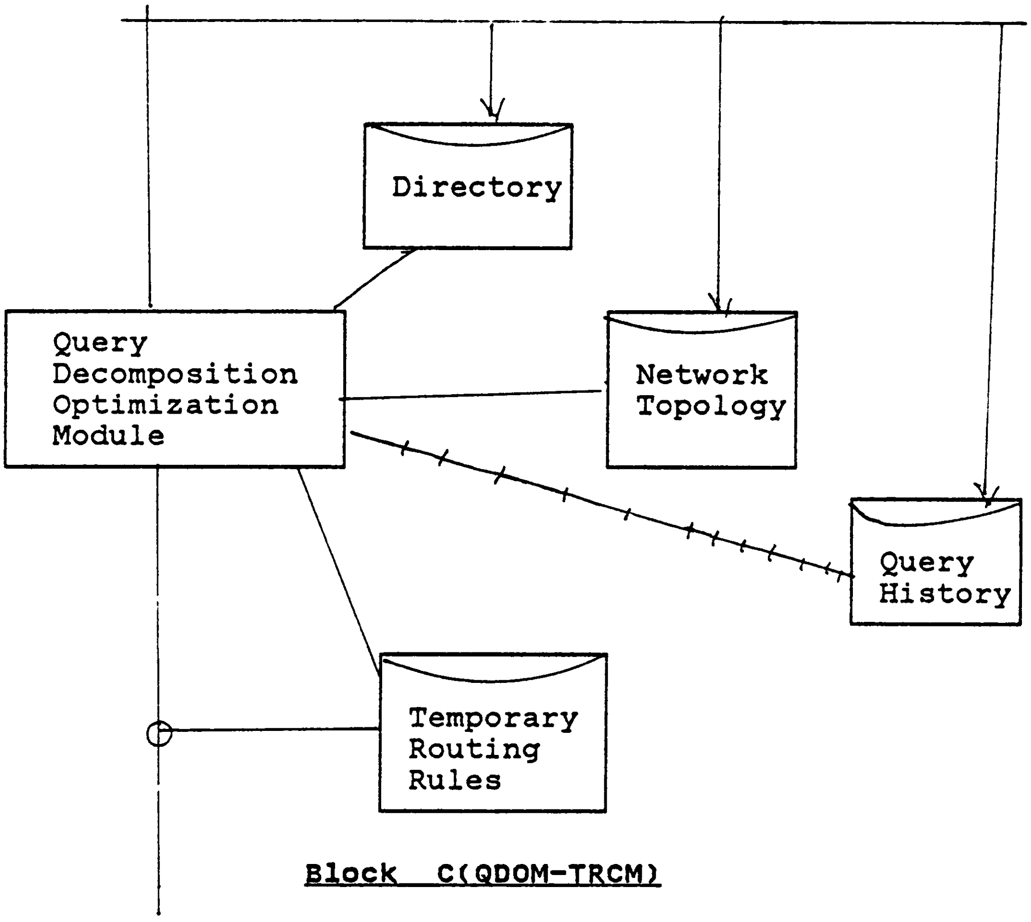
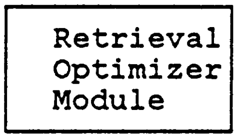


Figure 7. Conceptual Blocks 3

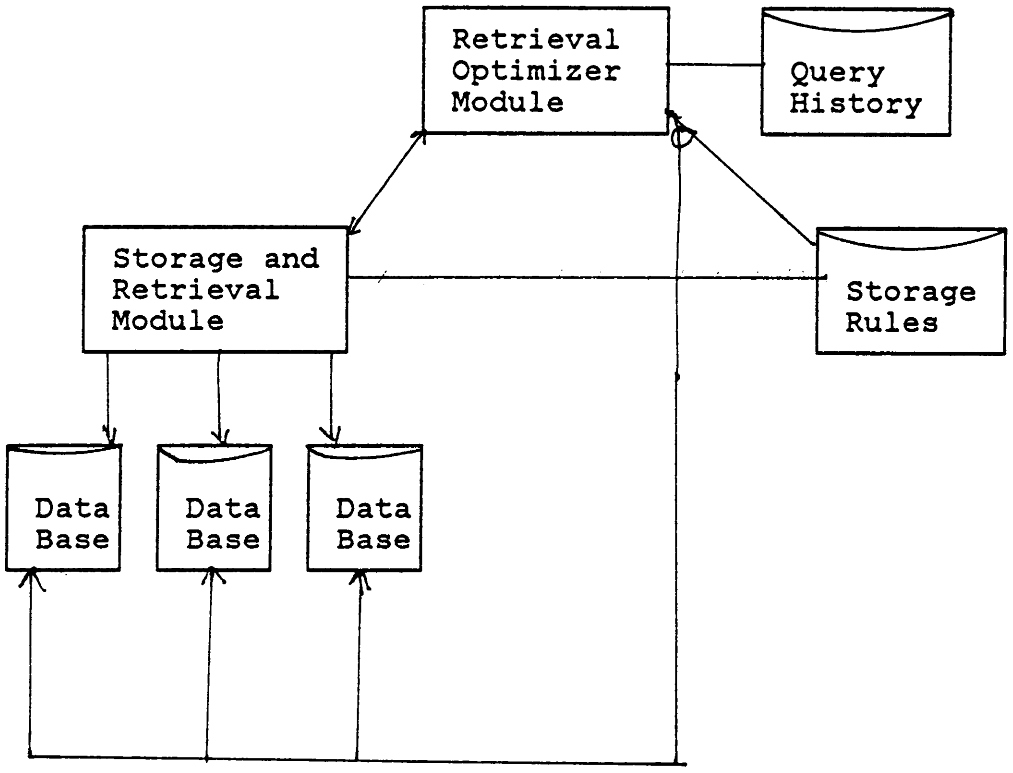
Storage
Rules

<u>**Block C(SCM)**</u>



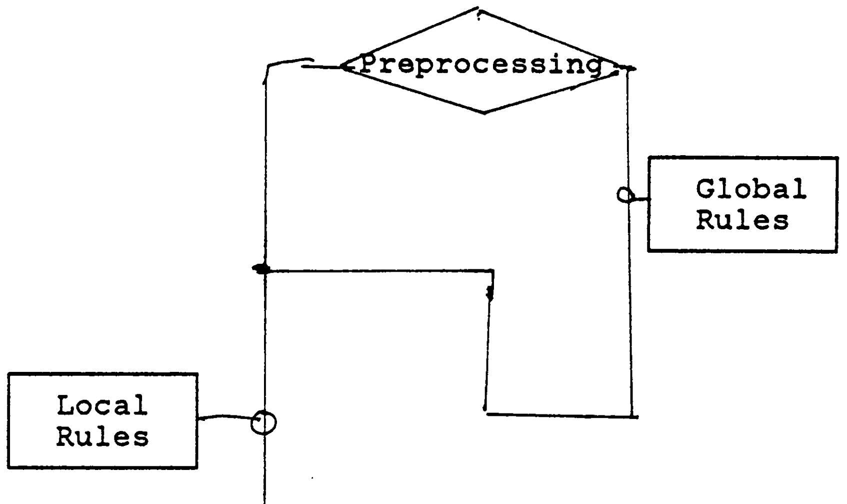
<u>**Block C(ROM)**</u>

Figure 8. Conceptual Blocks 4

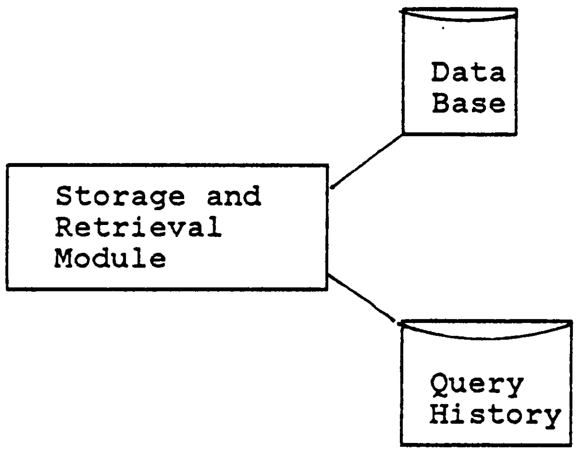


__Block C(ROM-SRM-SCM)__

Figure 9. Conceptual Blocks 5

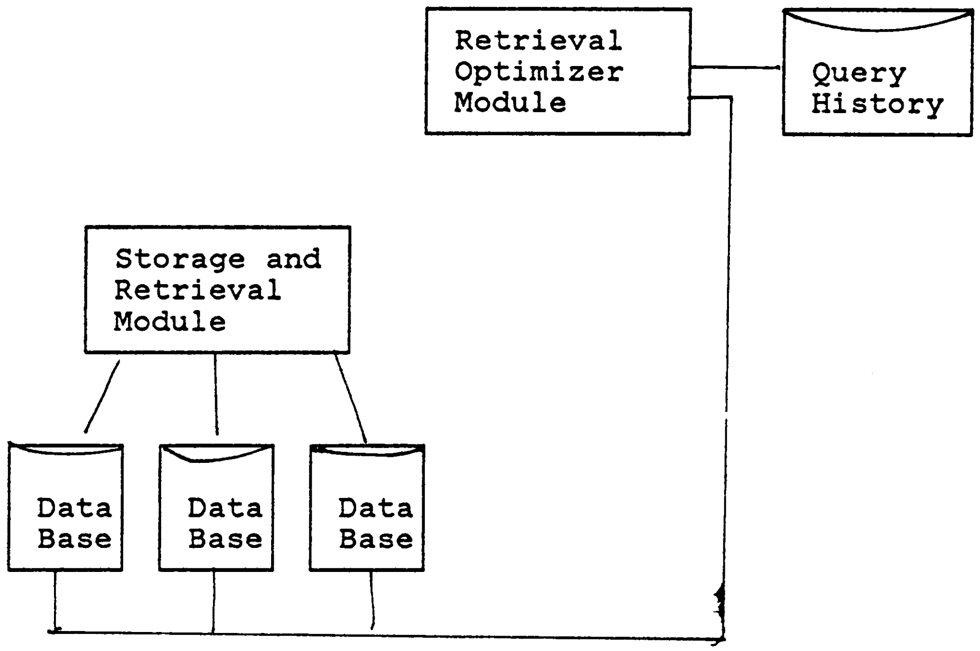


Block C(PP-GRM-LRM)



Block C(SRM-QH)

Figure 10. Conceptual Blocks 6



Block C(ROM-SRM)

Figure 11. Conceptual Blocks 7

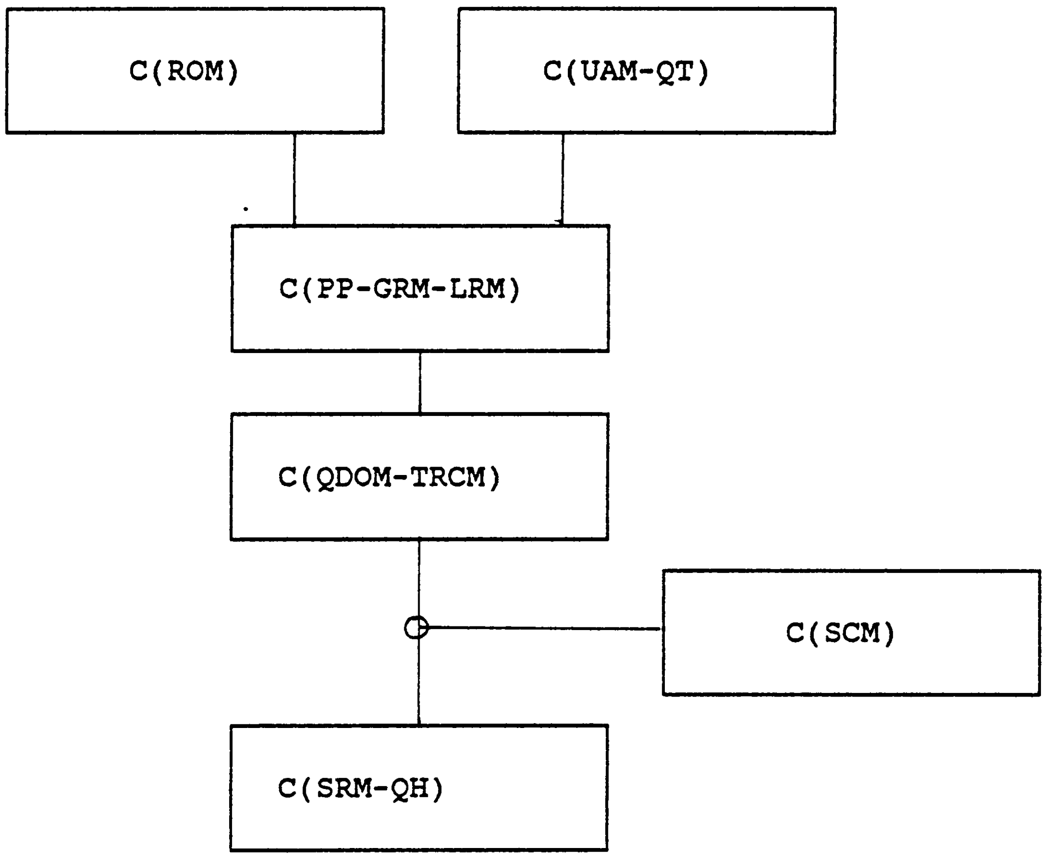


Figure 12. Conceptual Model PA, FC ROM as User.: Partial Autonomy, Flow Control Retrieval Optimizer as a User.

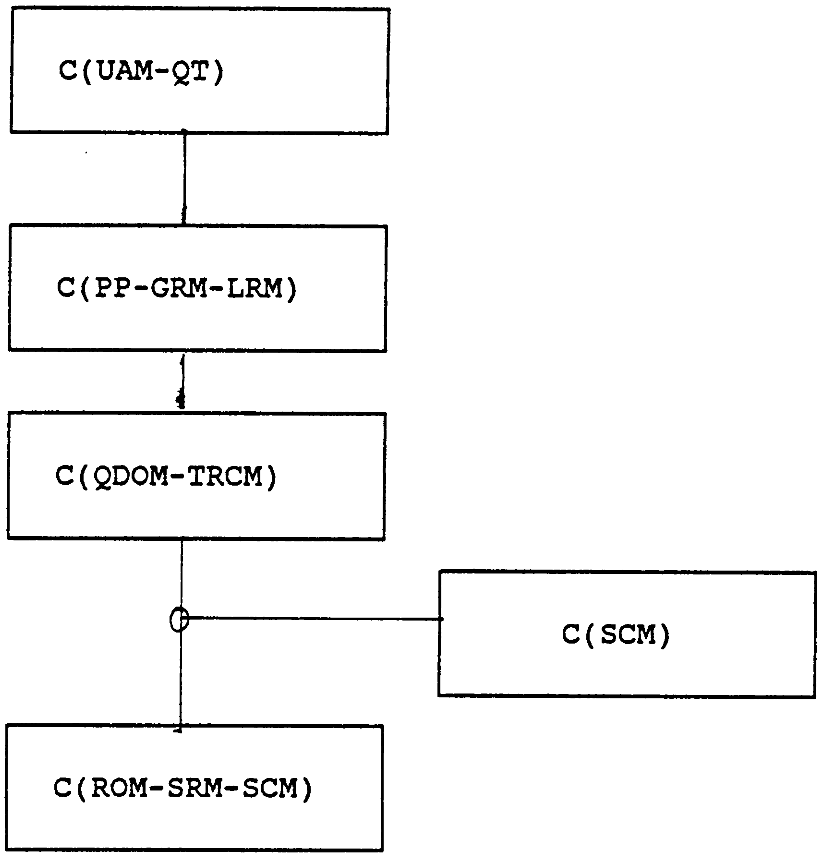


Figure 13. Conceptual Model PA, FC ROM Integrated: Partial Autonomy, Flow Control, Retrieval Optimizer Integrated.

The other conceptual models are given in appendix 1. In partial autonomy the organizations retain control over the definition and enforcement of the rules over their data. However, they do not control the location of data. The Retrieval Optimizer is the module responsible for the relocation of data in the network for efficiency reasons. Two cases have to be considered depending on where the Retrieval Optimizer is located:

1. If the Retrieval Optimizer is placed as a user the Storage and Retrieval Module is then responsible for accessing the Query History data base as well as moving the data in response to transactions initiated by the Retrieval Optimizer.

2. If the Retrieval Optimizer is integrated the Query History data base querying and the data moves are directly performed by the Retrieval Optimizer.

The models for these two cases use the same conceptual blocks except for the blocks containing the SRM and the ROM. When the Retrieval Optimizer is placed as a user, blocks ROM and SRM-QH are used. When the Retrieval optimizer is integrated, block SRM-QH is replaced by block ROM-SRM-SCM. In both cases a transaction issued by the User Access Module is first checked against the global and local rules. A preprocessing is performed to determine if the transaction is purely local, in which case it is not checked against the global rules, or if it is global, in which case it has to be checked against both global and local rules. It should be noted that the local rules are checked before the Query Decomposition and Optimization is performed, since the local rules enforcement is independent of the location of the data. The Query Decomposition and Optimization is performed on the transaction once the global and local checks have been completed. The transaction is then transmitted to the Storage and Retrieval Module for processing. The Query Decomposition and Optimization as well as the data moves are controlled by the flow control rules, i.e. the Temporary Routing and Storage Constraints.

4.3 LOGICAL LEVEL.

To visualize the logical relationships between the conceptual modules two logical sites are represented at this level. These logical sites do not necessarily correspond to physical sites. The distinction between logical sites is that a physical site need hold only a subset of the modules defined for a logical site.

The conceptual blocks are translated into logical blocks by applying the following rules:

1. A conceptual module of the form given in figure 14 A is translated in the form given in figure 14 B. if the data base accessed by the conceptual module is global, i.e. shared by modules of the same type at other sites. The global data base is then used as a link between the two logical modules of the same type. The synchronization is thus realized through the common data base. If the accessed data base is purely local each logical module accesses its own data base as shown in figure 14 C.

2. There are two different ways in which control rules can be translated as illustrated in figure 15.

   a. Distribution of functions.

      The control enforcement and the processing are realized simultaneously whenever the control rules are used to verify if a transaction satisfies the rules after having been analyzed by a processing module. In this way transactions violating the rules cannot be generated from the processing module since the enforcement of the rules and the processing are integrated. If the rules control more than one processing the enforcement is replicated with all the processing modules, and the enforcement function is therefore

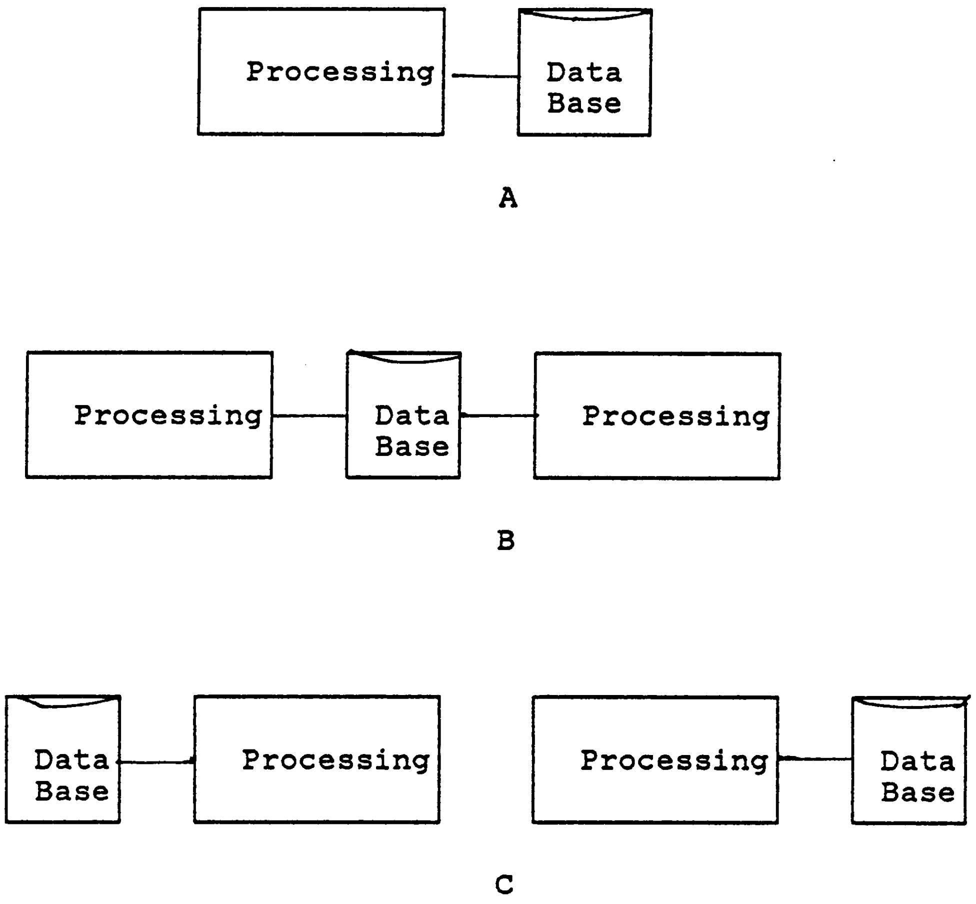
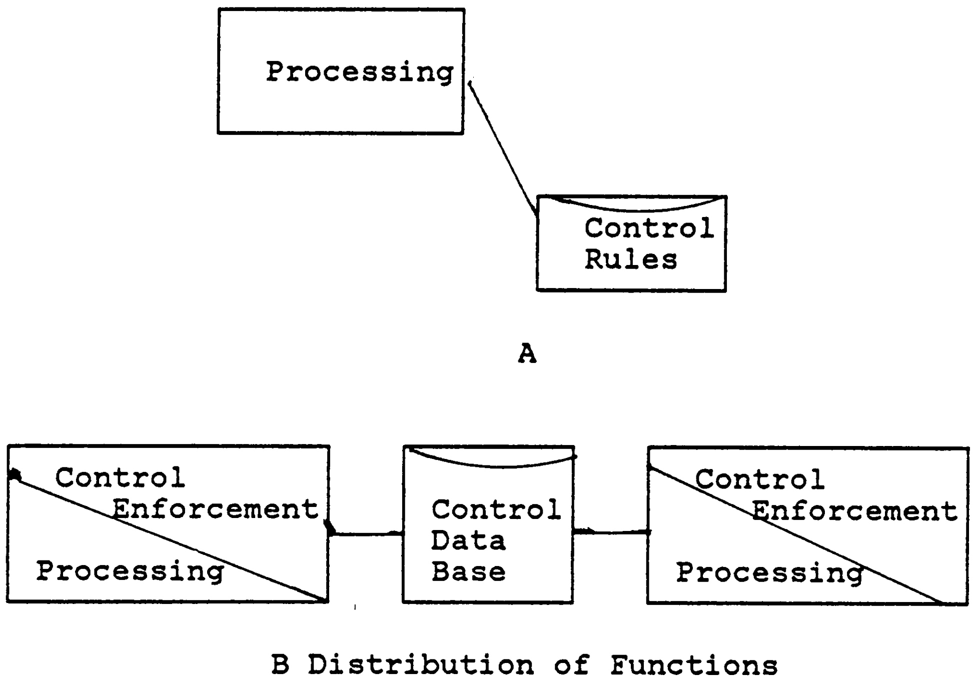


Figure 14. Conceptual to Logical Translation Rule 1



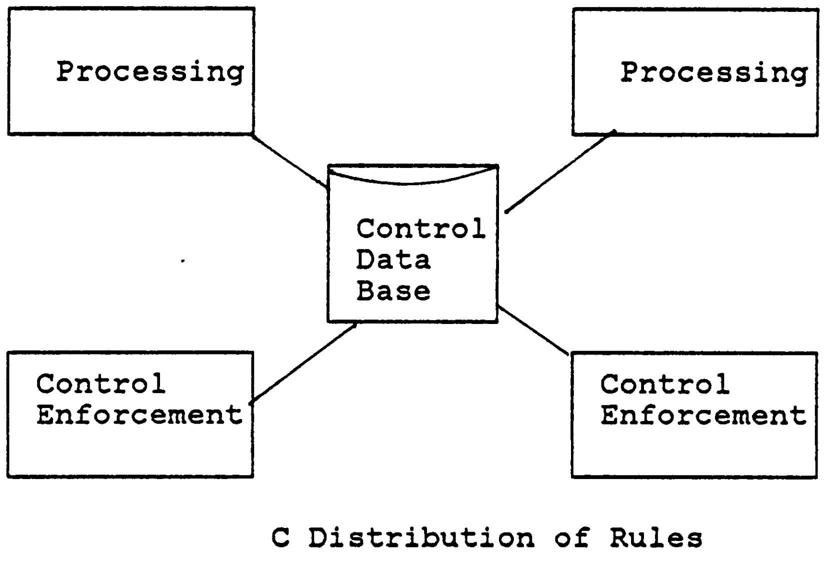


Figure 15. Conceptual to Logical Translation Rule 2

distributed. The processing-enforcement module is a security module and the rules checking are done within the module. The processing is always done accordingly to the control rules and the data base of control rules is accessed by all the integrated enforcement-processing modules. This data base is used for synchronization and communication between the integrated modules at the different logical sites.

It should be noted that in the case of distribution of functions the minimal security kernel property is violated since the security function is always associated with a processing function. However, the self containment property is satisfied since all rule checkings are done within the security module.

An example of an enforcement-processing module is the Query Decomposition and Optimization and the Temporary Routing Constraints enforcement (QDOM and TRCM). In a QDO-TRCM module subqueries are generated which satisfy the Temporary Routing Constraints. The decomposition is optimal given the constraints. A better decomposition could possibly be generated if the constraints did not have to be satisfied.

b. Distribution of rules.

When rules are distributed the processing and the enforcement are separated. The Control Rules data base is accessed by the processing as well as by the control enforcement. The processing module uses the control rules to make decisions which comply with the rules. However, the processing module is not a security module and the validity of the processing module decisions have to be checked by the control enforcement module. Unlike in the distribution of functions, violating transactions can be generated from the processing module, in which case the transaction is returned to the processing module by the enforcement module. As with distribution of functions the data base of control rules is the synchronization link between the processing and control enforcement at the different logical sites.

In the case of distribution of rules the security functions are clearly separated from the processing functions and the minimal security kernel property is satisfied.

The translation of conceptual blocks to logical ones is not always done on a one to one basis. Some conceptual blocks may be linked together in one logical block to better represent the interaction between rules enforcement and certain processing modules. A conceptual block may also be divided in two logical blocks so as to separate elementary functions which could be separately implemented.

The logical blocks are represented in figures 16 to 24.

BLOCK L(UAM-QTM).

Block L(UAM-QTM) is identical to block C(UAM-QTM).

BLOCK L(SRM-DDB).

Block L(SRM-DDB) is identical to block C(SRM-DDB).

BLOCK L(QDOM).

Block C(QDOM) is translated to block L(QDOM) following rule 1. Two Query Decomposition and Optimization Modules are represented. These modules access a common directory and network topology data base through which the synchronization between the two modules is realized.

BLOCK L(GRM).

The global authorizations and constraints are enforced by the Global Computational Control Enforcement module. The enforcement modules access the common data base of global rules, which is also used for synchronization between the enforcement modules.

BLOCK L(LRM).

The local authorizations and constraints are enforced by Local Computational Control Enforcement modules (LCCE). Each LCCE module accesses its own local rules data base.

BLOCK L(QDOM-TRCM-SCM).

The Query Decomposition and Optimization, Temporary Routing Constraints and Storage Constraints functions are grouped together in block L(QDOM-TRCM-SCM). The Query Decomposition and Optimization processing function and the two control functions Temporary Routing Constraints and Storage Constraints are translated following rule 2. Two cases have therefore to be considered:

1. Distribution of functions.

   The three functions, Query Decomposition and Optimization, Temporary Routing Constraints and Storage Constraints are combined in one module. A query transmitted to block L(QDOM-TRCM-SCM) is decomposed into subqueries satisfying the flow control rules. The directory and the two flow control rules data bases are accessed by all Query Decomposition and Optimization-Temporary Routing-Storage Constraints modules at the different logical sites and provide for a means of synchronization between the modules. The Query Decomposition and Optimization-Temporary Routing-Storage Constraints Module is a security module and is assumed to function properly. In this case the minimum security kernel property is not satisfied, the Query Decomposition and Optimization module being a non security function included in a security module. However, the integration of the different functions eliminates the redundant rules checking.

2. Distribution of rules.

   Each function is represented in a different module. The Query Decomposition and Optimization module function is first performed followed by the control checks done by the Temporary Routing Constraints and Storage Constraints Modules. Although the decomposition is made with the knowledge of the control rules, a double check is performed by the Temporary Routing Constraints and Storage Constraints modules and the Query Decomposition and Optimization module is therefore not assumed to function properly.

<u>BLOCK L(ROM).</u>

This block is represented in the same way as block C(ROM).

<u>BLOCK L(ROM-SRM-SCM).</u>

Block L(ROM-SRM-SCM) is divided in two blocks for convenience.

<u>BLOCK L(ROM-SCM).</u>

The Retrieval Optimizer and the Storage Constraints rules are represented in this block. Following rule 2 to translate the conceptual blocks into logical ones, two cases have to be analyzed.

1. Distribution of functions.

   The Retrieval Optimizer is combined with the Storage Constraints enforcement function in one module. The data moves initiated by the Retrieval Optimizer comply with the storage constraints rules because the Retrieval Optimizer and storage constraints functions are integrated. As with the Query Decomposition and Optimization-Temporary Routing-Storage Constraints module the minimum kernel property is not respected but the data moves and enforcement of the storage constraints are done in a more efficient manner than when the rules are distributed.

   The ROM-SCM module consults the Directory, Query History and the Storage Constraints data bases to perform the data move and storage constraints enforcement functions.

2. Distribution of rules.

   When the Retrieval Optimizer and the Storage Constraints enforcement functions are separated the data moves are initiated after consulting the Directory, Storage Constraints and Query History data bases. The data moves have to be accepted by the Storage Constraints enforcement module before being effective. The minimum kernel property is in this case respected but the security checks are done twice: once by the Retrieval Optimizer and a second time by the Storage Constraints enforcement module.

BLOCK L(SRM).

This block contains the Storage and Retrieval module which accesses the Data Base.

BLOCK L(PP-GRM-LRM).

This block is a security block and rule 1 is used to translate the conceptual block to a logical one. The global computational control is performed by the Global Computational Control Enforcement modules which communicate through the global rules data base. The local computational control is done by the Local Computational Control Enforcement. Each Local Computational Control Enforcement accesses directly its own data base of local rules, but the Local Computational Control Enforcement at the different logical sites may communicate. The organizations retain control over the enforcement and localization of the local rules over their data. However, an organization may decide to store the rules at another organization's site but still retain the enforcement. In this case, the two organizations need to communicate to exchange their local rules.

<u>BLOCK L(SRM-QH).</u>

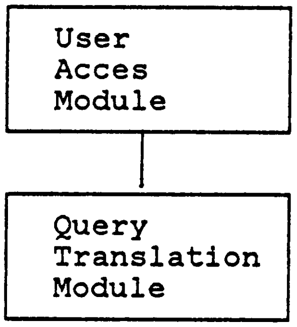
Block C(SRM-QH) is translated using rule 1 and each Storage and Retrieval Module accesses its local Data Base and the Query History data base. The Storage and Retrieval Modules communicate for synchronization purposes.

<u>BLOCK L(ROM-SRM).</u>

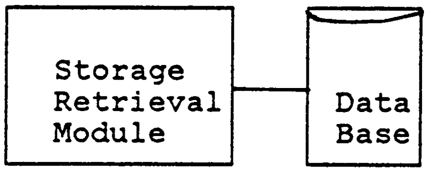
The Retrieval Optimizer modules relocate the data by a direct access to the Data Base, and by communicating across logical sites. The Query History data base is also a common data base and is used by all Retrieval Optimizer modules. The Storage and Retrieval modules communicate for synchronization.

As an illustration of the translation of the conceptual models to logical ones using the logical blocks defined we will present the logical model corresponding to the conceptual model described in the preceding section, i.e. partial autonomy and flow control. The other models are constructed similarly following the same rules.

The Retrieval Optimizer can be either integrated or be placed at the users level. The two cases are represented in figures 25-26. Blocks C(UAM-QTM), C(ROM) and C(SRM-QH) are translated by blocks L(UAM-QTM), L(ROM) and L(SRM-QH) at each logical site. Blocks C(QDOM-TRCM-QH) and C(SCM) are translated to block L(QDOM-TRCM-SCM) which covers the two logical sites. Block L(QDOM-TRCM-SCM) can be used for either distribution of functions or for distribution of

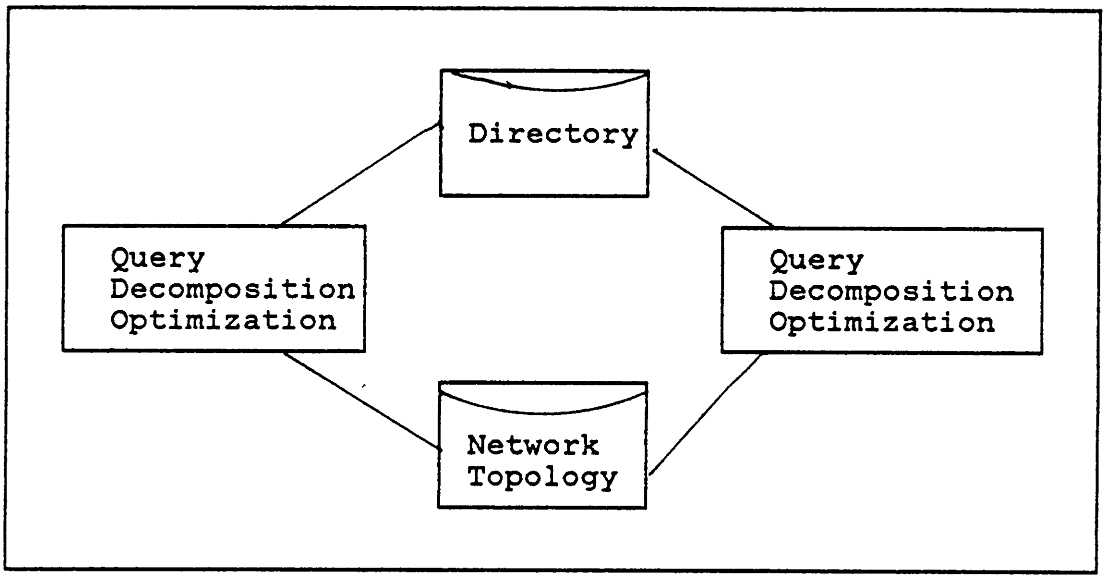


Block L(UAM-QTM)



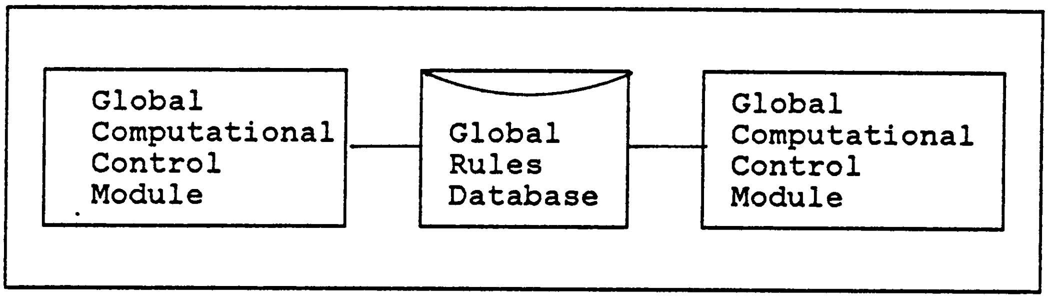
Block L(SRM-DDB)

Figure 16. Logical Blocks 1

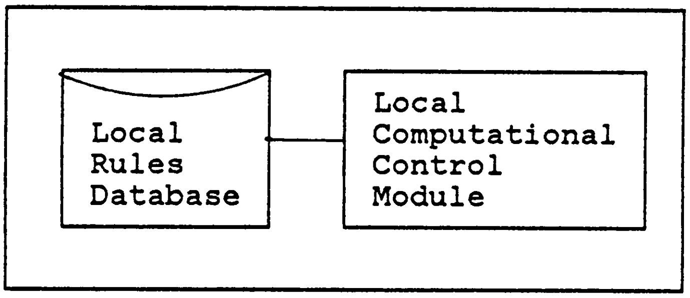

Directory
Query Decomposition Optimization
Query Decomposition Optimization
Network Topology

<u>**Block L(QDOM)**</u>

Figure 17. Logical Blocks 2

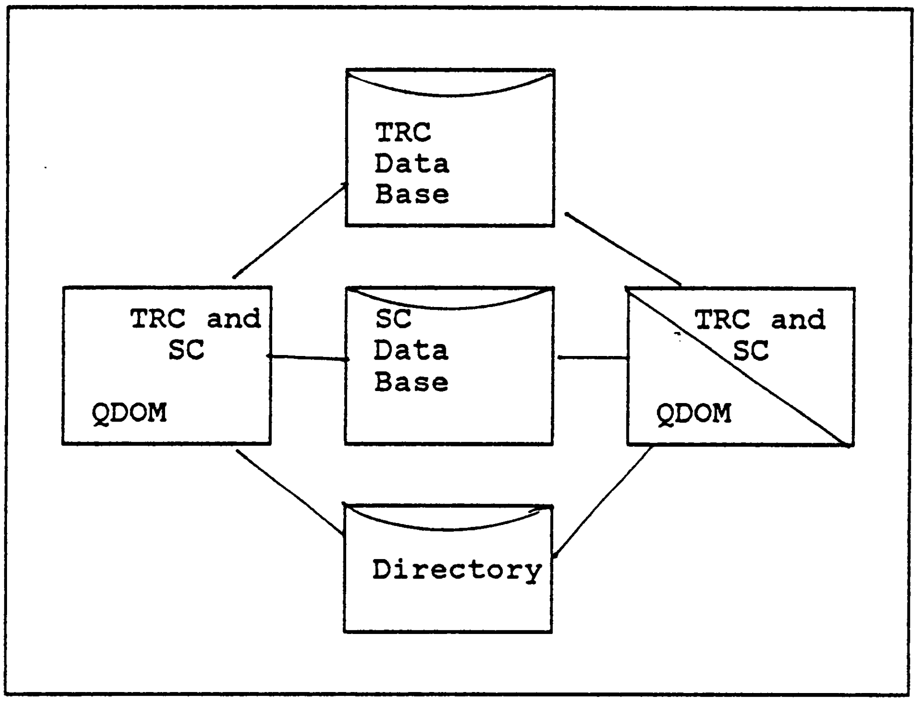


Block L(GRM)



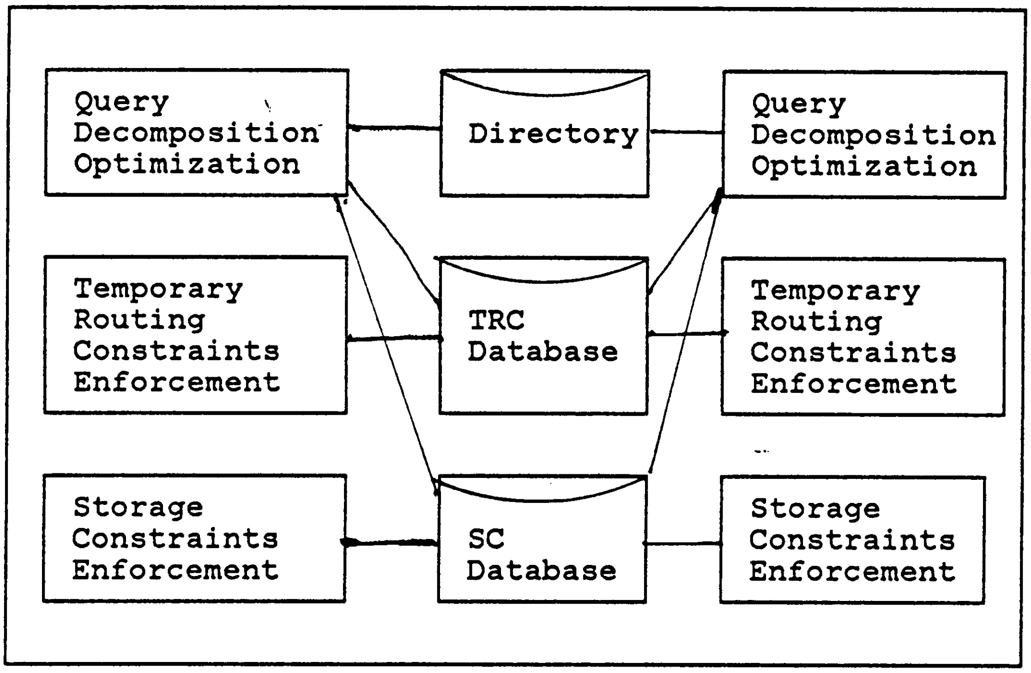
Block L(LRM)

Figure 18. Logical Blocks 3



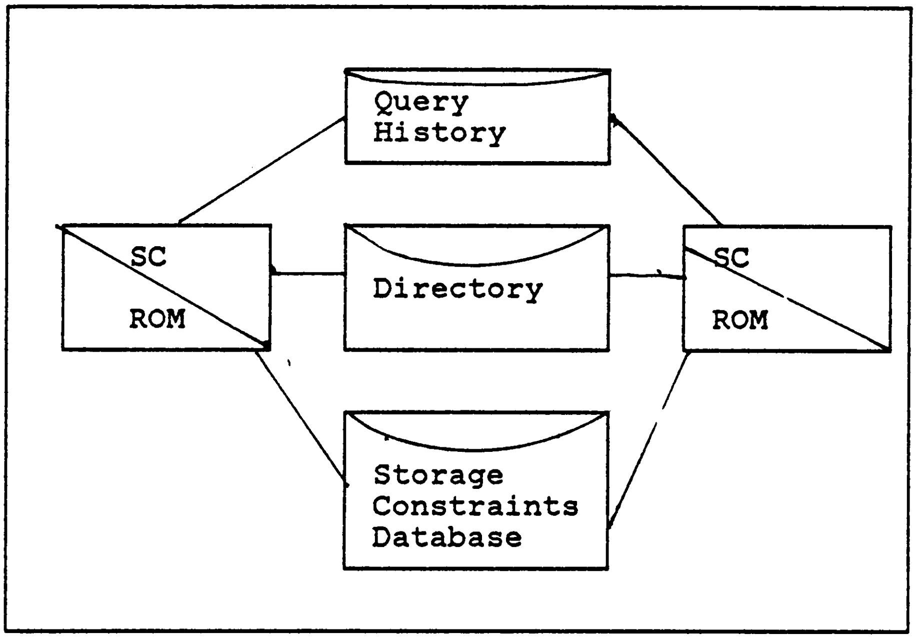
Block L(QDOM-TRCM-SCM) Distribution of Functions

Figure 19. Logical Blocks 4


Query Decomposition Optimization
Directory
Query Decomposition Optimization
Temporary Routing Constraints Enforcement
TRC Database
Temporary Routing Constraints Enforcement
Storage Constraints Enforcement
SC Database
Storage Constraints Enforcement

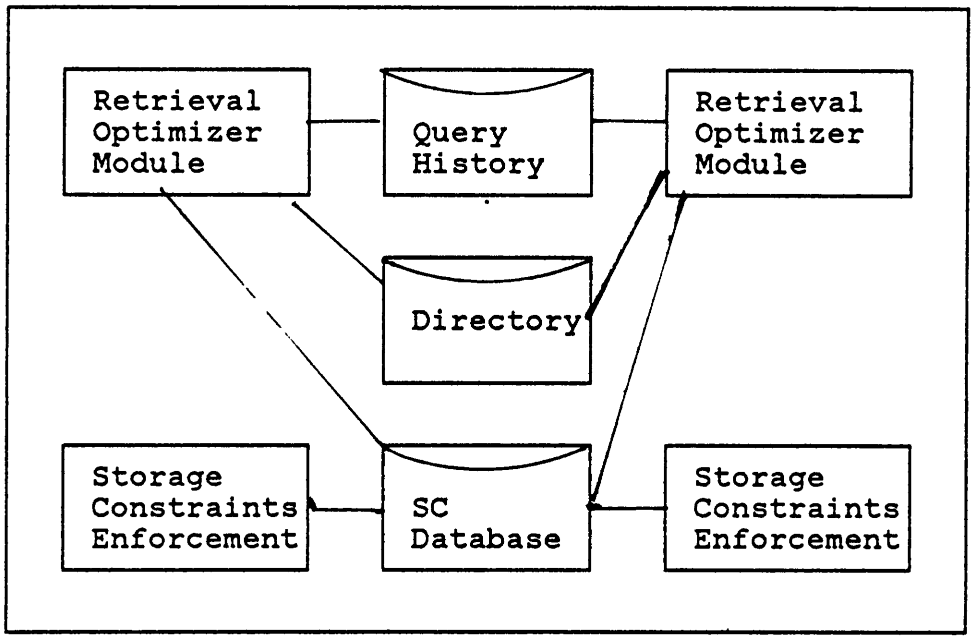
<u>Block L(QDOM-TRCM-SCM)</u> Distribution of Rules

Figure 20. Logical Blocks 5



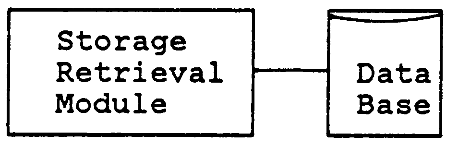
Block L(ROM-SCM) Distribution of Functions

Figure 21. Logical Blocks 6

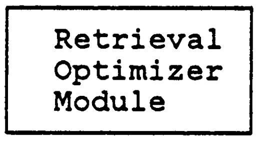

Retrieval Optimizer Module
Query History
Retrieval Optimizer Module
Directory
Storage Constraints Enforcement
SC Database
Storage Constraints Enforcement

<u>Block L(ROM-SCM)</u> Distribution of Rules

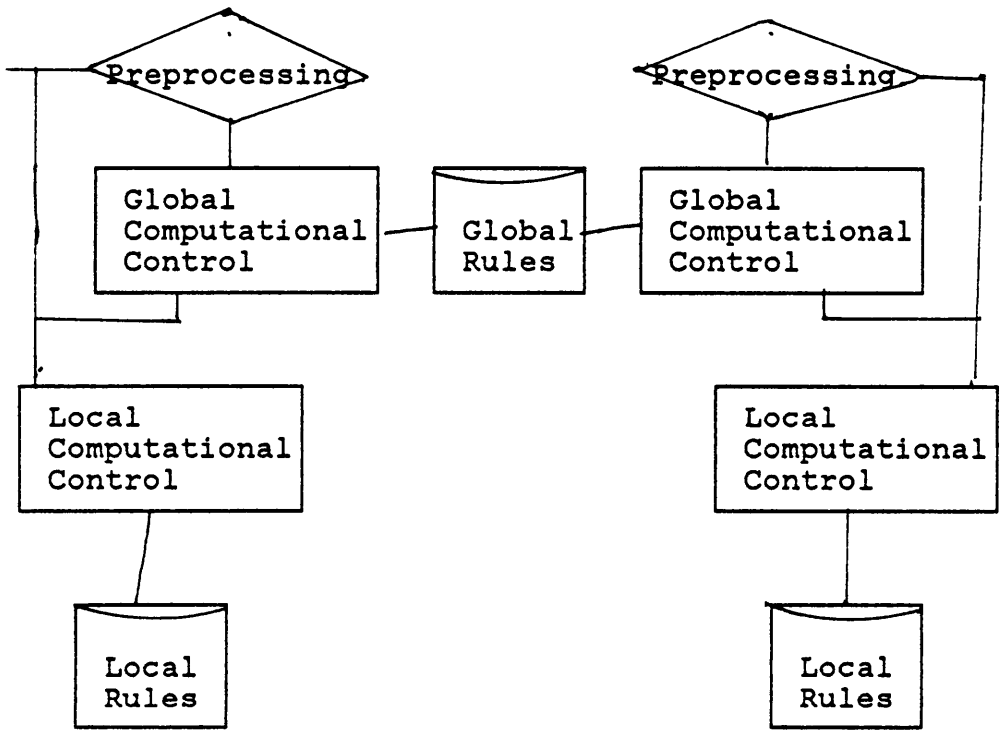
Figure 22. Logical Blocks 7



Block L(SRM)

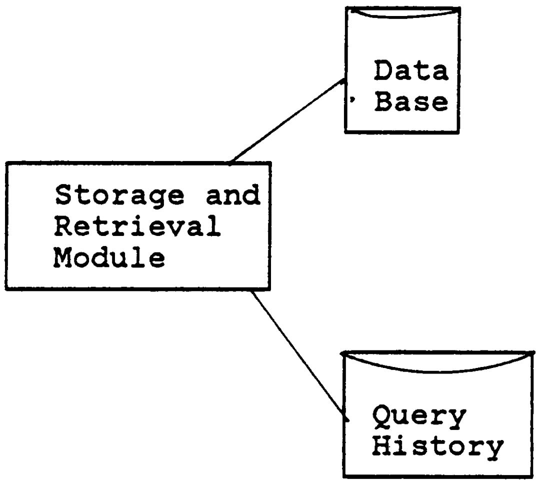


Block L(ROM)

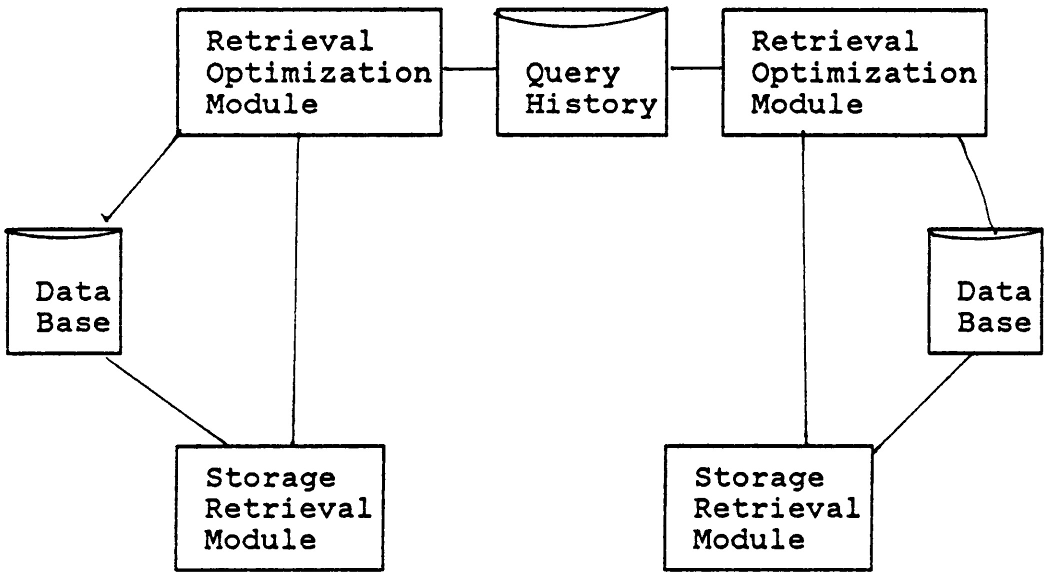


Block L(PP-GRM-LRM)

Figure 23. Logical Blocks 8



Block L(SRM-QH)



Block L(ROM-SRM)

Figure 24. Logical Blocks 9

rules. The Query Decomposition and Optimization module located at one logical site, in block L(QDOM-TRCM-SCM), communicates with other logical sites to transmit the subqueries. Block C(PP-GRM-LRM) is translated into block L(PP-GRM-LRM) covering the two logical sites. Finally block C(ROM-SRM-SCM) is translated in blocks L(ROM-SCM) which spans over the two sites and block L(SRM-SCM) of which there is one at each site. Block C(SRM-QH) is translated to block L(SRM-QH) for each site.

4.4 PHYSICAL LEVEL.

The mapping of the logical models into the physical topologies is made by implementing subsets of each logical site at different physical sites.
The communications between the physical sites is assumed to be implemented by a Network Access Modules (NAM). The translation of the logical modules into physical ones is therefore done by placing Network Access Modules at different levels in the logical model. This results in several components which are then implemented at different physical sites. However, the Network Access Modules cannot be placed arbitrarly between any two blocks because the structure of the logical model and the security of the DDBMS have to be preserved. The basic property which has to be satisfied is the isolation of User Access Modules

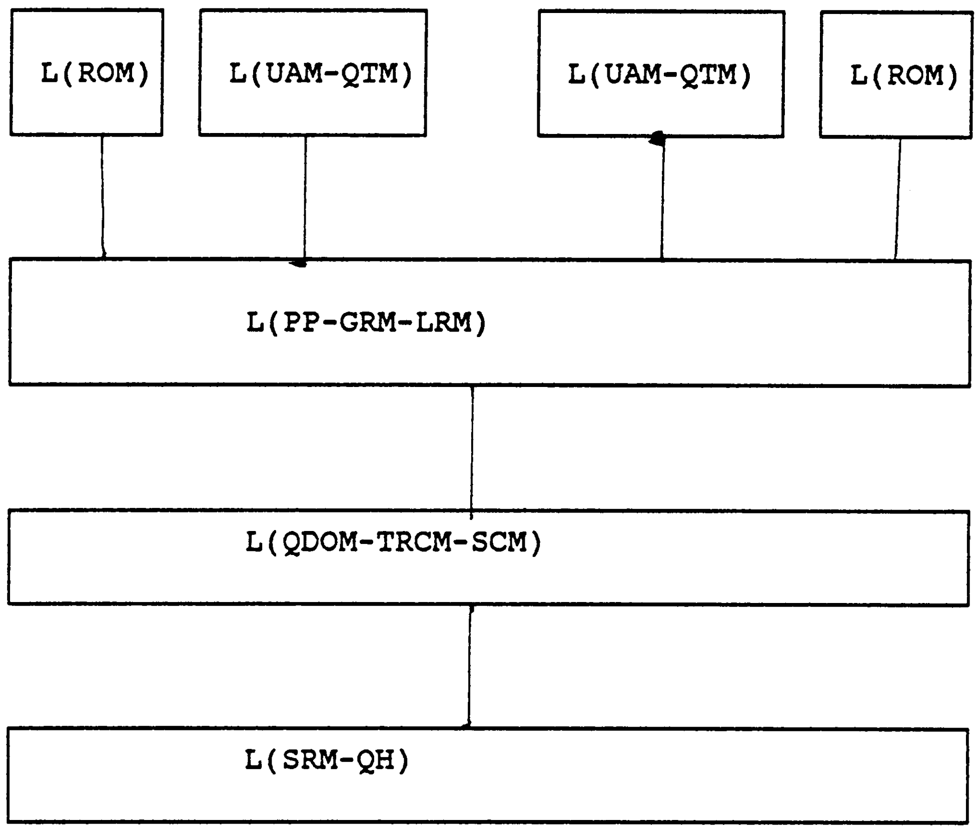


Figure 25. Logical Model PA and FC, ROM as User: Partial Autonomy, Flow Control, Retrieval Optimizer as a User.

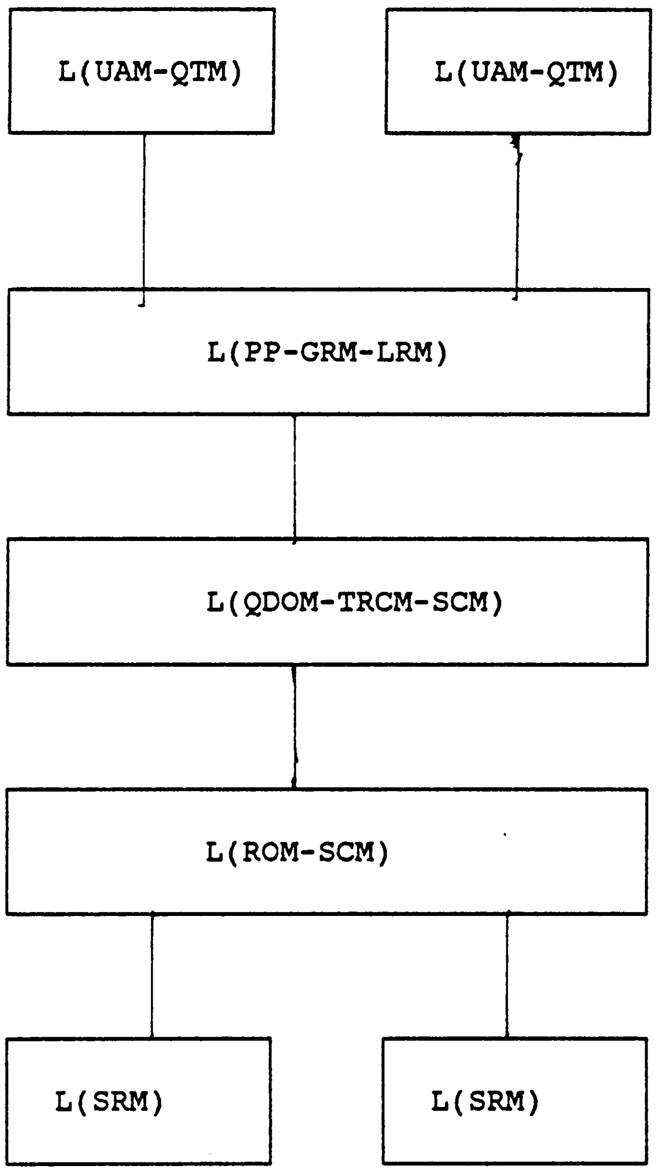


Figure 26. Logical Model PA, FC, ROM Integrated: Partial Autonomy, Flow Control, Retrieval Optimizer Integrated.

from Storage and Retrieval Modules through all the different security layers. In addition, a query decomposition cannot be effective if the temporary routing and storage rules are not checked. Similarly no data move can be completed without having been approved by the Storage Constraints enforcement module.

Different implementations are obtained depending on the underlying network topology. However, it is possible to implement a logical model physically independently of the network topology, if the following condition is satisfied:

<u>THEOREM.</u>

A necessary and sufficient condition to assure that the physical model is correctly implemented is the following: Network Access Modules are introduced in the logical model so that components of type User Access Module, i.e. containing a User Access Module, and Storage and Retrieval Module components, i.e. containing an Storage and Retrieval Module, are created. All User Access Module components should have a minimum set of security modules: SMUAM, which is the same for all User Access Module components, and all Storage and Retrieval Module components should have the same minimum set of security modules: SMSRM. All security modules are either in SMUAM or SMSRM.

SMUAM = security modules of all User Access Modules.

SMSRM = security modules of all Storage
and Retrieval Modules.

All security modules = (SMUAM) U (SMSRM).

The order of the modules in the logical models has to be preserved within each component.

PROOF:

We will first prove that the property defined in the theorem is a sufficient condition. Suppose that all components containing a User Access Module have a minimum set of security modules SMUAM and all components containing a Storage and Retrieval Module have a minimum set of security modules SMSRM. Suppose further that SMUAM and SMSRM form a partition of the set of all security modules, i.e. that any given security module is either in SMUAM or SMSRM. We also assume that the order of the modules in each component corresponds to the order in the logical model. This implies in particular that the User Access Module is the first module in any User Access Module component and that the Storage and Retrieval Module is the last module in any Storage and Retrieval Module component. A transaction issued from a User Access Module has to be transmitted to a Storage and Retrieval Module to access any data. Before being transmitted to a component containing a Storage and Retrieval Module, the transaction has to be first submitted to all the security modules of the User Access Module, and therefore to all modules in SMUAM. When transmitted to a Storage and Retrieval Module component the transaction is first analyzed by all the security modules in the Storage and Retrieval Module component and is therefore controlled by all modules in SMSRM. We can then conclude that the transaction has been subject to all the security checks since all security modules are in either SMUAM or SMSRM.

We will now prove that the condition listed in the theorem is a necessary one. Lets assume that not all User Access Module and Storage and Retrieval Module components hold the corresponding set of minimum security modules. This means that there exists a User Access Module component UAM1 and a Storage and Retrieval Module component SRM1 such that:

All security modules = (SMUAM1) U (SMSRM1) U (RSM)

where SMUAM1 and SMSRM1 are the security modules in UAM1 and SRM1 respectively, and RSM represent all the security modules which are in neither SMUAM1 nor in SMSRM1. According to our assumption RSM is not empty, and a transaction issued from UAM1 directed to SRM would be subject to all the security modules in (SMUAM1) U (SMSRM1) but not to the modules in RSM. The security of the system would then be violated, which leads us to conclude that it is necessary that the condition be satisfied for the system to be secure.

The theorem expresses the basic rules that need to be followed if the network topology is unknown. However, by using the properties of the network structure it is possible to build more efficient physical implementations. We will consider four basic network topologies for the implementation of the logical models: star, ring, tree and broadcast.

We will not go in the lengthy process of describing the implementation of each model in the different underlying topologies. We will instead analyze the implementation of one model from which the implementation of other models can easily be derived.

Before describing the different implementations we need to translate the security requirements for the physical implementation previously defined. The User Access Module needs to be separated from the Storage and Retrieval Module by all the corresponding security modules, and the query decompositions and data moves have to be validated by the Storage Constraints and Temporary Routing Constraints modules. These requirements translate into the following rules:

- Block L(UAM-QTM) should be separated from blocks L(SRM-DDB), L(SRM-SCM), L(SRM-QH) or L(ROM-SRM) by a block L(GRM) or L(PP-GRM-LRM).

- When local rules are in effect block L(UAM-QTM) should also be separated from blocks L(SRM-DDB), L(SRM-SCM),L(SRM-QH) or L(ROM-SRM) by block L(LRM) or L(PP-GRM-LRM).

- Blocks L(UAM-QTM) and L(SRM-SCM) should always be separated by block L(ROM-SCM).

- Block L(ROM) should be separated from block L(SRM-DDB), L(SRM-SCM), L(SRM-QH) or L(ROM-SRM) by block L(QDOM-TRCM-SCM).

Because of the structure of the blocks, query decompositions are always checked against the temporary routing and storage constraints if the logical blocks are implemented as basic physical units.

We will describe the implementation of the logical model for the case of partial autonomy, flow control and Retrieval Optimizer integrated under the different topologies.

Although several implementations for a given topology can be realized following the rules previously defined, we will consider only the most efficient one for each topology. The logical blocks previously defined are used in the physical implementation. However, block L(PP-GRM-LRM) is further divided in block L(GRM) and L(PP-LRM). Block L(GRM) handles the enforcement of the global rules and is represented in figure 27. Block L(PP-LRM) contains the local computational rules enforcement module as well as the preprocessing module. The two functions need to be separated at the physical level because they can be implemented at different sites.

In the case of partial local autonomy and flow control, the organizations retain control over the location and the enforcement of the rules defined on their data but not on the location of the data. Therefore, the enforcement of local rules cannot be done at the site of the data. In this case the global level has to be trusted to transfer the control to the corresponding local enforcement components. Local users can still be defined but the queries issued by these users, although checked only by local rules, may have to be directed through sites holding global rules or other local rules. The Network Access Module of each site is then responsible for transmitting directly to other Network Access Modules transactions which do not have to be checked by the security components of the site. In this way the security of the system is never jeopardized, but too many checks may be performed on the trans-

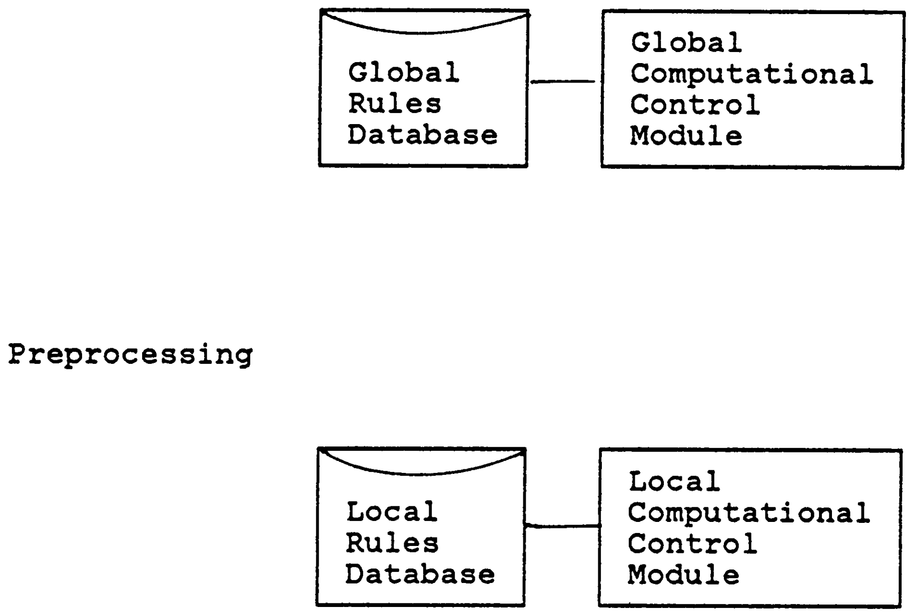

Global Rules Database
Global Computational Control Module
Preprocessing
Local Rules Database
Local Computational Control Module

Figure 27. Division of Block L(PP-GRM-LRM)

actions if the Network Access Modules do not function properly and the organizations may lose some of their autonomy. However, if the Network Access Modules are proven to function properly both the security of the system and the independence of the organizations can be easily achieved.

1. Star.

   In the star topology all nodes communicate through a central node: the hub. For efficiency reasons the Query Decomposition and Optimization module function is placed in the hub of the star as illustrated in figure 28. Block L(QDOM-TRCM-SCM) is therefore placed in the hub. The Retrieval Optimizer (block L(ROM-SCM)) and the global rules (block L(GRM)) are also placed in the hub of the star. Some specialized nodes of the star hold the data and the Storage and Retrieval Modules (block L(SRM-SCM)), while others are users sites and hold the User Access Modules (block L(UAM-QTM)) as well as the local rules enforcement (block L(PP-LRM)). Each site is equipped with an Network Access Module to communicate with other sites.

   When no flow control is performed the flow control components are simply omitted from the physical model. When the network organizations do not retain any autonomy the local rules do not have to be checked.

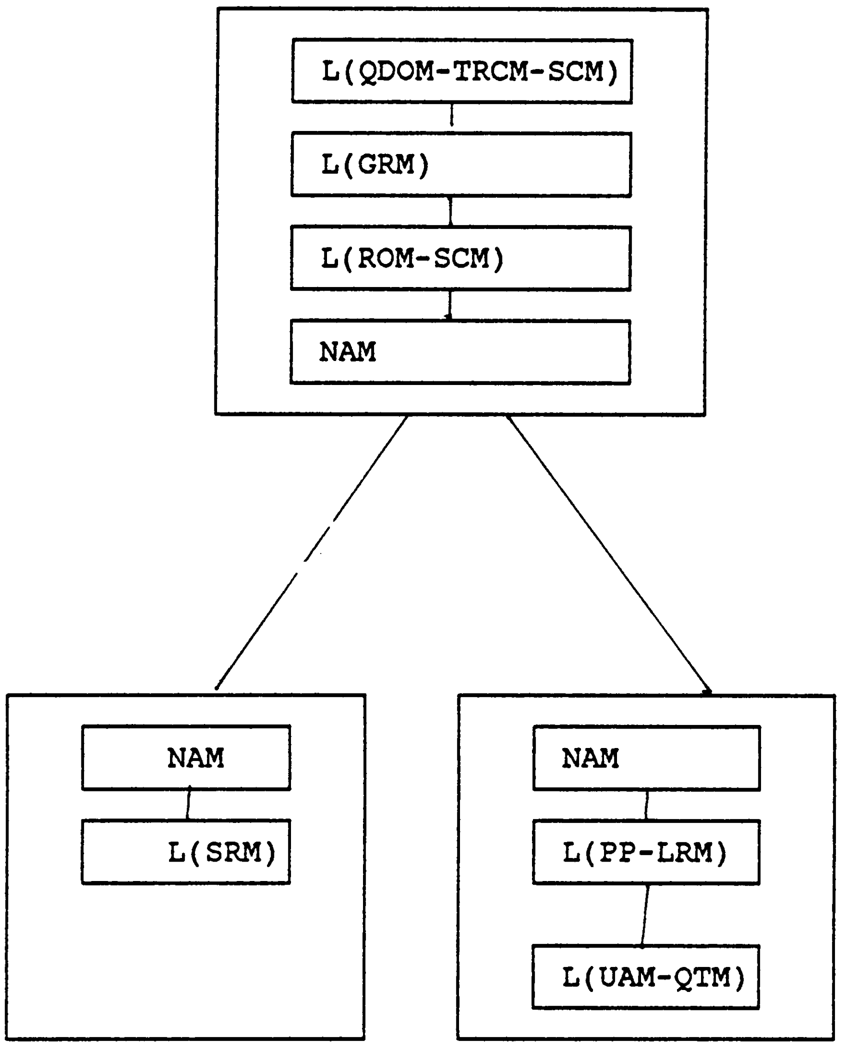


Figure 28. Physical Implementation Star PA, FC: Partial Autonomy, Flow Control

In the case of total autonomy the local rules can be associated with the data.

2. Ring.

We consider here a double direction ring. Each node communicates directly with two other nodes. In the design of the physical model for the ring, the structure of the ring is used to isolate the users from the Storage and Retrieval Modules by the security layer. All User Access Modules are located on one side of the ring, the Storage and Retrieval Modules are all located on the other side of the ring and the security layer is situated in between. While this structure is easy to apply in the case of no autonomy, where only one set of rules is defined, it has to be modified to fit the cases where local rules are introduced.

When the organizations retain their total autonomy, the ring has to be divided in sectors. In each sector two local rules enforcement modules and Storage and Retrieval Modules are implemented surrounding a group of User Access Modules considered local to the Storage and Retrieval Modules. The sector also includes two global rules enforcement as well as Query Decomposition and Optimization module and flow control modules surrounding the Storage and Retrieval Modules. In this way the User Access Modules are physically isolated from the Storage and Retrieval Modules for which they are global users, by the global rules.

In the case of partial autonomy with flow control, illustrated in figure 29, the Storage and Retrieval Modules (block L(SRM-SCM)) are all located on one side of the ring which is isolated from the users (block L(UAM-QTM)) and the local rules (block L(PP-LRM)) by a level of global rules, Query Decomposition and Optimization module and flow control rules (blocks L(GRM), L(QDOM-TRCM-SCM) and L(ROM-SCM)). Local rules enforcement nodes are defined which are located on each side of groups of User Access Modules which they control.

3. Tree.

When the nodes of the network are organized in a tree, the logical blocks are implemented so as to create subtrees which are complete, i.e. contain all the modules as is shown in figure 30. The Query Decomposition and Optimization module (block L(QDOM-TRCM-SCM)) is placed at the root of each subtree along with the Retrieval Optimizer (block 9A) each one associated with the corresponding flow control rules. The descendants of the subtrees roots are either Storage and Retrieval modules (block

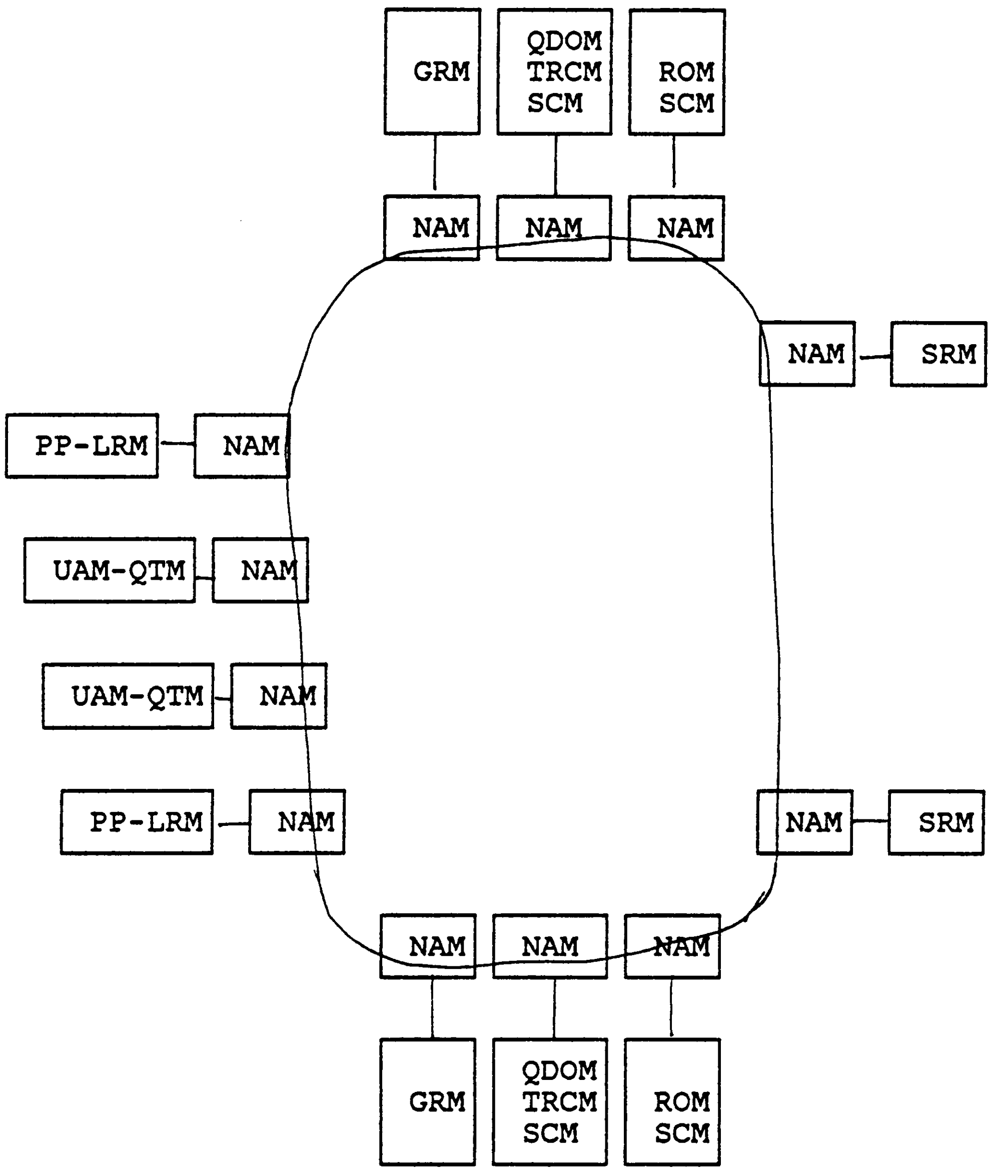


Figure 29. Physical Implementation Ring PA, FC: Partial Autonomy, Flow Control.

L(SRM-SCM)) global rules (block L(GRM)), or other subtree roots. The global rules in turn have local rules (block L(PP-LRM)) descendants. The leaves of the tree are User Access modules (block L(UAM-QTM)).

In the case of no autonomy the local rules do not have to be checked. When the organizations retain their local autonomy, the Storage and Retrieval Modules are associated with their local rules enforcement.

4. Broadcast.

In the broadcast topology all sites communicate directly. It is therefore not possible to exploit the network structure in this case, and the most general property previously defined in the theorem has to be applied to design the network. An example is given in figure 31.

4.5 POLICIES IMPLEMENTATION

To illustrate the flexibility of the model, we will show how the different policy choices can be supported by the model.

1.1.1.SITE OF THE USER INDEPENDENCE.

By using system dependent predicates including the site of the user in the access rules, namely authorizations and

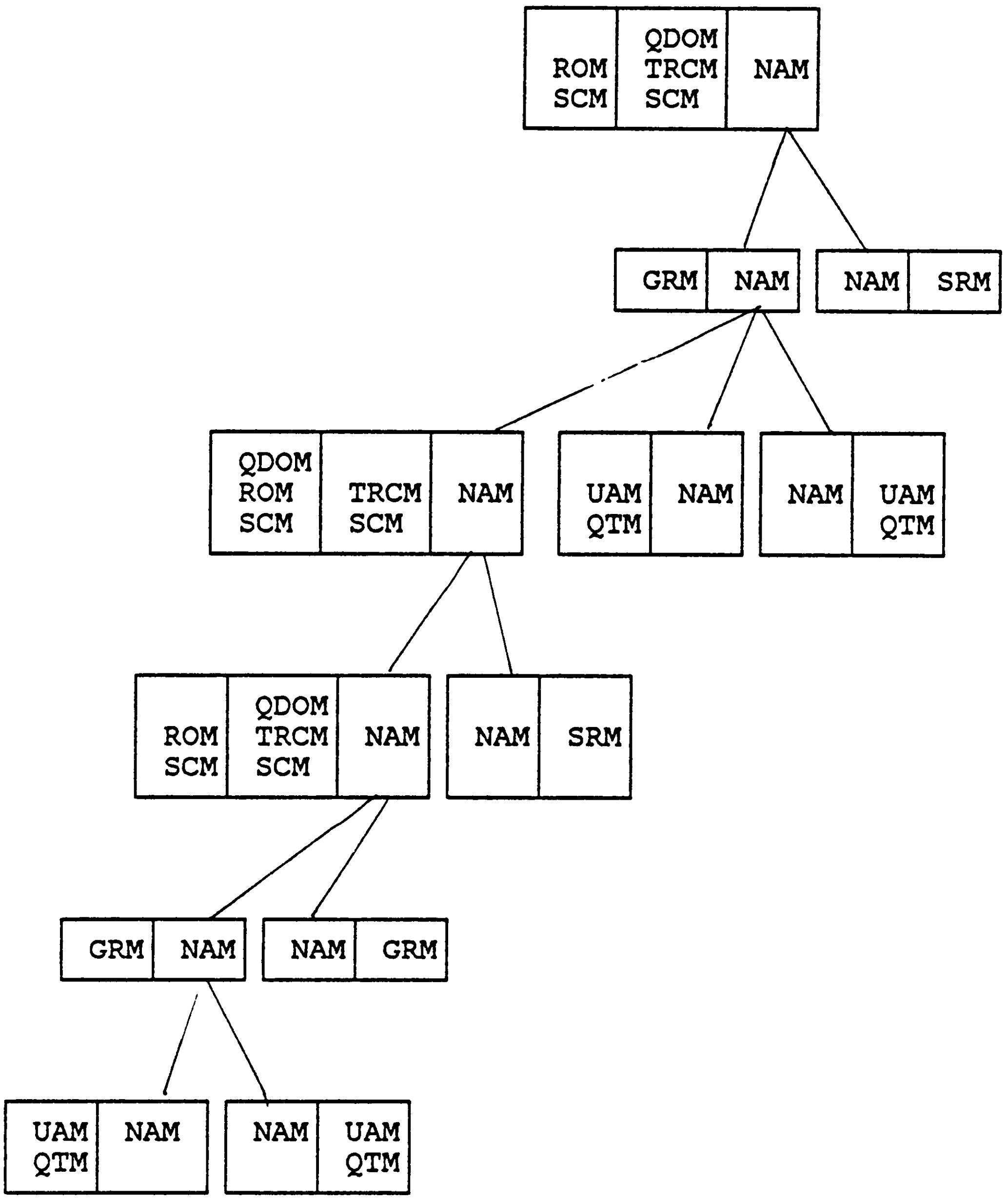


Figure 30. Physical Implementation Tree PA, FC: Partial Autonomy, Flow Control.

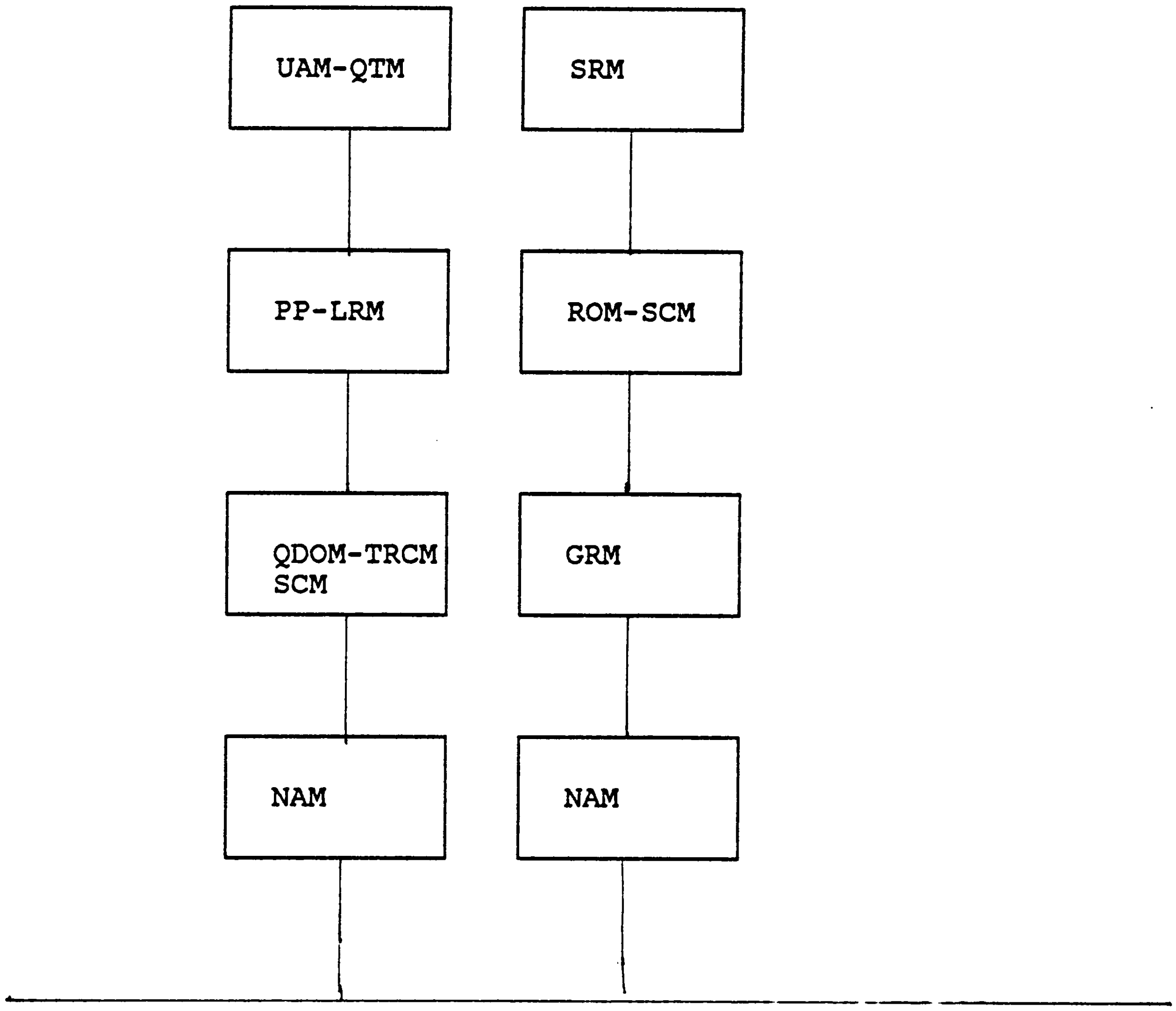


Figure 31. Physical Implementation Broadcast PA, FC: Partial Autonomy, Flow Control.

constraints, the responses to the queries can be made site of the user dependent. However, if the site of the user cannot be a parameter in the system predicate the query responses are always site of the user independent.

1.1.2.SITE OF THE DATA INDEPENDENCE

To assure site of the data independence, consistency of the rules on replicated data has to be maintained. The same local rules for all copies of a replicated data have to apply. The conssitency check can be achieved by using the partial authorizations defined for data controlled by multiple authorizers. Replicated data can be considered as a unique set of data controlled by all the authorizers associated with the different replications. A partial authorization is created whenever an authorizer grants a right to a replication of the data, but the authorization can be effective only when all authorizers for all the replications of the data grant the same right. Alternatively an authorization granted by any authorizer may be effective for all replications of the data even if the other authorizers did not grant the same right. Similarly two options exist for the constraints. A mutual agreement may be reached, or a constraint imposed by any of the constraint imposer always takes effect for all the replications of the data.

If the query responses can be site of the data dependent, it is then enough to allow for the local rules to be different. This correspond to allowing for local autonomy.

If the local rules are different but the information has to be released if available in the DDBMS, the Query Decomposition and Optimization Module can generate the decomposition which would retrieve the requested data if access to at least one replication is authorized. The Query Decomposition and Optimization Module generates the right decomposition according to the rules defined in the Local Rules Module.

1.2.DECOMPOSITION INDEPENDENCE

Because of the close relationship between the decomposition independence policy and the flow control and local autonomy policies, the policy choices for decomposition independence can be reflected by the variations on the organization trust policies. Responses to queries are query decomposition independent if no flow control is performed and if the organizations do not retain any autonomy. If flow control is performed it is then possible to implement the need to know policy stating that the information should be released to the user if there exists an authorized copy of the data in the DDB. This is done by the cooperation of the Query Decomposition and Optimization Module and the Routing Constraints Module. If the organizations retain some autonomy, then the query reponses may be query decomposition dependent. When the same data, managed by different organizations is named differently, the reponses can be made decomposition independent for certain users by defining their views as composed of only one replication of the data, as opposed to any replication of the data.

1.3.SEQUENCE OF QUERIES INDEPENDENCE.

To assure that the order of the queries does not affect the responses, the arc cutting algorithm can be used to enforce the computational constraints. This algorithm being a static one allows for a control which is independent of the sequence of queries order. With this algorithm all possible constraint violations are analyzed statically at the time of the introduction of a constraint. The arc cutting algorithm also assures that unauthorized information cannot be derived within or outside of the system. However, the tagging algorithm is a more efficient and flexible algorithm. But because of the inheritance mechanism incorporated in the algorithm and because the enforcement is performed in a dynamic fashion, the tagging algorithm is sequence of queries dependent.

2.1.1.RIGHTS PRESERVATIONS WITH NEW AUTHORIZATIONS.

To assure that a user is aware of the previously acquired rights that are affected by the granting of a new authorization, an analysis of all the authorizations and constraints of the user can be performed. However, the overhead incurred by this analysis may not be acceptable and the users may be given instead just a list of all their rights and the analysis may be left to them.

2.1.2.RIGHTS PRESERVATION WITH NEW GROUP MEMBERSHIP

The choices for this policy can be handled in a similar fashion to the ones for the rights preservation with new authorizations policy.

2.2.DISCLOSURE

Because of the organization of the authorizations and constraints in relational data bases, it is easy to query the data bases at any time to obtain all the rights a user has.

2.3.PRIVACY

If users are allowed to keep their rights confidential, the authorizers have to impose their constraints wihout having an exact knowledge of the users rights. To assure the maximum security, authorizers may assume that the users have access to most data in the DDB. However, this assumption may be too conservative and imply that too many constraints have to be formulated.

3.COLLECTIBILITY OF RIGHTS

When users rights can be collected from different groups, it is reflected in the enforcement algorithm. If rights of different groups cannot be collected, it is easy to suppress the corresponding step in the algorithm.

4.1.ACCOUNTABILITY

For any data a list of authorizations and constraints can easily be obtained by querying the data bases of authorizations and constraints.

4.2.SUPREMACY

Due to the integration property of the hybrid system, authorizations and constraints can be imposed on constraints. This allows to control the effect of constraints on authorizations. If the constraints cannot overide the authorizations if the constraint imposer does not have specific rights over the data, this can be expressed through a set of authorizations and constraints over the constraints themselves.

4.3.REVERSIBILITY

In the hybrid system rights can be passed through computed relations for which rights are automatically granted to user who computes relations which remains stored in the DDB. To be able to restore the protection state of the system after revocation of an authorization, a history keeping mechanism has to be maintained to record for all computed relations the list of relations used in the computation. The tagging algorithm allows to trace back all authorizations affected by a constraint and therefore to restore the protection state of the system after the annulment of a constraint. If the history of the system does not have to be kept and the system protection state can be different after revoking an authorization or annuling a constraint, the authorizations and constraints can then just be erased. In this case the effect of the revocation or annulment is not retroactive. However, if the history of the system is to influence the present protection state, the arc cutting algorithm can be used. Whenever an authorization is revoked, the corresponding arc in the user's graph is not cut. In this manner, the system has a record of all the authorizations that were granted at any point in time to the user.

4.4.MINIMUM POWER

A minimum set of rights for each user can be defined in the hybrid system by imposing constraints on the constraints that can be imposed for each user.

4.5.SCOPE

If the tagging algorithm is used, constraints do not affect any data not explicitly expressed in the constraint. The arc cutting algorithm allows for the other policy alternative to be implemented.

4.6 CONCLUSIONS.

The analysis of the different levels of the models allow us to derive some basic conclusions about the proposed models.

1. The modeling of the protection is affected only by decisions taken in two policy areas: local autonomy and flow control. Furthermore, the proposed model is powerful enough to adapt to any combination of the two policy choices.

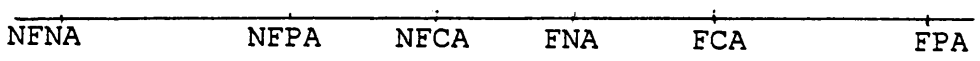
2. The degree of complexity introduced by a policy choice can be easily analyzed because of the modular approach taken to design the models. To compare two sets of policy choices, the blocks which appear in the model for one set of policy choices but not in the other one are compared for degree of complexity. Starting with the model for no local autonomy and no flow control, the effect of introducing different degrees of local autonomy and flow control can be observed by analyzing the blocks that have to be added or modified to support the different policy choices. Following this process we can draw three main conclusions:

   a. The flow control policy choice is the most influencial one in terms of the degree of complexity. It is clear that chosing to introduce flow control increases the complexity of the system substantially independently of the autonomy policy choice.

b. When there is no flow control performed the models become more complex when higher degrees of autonomy are introduced. The model supporting complete autonomy is more complex than the one supporting partial autonomy which in turns is more complex than the one representing the policy of no autonomy. However, when flow control is integrated in the model, the introduction of partial autonomy increases the complexity of the model more than the introduction of complete autonomy. This suggests that there is a strong interaction between partial autonomy and flow control.

c. The degrees of complexity introduced by various choices of the policies vary depending on the policy choices. For example, When no flow control is performed, the introduction of partial autonomy increases considerably the complexity of the model, while adding complete autonomy makes the model only slightly more complex than the one with partial autonomy. These differences in the level of complexity for the different models are graphically represented in figure 32.

d. In addition to the choices taken for the policies of flow control and local autonomy, a decision has



NFNA	No Flow Control No Autonomy
NFPA	No Flow Control Partial Autonomy
NFCA	No Flow Control Complete Autonomy
FNA	Flow Control No Autonomy
FCA	Flow Control Complete Autonomy
FPA	Flow Control Partial Autonomy.

Figure 32. Models Complexity

to be taken in relation with the positioning of the Retrieval Optimizer. The Retrieval Optimizer may be placed as a user and appear in the model at the same level as the User Access Module, or it may be integrated with the block containing the Storage and Retrieval and the Storage Constraints Modules. The integration of the Retrieval Optimizer increases the complexity of the model but allows for a cleaner division of the security and retrieval functions.

3. The proposed model can be implemented on top of any physical topology. We defined a property which ensures that the internal security of the model is always satisfied independently of the network topology. However, using the physical properties of the different topologies, it is possible to build more efficient physical models which also preserve the desired security properties. It is always possible to check if a physical model for a given topology satisfies the security requirements, i.e. that the User Access Modules are isolated from the Storage and Retrieval Modules by all the necessary layers of security. However, a more general property like the one defined when the network topology is not considered cannot easily be

characterized, due to the inherent differences between the topologies.

5.0 CHAPTER 5 CONCLUSIONS AND FURTHER RESEARCH

5.1 CONCLUSIONS

The trend toward distributed processing of the 1980's has given rise to new important protection problems. Although security is important in any centralized system, it becomes crucial when the data is to be shared among a variety of organizations. However, very little research has been conducted in the area of DDBMS protection. This work was intended as a contribution to the study and modeling of the DDBMS protection issues. We identified the main protection issues and requirements associated with DDBMS, presented a system satisfying these requirements, and finally integrated this system into a general model for DDBMS protection. The main contributions this work makes to the field of DDBMS protection can be classified in three major groups: analysis of the problem, hybrid protection system, and DDBMS protection models.

1. Analysis of the problem.

   By identifying the main differences between distributed and centralized systems, we characterized the specific protection problems for DDBMS. This was a first step in determining the main issues which need to be addressed. The second step consisted in translating the differences between centralized and distributed systems into requirements that need to be fullfilled for distributed protection, and which are not generally satisfied in centralized systems.

To complement the derived requirements, we identified and formalized a set of important policies. Different DDBMS will require different sets of policy choices to be implemented. However, the set of policies defined is general enough to include the needs of most organizations. The interaction of these policies was also studied to determine the effect a choice for one policy may have on the choices made for other policies.

2. Hybrid protection system

   We designed a hybrid protection system which satisfies the requirements previously determined. This system possesses a certain number of important characteristics:

   - The hybrid system is powerful enough to allow for variations in the policy choices. The hybrid system is flexible and of practical value since it can easily accomodate different types of environment represented by the different set of policy choices. Furthermore, while satisfying all the requirements for distributed protection, the hybrid system can handle protection for centralized systems as well.

- By using both authorizations and constraints, an economy of representation is achieved. With respect to a user, the DDB can be partitioned in two sets: the set of accessible data and the set of data for which access is restricted. By using either authorizations or constraints, the smallest set can always be explicitly defined. However, the security gap created by forgetting a constraint in an open system is avoided in the hybrid system, due to the authorization mechanism incorporated in the system.

- The flow of information between sites as well as among users can be completely controlled in the hybrid system. The information flow control among users, has already been studied and solved for several nondiscretionary systems. However, nondiscretionary systems require a very rigid structure which is not appropriate for the general type of applications typical of many DDBMS. The hybrid system offers the flexibility of discretionary systems while providing means for controlling the flow of information in the system and among users.

- By using the different tools provided by the hybrid system, it is possible to give a practical solution to the safety decidability problem. The safety problem while solved in some restricted cases, has been proven by Harrison (HARR76) undecidable in the general case. In the hybrid system authorizers have the certainty that no user can gain access to data if it was not the intent of the data's authorizer. This can be assured through the constraints mechanism.

- The hybrid system is constructed around a central mechanism of authorizations and constraints. This mechanism controls accesses to all data bases supported by the DDBMS, including the authorizations and constraints data bases themselves, as well as the flow of information in the system. The hybrid system is therefore an integrated system which is self regulating since authorizations and constraints can be defined on the data bases holding them.

3. DDBMS protection models

By integrating the hybrid system in a more general model for DDBMS protection we were able to provide a more complete representation of the protection issues in DDBMS and to link these issues to the other DDBMS functions. The main characteristics of the model design are summarized below:

- A modular approach was taken to develop the model. By using modules to define the model, a great flexibility was achieved and the main policy choices could easily be accomodated.

- Three levels of abstraction have been defined to represent the different aspects of DDBMS protection problems. This division was also used to focus more precisely on the different issues. The conceptual level describes the protection control of the DDBMS transactions and information flows, the logical level is concerned with the interaction between the different organizations participating in the DDBMS, while the physical level is involved with the architectural implementation of the logical level. These three levels are designed in a sequential fashion. The transition from one level to the next is governed by transition rules between conceptual and logical levels, and between logical and physical levels.

- The effect of the different network topologies on the model was studied. Basic protection properties which need to be satisfied to assure the security of the physical level model implementation were defined. A rule for the implementation of a secure physical level model independent of the network topology was defined. This rule shows that the model can be implemented in the same fashion independently of the network topology. However, we showed that the network topology can be used to design a more efficient physical level model which satisfies all the protection properties defined for the design of the physical level model.

- After studying the effect of the different policy choices on the design of the model it was determined that only two policies influence the design: organization autonomy and flow control. The organization autonomy policy defines the degree of control each organization retains over its data. The flow control policy determines the level of intersite trust, and if the flow of information between the sites have to be controlled or not.

By using the modular design it was easy to define the models for the different set of policy choices for these two policies. The different models can be constructed by choosing the necessary modules and organizing them correspondingly.

5.2 FURTHER RESEARCH

In this dissertation we identified the main DDBMS protection issues and presented some solutions, but much work still remains to be done in the area. In this section we identify some of the areas of the work presented in this disssertation which need to be investigated further.

- One of the strengths of the hybrid system lies in the fact that it was designed in an integrated fashion. This integration gives the system considerable power to handle a variety of protection problems. However, this same power if not controlled correctly could lead to unwanted results. An example of such problems is the deadlock that may occur in the system if a set of constraints CON1 is created which inhibits all authorizations of the system and for which another set of constraints CON2 exists which state that CON1 cannot be annulled. This deadlock can be avoided if a main authority like a network security administrator exists in the DDBMS. The system may then be initialized by writing authorizations which give complete powers to the administrator and constraints to assure that the administrator can never be constrained in any way. These authorizations and constraints, being part of the system initialization, could never be revoked. These rules would be associated with the network security administrator function and not with any individual in particular. In this manner, although the person assigned to the function may be changed, the rules associated with the function of network administrator remain in effect. The administrator, then, always has the power to resolve any deadlock that may occur in the system. However, this may be only a partial solution to the deadlock problem. This problem needs to be studied in more detail and all the possible implications of the property of integration of the system have to be investigated.

- A user interface exists in the model we presented which interprets users queries for processing by the DDBMS. A similar interface could be built to provide protection officers with a higher level language for protection rules definitions. By using this interface authorizers could express the protection rules in general terms without having to write explicitly all the authorizations or constraints, nor to understand the details of the mechanism which imposes constraints or authorizations.

- The implications of authorization revocations and constraints annulements with the different enforcement algorithms may have to be thoroughly investigated. Authorizations revocations and constraints annulments can be reflected in the authorizations constraints mechanism of the hybrid system, but the mechanisms and the consequences on the protection state of the system have not been studied in detail in this work.

- This work is intended as a theoretical contribution to the relatively new area of DDBMS protection. However, to show that the proposed models can perform satisfactorily a prototype needs to be built and a complete performance analysis made.

- The hybrid system is flexible enough to adapt to a great number of protection needs. The level of protection provided by the system can be dependent on the type of data protected. Very sensitive data can be fully protected using all the different types of authorizations and constraints, while the overhead for data of lower sensitivity can be reduced by using fewer types of constraints. The protection mechanism overhead can also be reduced by choosing policies which allow for more flexibility of the system. An example of such a policy is the consistency of response for users' transactions. When the same data is controlled by multiple authorizers, the overhead introduced by protection system is lower if some inconsistency between the rules of the different authorizers is allowed. The overhead introduced by the different levels of protection which can be maintained by the hybrid system has to be studied to give potential network contributers an aid in defining their protection needs.

- We studied in detail the enforcement mechanisms for the most difficult and most critical type of constraints in the hybrid system, namely the computational constraints. Similar enforcement algorithms have to be developed for user flow and routing constraints.

APPENDIX A. CONCEPTUAL MODELS

We will present in this appendix the conceptual models for the different policy choices. These models are represented in figures 33-39. By combining the different choices for the policies of trust between sites we obtain six cases.

1. No autonomy and no flow control.

   This is the simplest model. Transactions issued from the User Access Module are processed by the Query Decomposition and Optimization module after having been controlled by the global rules. Since the organizations do not retain any form of autonomy, the only protection rules are the global ones. The Query Decomposition and Optimization module communicates directly with the Storage and Retrieval Module to retrieve data since no flow control is performed. Two cases have to be considered depending on where the Retrieval Optimizer is located.

   a. If the Retrieval Optimizer is placed as a user the Storage and Retrieval Module is then responsible for accessing the Query History data base as well as moving the data in response to transactions initiated by the Retrieval Optimizer.

b. If the Retrieval Optimizer is integrated the Query History data base querying and the data moves are directly performed by the Retrieval Optimizer.

2. No autonomy and flow control.

   As with no flow control two cases have to be considered: integrating the Retrieval Optimizer or placing it as a user. Blocks C(QDOM-TRCM) and C(SCM) replace block C(QDOM) since the query decomposition and optimization as well as the data moves have to be controlled by the flow control rules, i.e. the temporary routing and storage rules. If the Retrieval Optimizer is placed as a user block C(ROM-SRM) is replaced by block C(SRM-QH) and the Retrieval Optimizer accesses to the DB are controlled by the Storage Constraints.

3. Total autonomy and no flow control.

   When the organizations retain complete autonomy the data is not relocated in the network by the Retrieval Optimizer since the control of the site of data is done by the organizations themselves.

   When the organizations are allowed total autonomy it is understood that global rules are still defined for data controlled by more than one organization at a time such as the join of two relations owned by different organizations. However, if a transaction is purely local to the organization, i.e. submitted by users in the organization and not involving data at other organizations, it should not be controlled by any global rules. A preprocessing module is therefore introduced to determine if a transaction is purely local or if it involves any global data. A local transaction is routed directly to the Storage and Retrieval Module and controlled by the local rules. If the transaction involves global data it has to be controlled by the global rules before it is transmitted to the Query Decomposition and Optimization module which in turn transmits it to the Storage and Retrieval Module under the control of the local rules. The Retrieval Optimizer is not present for the case of total autonomy since the organizations have total control over the placement of their data.

4. Total autonomy and flow control.

   When flow control is introduced block C(QDOM) is replaced by blocks C(QDOM-TRCM) and C(SCM) as in the case of no autonomy

5. Partial autonomy and no flow control.

   In partial autonomy the organizations retain control over the definition and enforcement of the rules on their data. However, they do not control the location of the data. The Retrieval Optimizer has therefore to be introduced in the partial autonomy case. In the same way as in the case of no autonomy the Retrieval Optimizer can be placed as a user or be integrated.

The difference between this model and the one of total autonomy and no flow control lies in the position of the local and global controls. For the case of total autonomy, the global checks have to be made before the Query Decomposition and Optimization module while the local checks have to be made after the Query Decomposition and Optimization module since the rules enforcement has to be associated with the data. When the organizations do not keep a control over the location of the data, the local rules can be checked before the query decomposition since the local rules enforcement is independent of the location of the data. This implies in particular that replicated data are controlled by the same set of local rules. The difference between the no autonomy and partial autonomy models is that block C(GRM) is replaced by block C(PP-GRM-LRM) in the partial autonomy model.

6. Partial autonomy and flow control.

This case was the one used to illustrate the conceptual model in chapter 3. The transition from the case of no autonomy to the case of partial autonomy is made by replacing block C(GRM) by block C(PP-GRM-LRM).

APPENDIX B. CORRESPONDENCE BETWEEN MNEMONICS AND MODULE NAMES

Table of correspondence between mnemonics and module names.

UAM-QT	User Access Module and Query Translation block.
SRM-DDB	Storage and Retrieval Module and Distributed Data Base block.
QDOM	Query Decomposition and Optimization Module block.
GRM	Global Rules Module block
LRM	Local Rules Module block
QDOM-TRCM	Query Decomposition and Optimization Module and Temporary Routing Constraints Module block.
SCM	Storage Constraints Module block
ROM	Retrieval Optimizer Module

ROM-SRM-SCM	Retrieval Optimizer Module, Storage and Retrival module and Storage Constraints Module block
PP-GRM-LRM	Pre Processing, Global Rules Module and Local Rules Module block
SRM-QH	Storage Retrieval Module and Query History block
ROM-SRM	Retrieval Optimizer and Storage and Retrieval Module block.

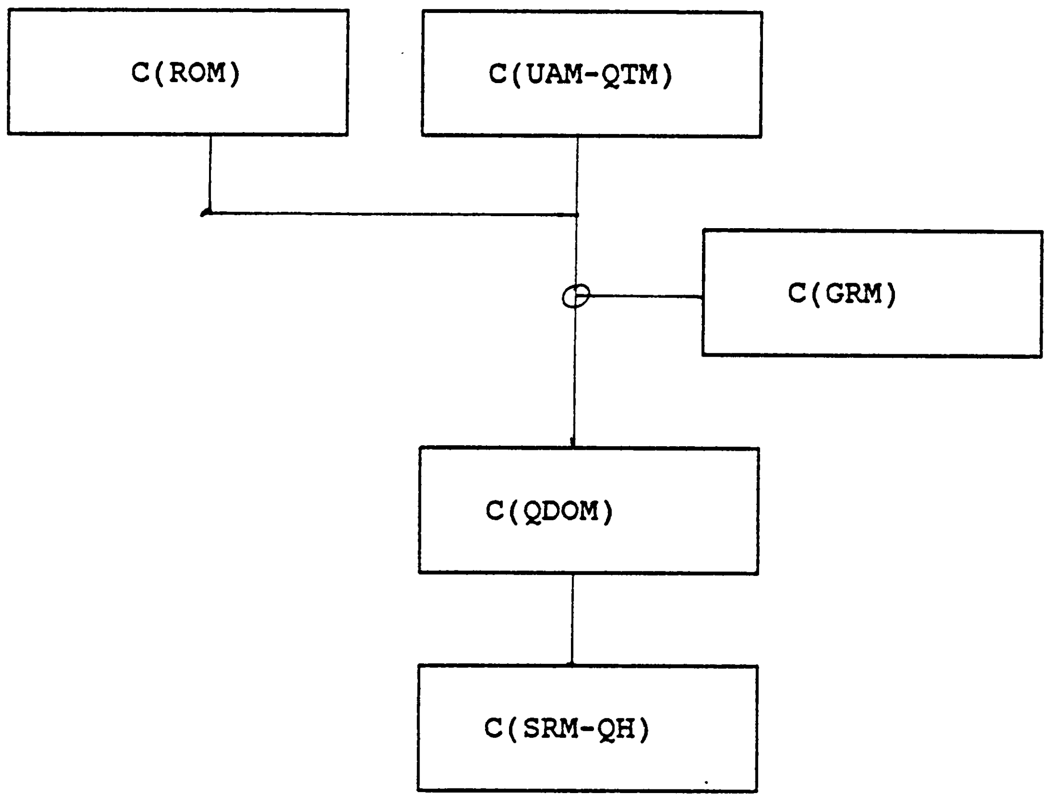


Figure 33. Conceptual Model NA, NFC, ROM as a User: No Autonomy, No Flow Control, Retrieval Optimizer as a User

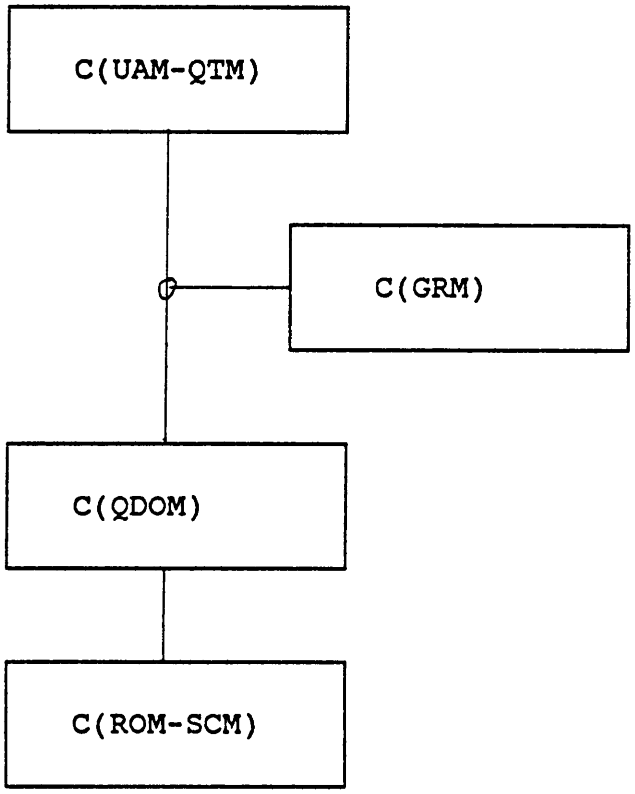


Figure 34. Conceptual Model NA, FC, ROM Integrated: No Autonomy, Flow Control, Retrieval Optimizer Integrated.

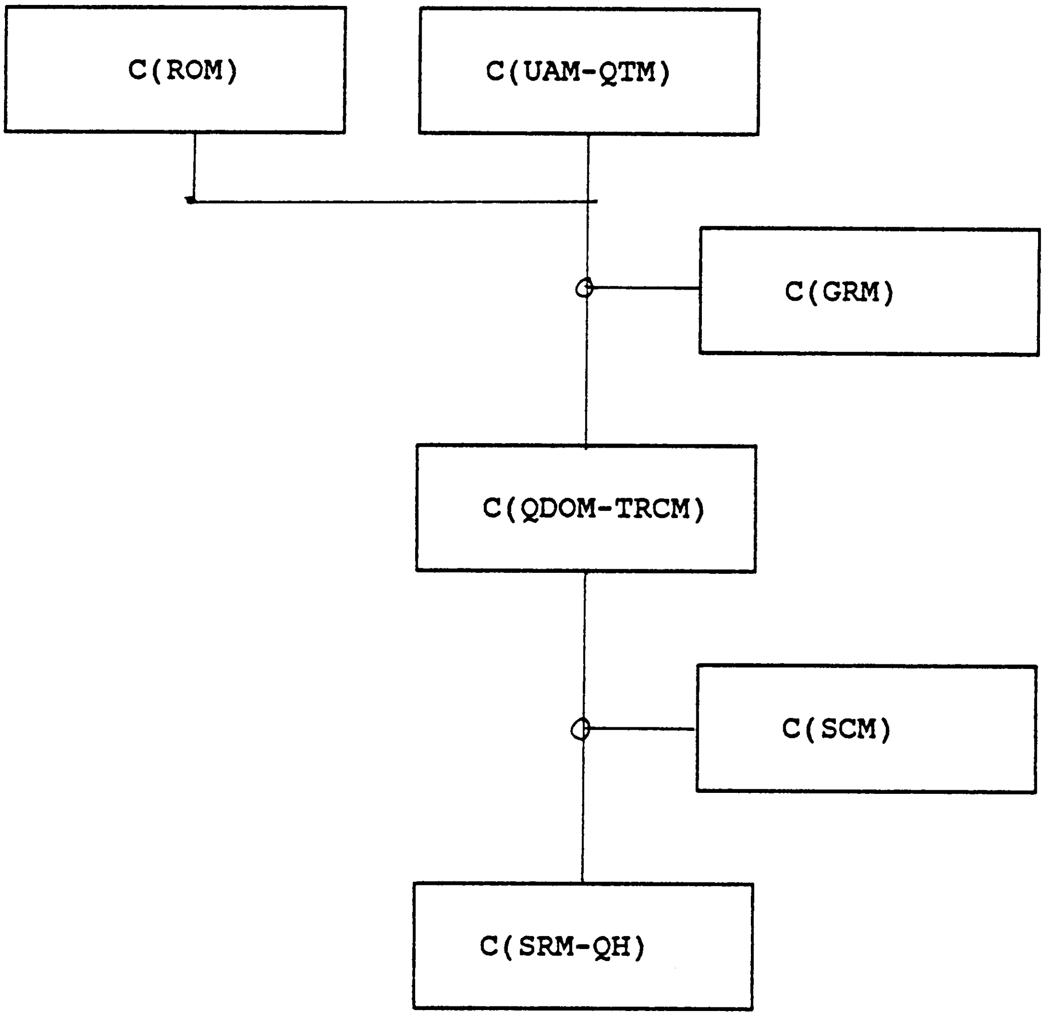


Figure 35. Conceptual Model NA, FC, ROM as a User: No Autonomy, Flow Control, Retrieval Optimizer as a User.

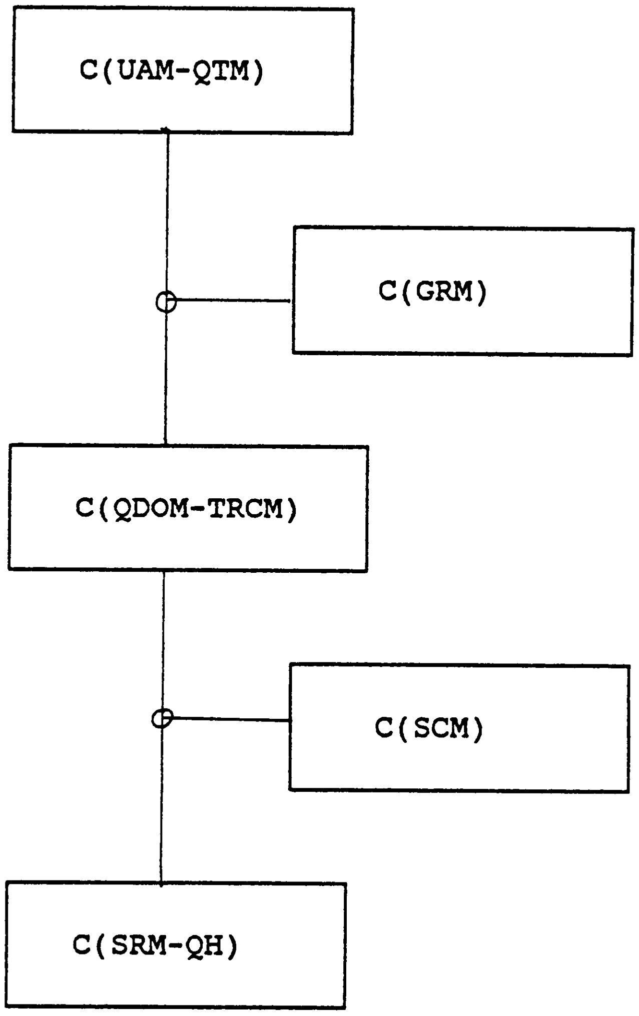


Figure 36. Conceptual Model NA, FC, ROM Integrated: No Autonomy, Flow Control, Retrieval Optimizer Integrated.

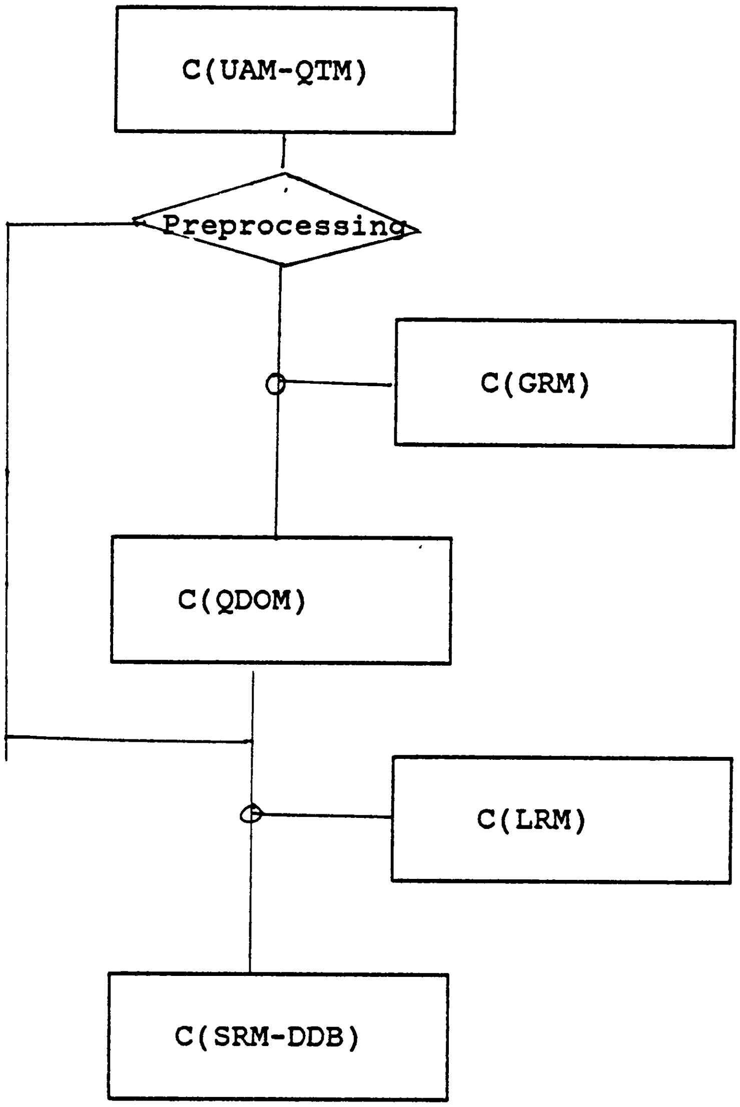


Figure 37. Conceptual Model CA, NFC: Complete Autonomy, No Flow Control.

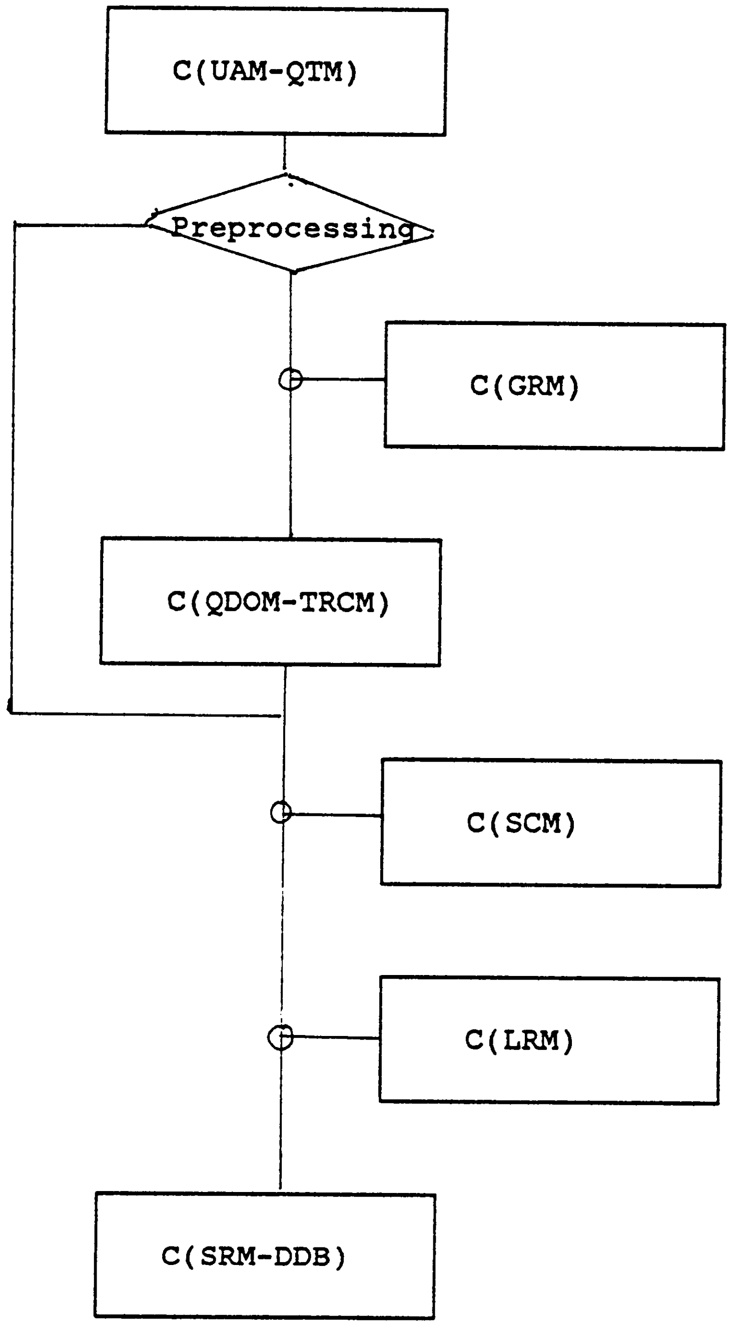


Figure 38. Conceptual Model CA, FC: Complete Autonomy, Flow Control.

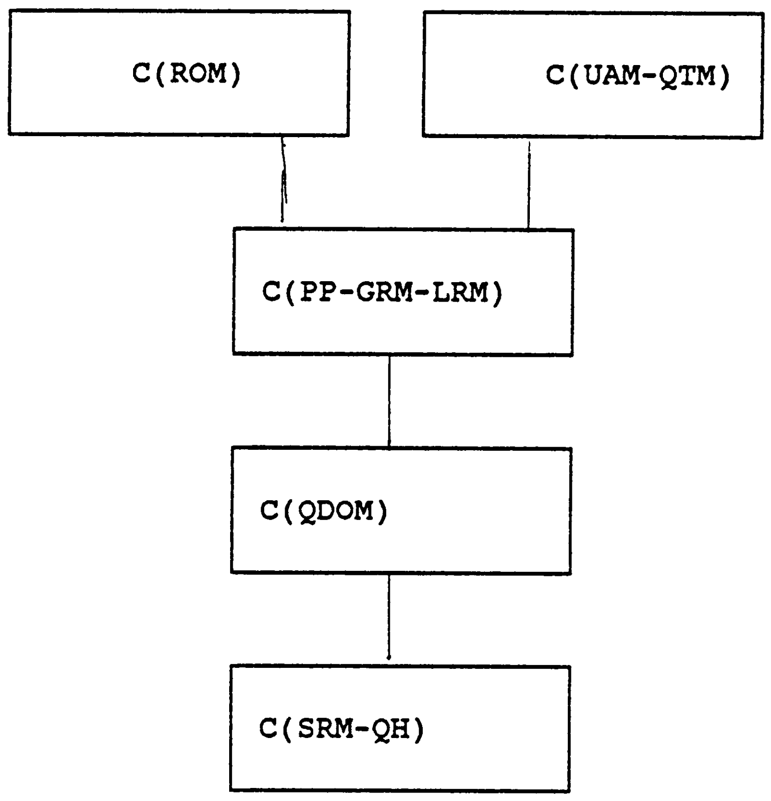


Figure 39. Conceptual Model PA, NFC, ROM as a User: Partial Autonomy, No Flow Control, Retrieval Optimizer as a User

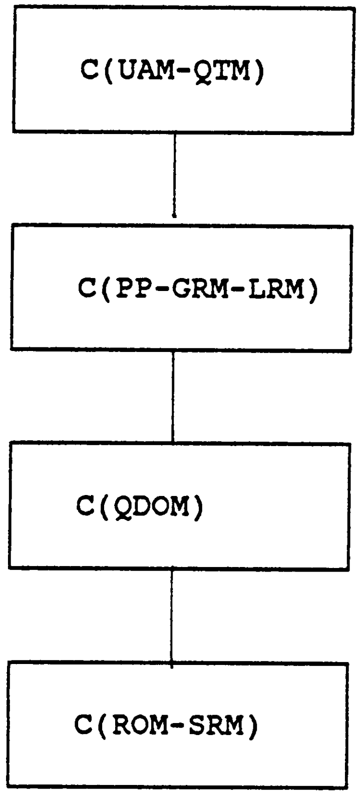


Figure 40. Conceptual Model PA, NFC, ROM Integrated: Partial Autonomy, No Flow Control Retrieval Optimizer Integrated

BIBLIOGRAPHY

ADIBA76 Adiba M.; C. Delobel; and M. Leonard, "A Unified Approach for Modelling Data in Logical Data Base Design", in Modelling in Data Base Management Systems, G.M. Nijssen (ed.), North Holland Publishing Company, 1976, pp. 311-338.

ADIBA78 Adiba, M.; J.C. Chupin; R. Demolombe; G. Gardarin; and J. Le Bihan, "Issues in Distributed Data Base Management Systems: a Technical Overview", Proceedings of the Conference on Very Large Databases, September 13-15 1978, pp. 89-110.

ADIBA82 Adiba, Michel, "Distributed Data Base Research at Grenoble University", Database Engineering, IEEE, September 1982, Vol 5, No 4, pp 2-8.

BERNS81 Bernstein, Philip et al., "Query Processing in a System for Distributed Data Bases (SDD1)", ACM Transactions on Database Systems, Vol 6, No 4, Dec 1981, pp 602-625.

BILLE79 Biller, Horst, "On the Equivalence of Data Base Schemas A Semantic Approach to Data Translation", Information Systems, Vol4, 1979, pp 35-47.

BUSSO80 Bussolati, U., G., Martella; "On Designing a Security Management System for Distributed Data Bases", Proc. of IEEE Fourth Int. Computer Software and Application Conference, COMPSAC 80, Chicago, Oct. 1980.

BUSSO81 Bussolati, U., and G., Martella, "A Data Base Approach to Modelling and Managing Security Information", Proceedings of the 7th International Conference on Very Large Databases, 1981, Cannes, 9-11 September, pp. 532-542.

CARY81 Cary, John M., "Data Security and Performance Overhead in a Distributed Architecture System", UMI Research Press, 1981.

CERI82 Ceri, S.; G. Paolini; G. Pelagatti; F.A. Schreiber; "Distributed Database Research at the Politecnico of Milano", Database Engineering, IEEE, Vol 5, No 4, September 1982, pp 9-13.

CHAN82 Chan, A; and D. R. Ries; "Distributed Database Management Research at Computer Corporation of America", Database Engineering, IEEE, Vol. 5, No 4, September 1982, pp. 14-19.

COHEN77 Cohen, David, "Design of Event-Driven Protection Mechanisms." Ph.D. Thesis, Com puter Science Dept., Ohio State University, 1977.

DAVIE81 Davies, Donald W., "Protection " in Distributed Systems Architecture and Implementation, Lecture Notes in Computer Science, No 105, Ed. G. Goos and J. Hartmans, pp 211-245.

DENNI82 Denning, D.E.R., "Cryptography and Data Security", Addison Wesley Publishing Co., 1982.

DEPPE76 Deppe, Mark E.; and James P. Fry, "Distributed Data Bases a Summary of Research", Computer Networks 1 (1976), pp. 130-138.

DRAFF80 Draffan I. W.; and F. Poole, "The Classification of Distributed Data Base Management Systems", in Distributed Data Bases, I.W. Draffan and F. Poole (ed.), Cambridge University Press 1980, pp. 57-81.

FERNA81 Fernandez, E.B.; R.C., Summers; and C. Wood , "Security and Integrity in Distributed Data Base Systems", in Data Base Security and Integrity,Addison Wesley, 1981, pp 267-288.

GRIFF76 Griffiths, P.P.; and B.W., Wade, "An Authorization Mechanism for a Relational Database System", ACM Transactions on Database Systems, Vol. 1(3), September 1976, pp. 242-255.

GLORI82 Glorieux, A.,M.; and W., Litwin, "Distributed Data User's Needs: Experience from some Sirius Project Prototypes.", Database Engineering, IEEE, Vol 5, No 4, September 1982, pp 23-27.

HAAS82 Haas, L. M. et al., "R* : A Research Project on Distributed Relational DBMS", Database Engineering, IEEE, Vol 5, No 4, September 1982, pp 28-32.

HARRI76 Harrison, M.A., W.L. Ruzzo, and J.D. Ullman, "On Protection in Operating Systems," Communications of the ACM 19, 8 (August 1976), pp 461-471.

HARTS81	Hartson, H., Rex; "Implementation of Predicate Based Protection in Multisafe" Technical Report CS80010R, Department of Computer Science, VPI&SU, Blacksburg, Va 24061.
HOFFM71	L.J. Hoffman, "The Formulary Model for Flexible Privacy and Access Controls", Proceedings of the AFIPS FJCC (1971), pp. 587-601.
JONES73	A.K. Jones, "Protection in Programmed Systems", PhD thesis, Carnegie Mellon, June 1973.
KERST81	Kersten, M.L; Reind P. van de Riet; and Wiebren de Jonge, "Privacy and Security in Distributed Data Base Systems", Second Seminar on Distributed Data sharing Systems, Amsterdam, June 3-5 1981, pp. 229-241.
LAMPB71	Lampson, B.W., "Protection", Proceeding of the Fifth Princeton Symposium on Information Science and Systems, Princeton University (March 1971), pp 437-443; reprinted in ACM SIGOPS Operating Systems Review 8,1 (January 1974), pp 18-24.
LIENT80	Lientz, Bennet, P; and Ira, R. Weiss, "Trade offs of Secure Processing in Centralized Versus Distributed Networks", Computer Networks 2, 1978, pp 35-43.
MARYA78	Maryanski, Fred J., "A Survey of Developments in Distributed Data Base Management Systems", Computer, February 1978, pp. 28-38.
MIRAN80	Miranda, Serge, "Aspects of Data Security in General Purpose Data Base Management Systems", IEEE 1980 Symposium on Data Security and Protection, April 14-16, 1980 Oakland, pp. 46-58.
MIRAN81	Miranda, Serge, "Final Report on Data Security", Second Seminar on Distributed Data Sharing, Amsterdam, June 3-5 1981, pp. 221-227.
NAUMA82	Nauman, John, "ENCOMPASS Evolution of a Distributed Database/Transaction System", Database Engineering, IEEE, Vol 5, No 4, September 1982, pp 37-41.
PELEG78	Pelagatti, G.; P. Paolini; and G. Bracki, "Mapping External Views to a Common Data Model ", Information Systems, 1978, Vol 3, pp 141-151.

POPES80 Popescu-Zeletin, R.; and H., Weber, "Some Considerations about Distributed Data Bases on Public Networks", in Distributed Data Bases; North Holland, C. Delobel and W.Litwin (eds), 1980, pp. 1-15.

RIET80 Van de Riet, Reind P.; Martin, L. Kersten, and Anthony I. Wasserman; " A Module Definition Facility for Access Control in Distributed Data Base Systems." Proceedings of the 1980 Symposium on Security and Privacy, April 14-16 1980, Oakland California, pp. 59-66.

ROTHN77 Rothnie, Jamea B.; and Nathan Goodman, "A Survey of Research and Developement in Distributed Database Management", Proceedings of the Third International Conference on Very Large Data Bases, 1977.

ROTHN80 J.B. Rothnie et al. , "Introduction to a System for Distributed Databases (SDD1)", ACM Transactions on Database Systems, Vol 5, No 1, March 1980, pp. 1-17.

SCHMI83 "Relational Database Systems" ed. by J.W. Schmidt, Brodie Michael, 1983.

SCHRE77 Schreiber, Fabio, A., "A Framework for Distributed Database Systems", International Computing Symposium 1977, North Holland Publishing Company, 1977, pp. 475-482.

SELIN80 Selinger, Patricia G.; "Authorization and Views", in Distributed Data Bases, Ed. I.W. Draffan and F.Poole, 1980, pp 233-296.

SPACC80 Spaccapietra, Stefano, "Heteregeneous Data Base Distribution", in Distributed Data Bases, ed. by I.W. Draffan and F.Poole, 1980, pp 155-193.

TOAN79 Toan, Nguyen Gia, "A Unified Method for Query Decomposition and Shared Information Updating in Distributed Systems", Proceedings of the First International Conference on Distributed Computing Systems, Huntsville, Alabama, October 1-5 1979, pp 679-685.

TOTH78 Toth, K.C.; S.A. Mahmoud; J.S. Riordon; and O. Sherif, "The ADD System: an Architecture for Distributed Databases", Proceedings of the

International Conference on Very Large Data Bases, 1978, pp. 462-471.

TRUEB80 Trueblood, Robert, P.; H., Rex, Hartson, and Johannes, J., Martin; " Multisafe A Modular Approach to Secure Data Base Management ", Technical Report, CS8008R, Department of Computer Science,VPI$SU, Blacksburg, Va 24061.

WILMS81 Wilms P.F.; and B.G. Lindsay, "A Data Base Authorization Mechanism Supporting Individual and Group Authorization", Second Seminar on Distributed Data Sharing Systems, Amsterdam, June 3-5 1981, pp. 273-292.

WITHI80 Withington, P. Tucker, "The Trusted Function in Secure Decentralized Processing ",Proceedings of the 1980 Symposium on Security and Privacy, April 12-14 1980, Oakland California, pp. 67-79.

WOOD79 Wood, C.; R.C. Summers; and E.B. Fernandez, "Authorization in multilevel Database models ", Information Systems, Vol 4, 1979 pp. 155-161.

WONG76 Wong, Eugene; Karel Youssefi, "Decomposition- A Strategy for Query Processing", ACM Transactions on Database Systems, Vol 1, No3, Sept 1976, pp 223-241.

YAO79 Yao, S. Bing, "Optimization of Query Evaluation Algorithms", ACM Transactions on Database Systems, Vol4, No 2, June 1979, pp 133-145.

ZIMME81 Zimmerman, Hubert; and Louis Pouzin, "The Standard Network Architecture developed by ISO", Data Communications and Computer Networks, CSI/IFIP, 1981, pp 1-15.