A PROTECTION MODEL INCORPORATING BOTH AUTHORIZATIONS AND CONSTRAINTS

by

Dr. Dennis Kafura
Dr. Atika Laribi

TR-85-30

August 1985

A PROTECTION MODEL INCORPORATING BOTH AUTHORIZATIONS AND CONSTRAINTS

Dr. Dennis Kafura
Dr. Atika Laribi

Department of Computer Science
Virginia Polytechnic Institute
Blacksburg, VA 24061

Abstract

This paper presents a powerful and flexible protection model which includes both authorizations of open systems and constraints of closed systems. In this model, rules of "inheritance" determine the authorizations which are created for new data derived by authorized computations from existing data. These rules create a middle–ground between purely discretionary and purely non–discretionary systems. Although the proposed protection model is quite general, it is presented in this paper in the context of a distributed relational database system. The core mechanisms of the model control access to all data bases including the authorization and constraint data bases themselves. It is, therefore, a self–regulating and integrated system. The power and flexibility of the model derive from its use of authorizations and constraints as two complementary and interrelated types of control. The tight protection provided by closed systems is maintained since constraints are defined only as a complement to authorizations and not as a substitute. An enforcement algorithm is given which shows how the effects of the authorizations and constraints can be efficiently realized. Among other applications, it is shown how this model provides a useful, partial answer to the question of safety decidability.

I. Introduction

In this paper we present a protection model incorporating both authorizations and constraints. In the usual sense, an authorization conveys the ability to perform some act. Constraints, however, impose limitations on how the authorizations may be used or distributed among users. The power and flexibility of this model derives from the use of authorizations and constraints as complementary forms of control and the manner in which the authorizations and constraints are integrated to form a self–regulating mechanism. The model we present, termed the ***hybrid model***, is used to solve two general protection problems which occur in many applications and acutely in distributed database managements systems (DDBMS). These two problems are: (1) controlling the information which may be derived from the database, and (2) regulating the flow of information (equivalently, transfer of rights) among users.

In the remainder of this section the two protection problems mentioned above will be explained in greater detail and the relevance of these problems to DDBMS will be explained. Also in this section other protection models are reviewed in the light of these two problems. Section II describes the structure of the proposed model. Section III formalizes the notion of "inheritance" which plays a critical role in the model's operation. Section IV presents a simple example of how the model can be used to solve the two protection problems for a small distributed database. Section V contains the algorithm used to enforce the model's protection rules. Section VI briefly explains how the model can be used to handle several other problems which arise in a distributed database system (e.g., views, revocation).

Two Protection Problems

The first of the two protection problems is how to allow authorizers to regulate the access to data not yet existing in the DDBMS but which the user can obtain through a series of database computations. Such a computation is termed a "serial computation". Data derived from the DDBMS, as illustrated in Figure 1, may be viewed as follows. Let:

R	be the set of all relations stored in the DDBMS.
$R(A)$	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 a serial computation.
$R(U)$	be the set of data user U has access to.
$D(R(U))$	be the set of data that can be derived from $R(U)$.

The intersection of $D(R(A))$ and $D(R(U))$ is the set that needs to be controlled by authorizer A. This set represents data not yet existing in the DDBMS, but whose derivation and/or

access 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.

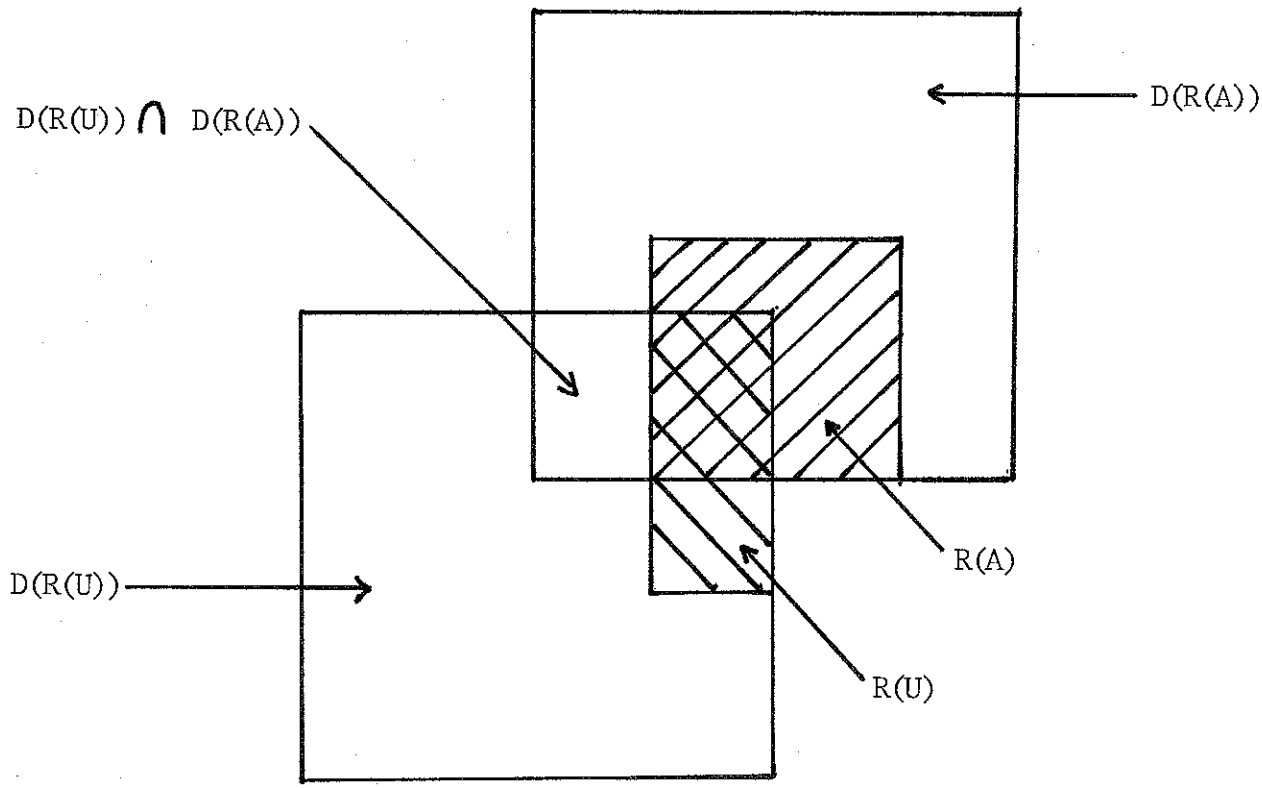

Figure 1. Derived Data

In protection systems using only authorizations a dichotomy arises between achieving precision at high overhead versus loosing control over derived data if this overhead cannot be tolerated. For example, the authorizer may grant minimal authorizations over $R(A)$. With this choice 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 every specific data base operation they require and the authorizer must appreciate all of the potential effects of granting or withholding the requested authorizations. This is a considerable overhead for both user and authorizer. On the other hand, authorizations can be defined over $R(A)$ which permits any serial computation using $R(A)$. 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 $D(R(U))$. The same dichotomy exists in systems using only constraints.

The second protection problem is how to control of information flow among users. A network user U_1 having an authorized access to some data R_1 could use R_1 in some, perhaps trivial, serial computation and give access to the result of this computation to another user U_2 not having any access to R_1 itself. The authorizer of R_1 should be able to control such a flow of information to user U_2. This is particularly important in DDBMS where the authorizer may not have complete information on the capabilities of each user.

The flow of information among users is handled in two basically different ways by systems which are termed either discretionary or non–discretionary. In discretionary systems the creator of data is allowed to use this data in any way and to grant any and all authorizations for the data. Typical discretionary models are the access matrix model [LAMPS71], authorizations lists and capabilities [GRAHA72], formularies [HOFFM71], and predicate–based models [HARTS76] [HARTS84] [TRUE83]. It is typically difficult to provide effective controls on the flow of information when a discretionary approach is used. Non–discretionary systems, on the other hand, assign a "classification" level to each object and a "clearance" level to each user. The classification of a computed object depends on the classifications of the objects from which it was derived and not on the user who created the object. Access to the newly created object is determined by the relationship between the object's classification and the user's clearance. The creator has no discretion to award or deny access. While it is easier to control the flow of information with non–discretionary techniques, this approach is usually quite rigid. For example, the classification levels must be hierarchical [BELL73] or form a lattice [DENNI82]. Such rigidity is not desirable.

The system that we propose is a hybrid one which combines the authorizations of open systems and the constraints of closed systems. The system also is a middle ground between discretionary and non–discretionary system since, via a concept of "inheritance", the protection rules that apply to a computed result are derived from the user–defined rules which applied to the data used to generate this result.

One illustration of the flexibility of the hybrid model is the following. Suppose that an authorizer grants to a user a set of authorizations allowing some stated access to the data which the authorizer owns and which is stored in the DDBMS. Also suppose that from this set of data the user 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.

The solution to these two protection problems is elaborated in the context of a distributed relational database management system (DDBMS). It is important to remember, however, that neither these problems nor the concept of combining authorizations and constraints is inherently limited to the database domain. A DDBMS is used to illustrate the protection problems for the following reasons. First, such systems are important applications which, in the most realistic cases, exhibit stringent protection requirements. Second, by contrast with other topics in DDBMS research, the security issue seems to be largely unsolved. Finally, a relational data model is used because most DDBMS prototypes have been relational ones [ADIBA82] [CERI82] [CHAN82] [HAAS82], and because of the claimed advantages of the relational model in a distributed environment [SCHMI83].

The two protection problems described above arise in DDBMS from two principal causes: scale and distributed authority. In addition to the sheer volume of information possible in a general DDBMS, the scale problem is also reflected in the diverse types of both information and users. This diversity makes it difficult to grant or revoke authorizations which conform to the legitimate needs of potential data users as well as safeguard against potential misuse of these authorizations. The distributed authority which results when different organizations are involved in a common DDBMS confuses the concepts of ownership and cost. When a database — distributed or otherwise — serves only one organization the data belongs , in the last instance, to the organization as a whole and the costs to collect and organize the data are borne by this single organization. In the case of distributed authority, it becomes much more important to control the transfer of costly data between organizations. The protection requirements of DDBMS are presented in more detail in [LARIB85].

Related Work

Protection models incorporating both authorizations and constraints have appear previously. Most recently Minsky and Lockman [MINSK85] use an actor based model to illustrate the power of combining both types of protection rules. Their work has proceeded independently, and apparently in parallel, with out own. While their approach is similar to ours, there are two major points of difference. First, they characterize their work as being " ... fairly abstract, in the sense that we do not assume any particular computational context, such as a programming language, or any specific application domain. Indeed, some of the options to be considered below for the various components of an obligation would not be practical in some computational contexts, and some would be useful only for certain kinds of applications. This lack of specificity of the various constructs means, unfortunately, that we cannot be very precise about the syntax and semantics of the various constructs which we introduce, or about their implementation. " [MINSK85,page 8]. In this paper we are much more specific about the use of authorizations and constraints

in a database domain. This allows us to clarify the semantic issues involved and to show the details of an enforcement algorithm. Second, the "inheritance" mechanism described in Section III has no counterpart in the Minsky and Lockman model. This mechanism plays a key role in solving the two protection problems described above. The concept of constraints has also been mentioned in [FERNA81], [ASTRA76], [BRACC79]. The later two of these were referred to in Minsky and Lockman.

The hybrid system also bears some resemblance to the "history–keeping" scheme proposed by Cohen [COHEN77]. In Cohen's scheme 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. Cohen does address the issue of derived data and gives a solution to the problem. However, we believe that the "tagging" algorithm, presented in Section V, is more efficient because less information has to be retained in the system. Cohen addressed only the issue of computational constraints, while the hybrid model contains a larger group of constraints including constraints on the flow of information between users and between sites. In addition, 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 data base.

Other researchers have also explored the more specific problem of DDBMS security. Bussolati [BUSSO80] proposed a scheme — based on the ANSI model — which allows for different data and security models in the same DDBMS. This approach seems overly ambitious. Because the design of Bussolati's system is heavily based on mappings between different data and security models, a considerable overhead is introduced. Furthermore, the application oriented approach of the ANSI model implies an inflexible system in which all application programs have to be known in advance. Finally, the control of information between sites or between users is not considered in Bussolati's work.

Kersten considered how to determine which authorizations apply when an object is accessed through a series of calls of programs owned by different subjects [KERST81]. The scheme which he proposes is extended to communicating DBMS, where the integration is minimal and the user is not given a unique view of the entire database. Only one very specific problem is considered in the work without placing 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.

The idea underlying the model proposed by Van de Riet and Kersten [RIET80] is that a unified approach — integrating programming languages and database management —

should be taken. A language–based scheme can be quite powerful when a uniform environment exists. Our view, however, is that in a DDBMS involving different organizations, a single language assumption is not realistic. Instead we make the weaker assumption that only a common, and fairly neutral, data model is required.

A generalization of the authorization mechanism of system R to the distributed case is given in [WILMS81]. One of the basic policies used in this extended system is that each site controls access to the objects it stores, and also controls the granting and revocation of access rights to those objects. This specific policy is only one of several which can be supported by the hybrid system we propose. Other policy alternatives are considered in [LARIB85].

II. Authorizations and Constraints

The driving mechanism of the hybrid system is the authorization-constraint mechanism. This mechanism is represented in Figure 2 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. The authorizations format is derived from Hartson's predicate–based model [HARTS76] [HARTS84]. The authorizations are themselves stored in a relation with the domains defined below. The following two types of authorizations are defined, depending on the type of operations authorized.

$$AUTO = (A, U, O, R, B, S, C)$$

for the access operations $READ$, $WRITE$, $UPDATE$, and $DELETE$, and

$$AUTC = (A, U, J, R_1, R_2, B_1, B_2, S, C)$$

for $JOIN$ operations, where:

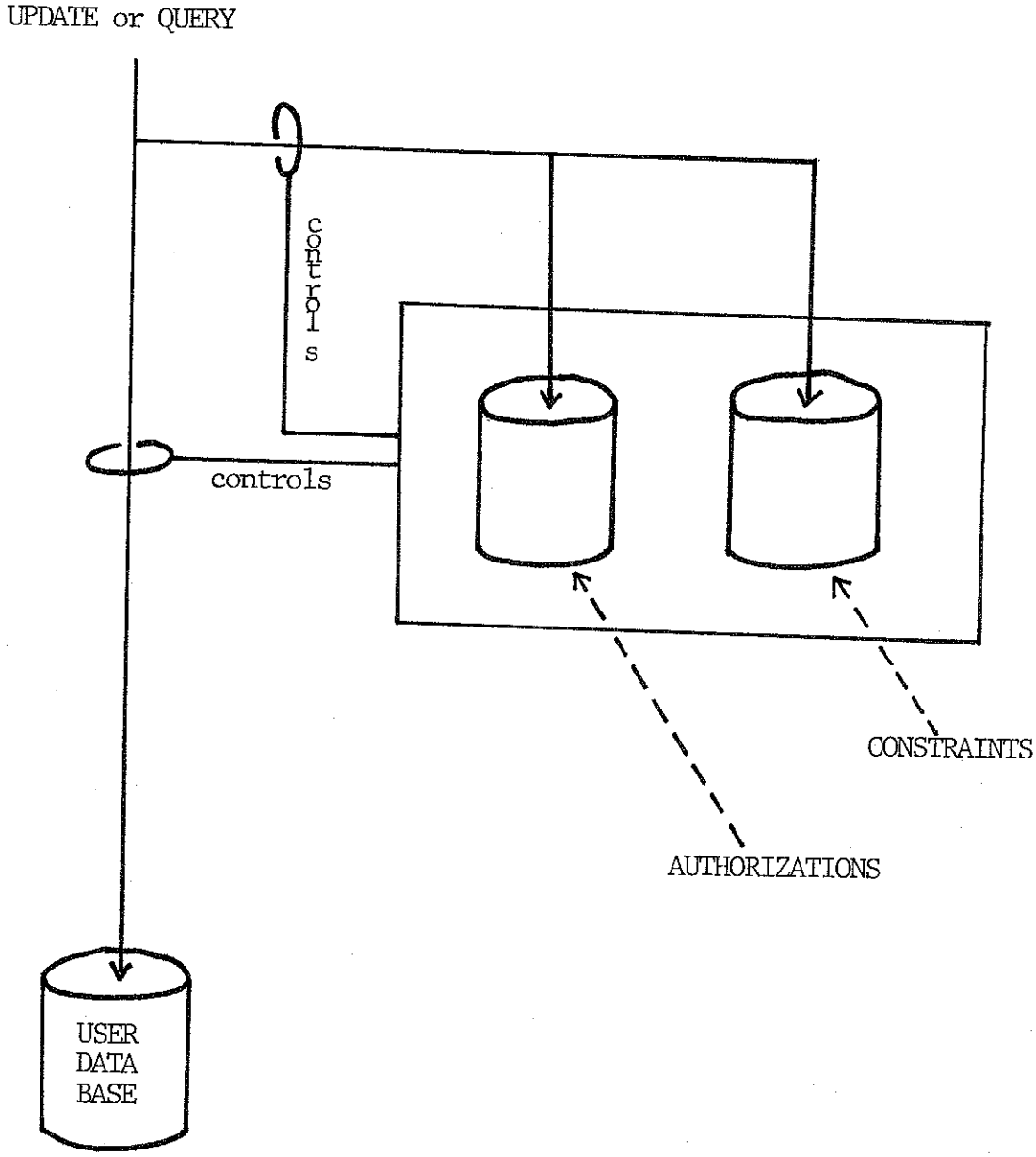

Figure 2. Authorization-Constraint Mechanism

A	is the authorizer's name
U	is the user's name
O	is the 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.
R_1, R_2	are the relations U is authorized to *JOIN*.
B, B_1, B_2	are the bit coded fields used to define PROJECTIONS on the data fields. 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.
C	is a constraint set defining limitations on the users ability to perform actions using this authorization.

The following notation will be used to refer to the various fields of a specific authorization. For the authorization *AUT*, *A*(*AUT*) will be used to denote the name of the authorizer for *AUT*, *O*(*AUT*), will denote the operation being authorized, etc.. In general, an "*" can replace any of the fields in an authorization (or constraint) to specify that the authorization (or constraint) is to be effective for any value of that specific field. The system conditions can mention the user's site, the data site, or network characteristics such as the traffic status or the state of each node operating.

Recall that the authorizations can be defined on any of the data bases maintained by the system including the main user database, the authorization database, or the constraint database. The database to which an authorization refers could be explicitly named by an additional field in the authorization. However, for simplicity, we will assume that the relation names are unique across all of the databases. Therefore, the relation name will imply the intended database. As suggested in Figure 2 the names *AUTHORIZATIONS* and *CONSTRAINTS* will be used to refer to the relations which are the databases used by the hybrid model itself.

The second major element of the hybrid model is the constraint. Each type of constraint specifies an action which is not allowed to occur regardless of any authorizations which may permit that action. The format of constraints is very similar to that of authorizations. The domains defined in the constraints are the same as the ones in authorizations.

There are several types of constraints defined in [LARIB85]. Some of these constraints control the user's rights to perform access operations and join operations. Other constraints control the transmission and storage of data among sites in the distributed database system. In this paper attention will be focussed on the two most important types of constraints.

The first form of constraints, computational constraints, allow for the control of data derived by users through authorized computations. This form of constraint prevents a user from obtaining two domains together in a computed relation. These constraints have the following format:

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

where D_1 and D_2 are the domains which cannot appear together in any serial computation. Computational constraints are implemented by associating the constraints with each relation containing either one of the two domains named in the constraint. In particular, when presented with a computational constraint, the hybrid model will associate the constraint (A, D_1, D_2, S) with every authorization for every relation containing domain D_1 and it will associate the dual constraint (A, D_2, D_1, S) with every authorization for every relation containing domain D_2. When a query is submitted for processing the enforcement algorithm described in Section V will form the set of all applicable constraints. The query is rejected if this set contains both a computational constraint and its dual.

The second form of constraints, termed flow constraints, regulate the transfer of rights, and therefore information, between users. Flow constraints have the following format:

$$CONF = (A, R, O, U_1, U_2, S)$$

where authorizer A specifies that user U_1 is not allowed to authorize user U_2 to perform operation O on relation R under the system condition S. Using flow constraints data may be protected by not allowing users to distribute rights (information) which they themselves have obtained from other users. Without some safeguard the original authorizer, who might not approve of the attempted transfer, would be unaware of and unable to prevent the transfer.

Different and flexible relationship among users can be defined using flow constraints. 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 great number of types of users.

The implementation of flow constraints makes use of the integrated nature of the hybrid model. A right is transferred from user U_1 to user U_2 when U_1 performs an *UPDATE* operation on the *AUTHORIZATIONS* relation. However, as with all operations, $U_1's$ ability to succeed in this *UPDATE* is conditioned by constraints associated with the authorizations which U_1 possesses. When a flow constraint is presented to the hybrid model, the reduced constraint (A, O, U_2, S) is associated with every authorization which U_1 has to perform operation O on relation R. When a transfer of rights is attempted the enforcement algorithm presented in Section V will gather together all authorizations which U_1 possesses for the relation named in the transfer. If any of these authorizations is associated with a constraint prohibiting the transfer, then the transfer (i.e., the *UPDATE* of the *AUTHORIZATIONS* relation) is rejected.

III. Authorization and Constraint Inheritance

The relation resulting from a user's query can be stored in the database for later use. New authorizations and constraints have to be associated with this computed relation. The process by which these authorizations and constraints are derived is referred to as the inheritance process.

The inheritance algorithm makes use of a concept called "tagging". An authorization is said to be tagged when a constraint is associated with it. The effect of this constraint is preserved across computations by the inheritance mechanism because authorizations for a computed relation are tagged with the union of the constraints associated with the relations from which the result was derived. Using the inheritance process and the implementation of the computational constraint enforcement presented in the Section V, the authorizers are guaranteed that the limitations they put on the use of their data are respected even, and especially, 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. In particular, 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.

The policy used for the inheritance process is that a user deriving 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. However, such ownership is conditioned because all the constraints associated with the data used in the derivation are inherited and will take effect whenever the computed relation is used in subsequent computations. The user is also authorized to grant other users *READ*, *WRITE*, *UPDATE*, and *DELETE* rights over the new relation provided no constraint restricting this transfer was inherited. The user also inherits *JOIN* rights depending on the *JOIN* rights used in the derivation. A *JOIN* right between the new relation and R_i is granted if and only if the user the right to *JOIN* R_i with all relations used to compute *CR*. The inheritance process is formalized below, followed by an example.

Let $CR = f_{U_1}(R_1, R_2, \ldots, R_n)$ express that CR is derived by user U_1 through a computation involving relations R_i for $i = 1, \ldots, n$. Authorizations for *CR* are created and inherit all computational and flow constraints associated with any of the R_i. To express formally the inherited constraints, we first have to define the sets of authorizations and constraints associated with the $R_i's$. Let:

$$AAUT(R_i) = \{AUT_j \mid R(AUT_j) = R_i \quad and \quad U(AUT_j) = U_1\}$$

be the set of access authorizations for R_i.

$$JAUT(R_i) = \{AUT_j \mid (R_1(AUT_j) = R_i \quad or \quad R_2(AUT_j) = R_i) \quad and \quad U(AUT_j) = U_1\}$$

be the set of JOIN authorizations for R_i. Now let

$$UCCONSET(R_i) = \{C(AUT_j) \mid AUT_j \in AAUT(R_i)\}$$

be the set of all constraints associated with the access authorizations for R_i,

$$UJCONSET(R_i) = \{C(AUT_j) \mid AUT_j \in JAUT(R_i)\}$$

be the set of all constraints associated with the *JOIN* authorizations for R_i, and

$$UCONSET(R_i) = UCCONSET(R_i) \bigcup UJCONSET(R_i)$$

be the set of all computational and flow constraints imposed on U for the use of R_i. Finally, the set of constraints which are then imposed on *CR* is the union of all the constraints imposed on the various $R_i's$:

$$CONSET(CR) = \bigcup_i UCONSET(R_i).$$

Then the following authorizations giving U_1 *READ*, *WRITE*, *UPDATE*, and *DELETE* access to *CR* are created:

$$AUT_R(CR) = (DBA, U_1, READ, CR, *, CONSET(CR))$$
$$AUT_W(CR) = (DBA, U_1, WRITE, CR, *, CONSET(CR))$$
$$AUT_U(CR) = (DBA, U_1, UPDATE, CR, *, CONSET(CR))$$
$$AUT_D(CR) = (DBA, U_1, DELETE, CR, *, CONSET(CR)).$$

The *JOIN* authorizations created for relation CR are derived from the *JOIN* authorizations which exist for the $R_i's$. An authorization for the join of CR with relation R_k is created if U_1 has *JOIN* rights for R_k and R_i for all $R_i's$ used in computing CR. Formally, if there exist join authorizations AUT_{ki} where:

$$R_1(AUT_{ki}) = (R_k \quad and \quad R_2(AUT_{ki}) = R_i) \quad or \quad (R_1(AUT_{ki}) = R_i \quad and \quad R_2(AUT_{ki}) = R_k)$$

for each $R_i \in CR$, then create a join authorization of the form:

$$AUT(CR) = (DBA, U_1, J, CR, R_k, *, CONSET(CR)).$$

This completes the set of authorizations created for computed relations. Note that 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_1 obtains on CR. That is, the state conditions are not inherited.

The inheritance process is powerful enough to provide authorizers with a tool for an efficient control of the data the user derives from the database 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_1 is under the control of authorizers A_1, A_2, and A_3. 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 A_1 grants user U_1 an access O_1 to R_1 and A_2 grants U_1 access O_2 on R_1, then whenever a constraint on the use of R_1 is imposed by either A_1, A_2 or A_3, it has to be reflected on all authorizations granted to U_1 for R_1

by either authorizer. This is done by tagging all authorizations for the access of R_1 by U_1 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_1 is replicated and controlled by different agencies and one agency does not allow U_1 to obtain R_1 from its data base but does not object to U_1 obtaining it from another source in the network. In this case each authorizer performs a different control over the data. Lets suppose that authorizer A_1 grants authorization AUT_1 to U_1 over R_1, and authorizer A_2 grants authorization AUT_2 to U_1 over R_1, and that these are the only authorizations granted to U_1. Lets further suppose that A_1 imposes a constraint $CONS_1$ on the use of R_1 by U_1. This constraint will take effect only if U_1 uses AUT_1 to access R_1. Using the tagging algorithm, only AUT_1 will be tagged by the constraint $CONS_1$. 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.

IV. Example

To illustrate how the security requirements for DDBMS can be satisfied with the proposed system we will present an example of a small distributed database and a number of access control rules including both authorizations and constraints. As shown in Figure 3, this database is composed of four relations distributed among three sites.

Site	Relation	Domains
Payroll Office	Employee	SSN, Name, Dept#
	Department	Dept#, Name of Dept., Code# of Head, Address of Head
Bank	Account	Account#, Code#, Balance, Address
University Registrar	Course	Course Name, SSN of Instructor, Address of Instructor

Figure 3: Example Distributed Data Base

An employee in the payroll office,U_1, is given access to these four relations and their respective joins by means of twelve authorizations. These authorizations are granted to user U_1 by the payroll manager (PM), the bank manager (BM), the university registrar

(UR), or by the database administrator (DBA):

$$AUT_1 = (DBA, U_1, R, Employee, 111, P_1, \phi)$$
$$AUT_2 = (PM, U_1, RW, Employee, 011, P_2, \phi)$$
$$AUT_3 = (DBA, U_1, R, Department, 1101, P_3, \phi)$$
$$AUT_4 = (PM, U_1, RW, Department, 1101, P_4, \phi)$$
$$AUT_5 = (DBA, U_1, R, Account, 1111, P_5, \phi)$$
$$AUT_6 = (UR, U_1, RW, Course, 111, *, P_6, \phi)$$
$$AUT_7 = (PM, U_1, J, Employee, *, 111, P_7, \phi)$$
$$AUT_8 = (DBA, U_1, J, Employee, *, 111, P_8, \phi)$$
$$AUT_9 = (DBA, U_1, J, Department, *, 1111, P_9, \phi)$$
$$AUT_{10} = (PM, U_1, J, Department, *, 1111, P_{10}, \phi)$$
$$AUT_{11} = (BM, U_1, J, Account, *, 1111, P_{11}, \phi)$$
$$AUT_{12} = (UR, U_1, J, Course, *, 111, P_{12}, \phi)$$

where ϕ in the last field indicates that there are no restrictions put on the authorizations at this point.

This set of authorizations and constraints gives U_1 read and write access to the relations *Employee*, *Department*, and *Course*, and write access to the relation *Department*. U_1 is also given the rights to join relations *Employee*, *Department*, *Account*, and *Course* with any other relation. The authorizations are to take effect when their respective access conditions are satisfied.

We will not spell out all the access conditions to avoid unnecessary details for the example. As an illustration we will give possible interpretations for predicates P_1 and P_2. P_1, for example, might expresses that the terminal used should be a terminal at the Payroll office. P_2 might specifies that the authorization is to take effect only if the relation Employee, is accessed by U_1 between 8:00 a.m. and 5:00 p.m.

Suppose however, that the data base administrator wants to prevent U_1 from obtaining the *Name* and the *Balance* or the *SSN* and *Account* of any employee. U_1 could obtain the *Name* and *Balance* of an employee in at least two different ways. By 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 heads of 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* allows the user to derive the *Name* and *Balance* of instructors.

Using computational constraints the desired constraints can be expressed as:

$$CONC_1 = (DBA, Name, Balance, P_{21})$$
$$CONC_2 = (DBA, Balance, Name, P_{21})$$
$$CONC_3 = (DBA, SSN, Account, P_{22})$$
$$CONC_4 = (DBA, Account, SSN, P_{22})$$

$CONC_1$, for example, would be associated with every relation containing the domain *Name* and would prevent such relations from appearing in *JOIN* operations with any other relation which contained the domain *Balance*. $CONC_2$ is the dual of $CONC_1$. Recall that any computation which brings together a constraint and its dual will be rejected.

The authorizations would then be tagged as follows:

$$AUT_1 = (DB, U_1, R, Employee, 111, P_1, \{CONC_1\})$$
$$AUT_2 = (PM, U_1, RW, Employee, 011, P_2, \{CONC_1\})$$
$$AUT_5 = (DB, U_1, R, Account, 1111, P_5, \{CONC_2, CONC_3\})$$
$$AUT_6 = (UR, U_1, RW, Course, 111, *, P_6, \{CONC_4\})$$
$$AUT_7 = (PM, U_1, J, Employee, *, 111, P_7, \{CONC_1\})$$
$$AUT_8 = (DB, U_1, J, Employee, *, 111, P_8, \{CONC_1\})$$
$$AUT_{11} = (BM, U_1, J, Account, *, 1111, P_{11}, \{CONC_2, CONC_3\})$$
$$AUT_{12} = (UR, U_1, J, Course, *, 111, P_{12}, \{CONC_4\})$$

Lets further suppose that U_1 performs a *JOIN* of relations *Employee* and *Course* on domain SSN, obtaining the relation CE. Authorizations AUT_7 and AUT_{12} are used to allow the *JOIN* to be performed. An authorization AUT_{13} is granted to U_1, and the tags of AUT_7 and AUT_{12} are transmitted to AUT_{13} as:

$$AUT_{13} = (DBA, U_1, RW, CE, 11111, *, \{CONC_1, CONC_4\})$$

If U_1 later requests a *JOIN* of relations CE and *Account* on the domain *Address* to be performed, the query is rejected because it violates constraint $CONC_1$ which is tagged in the authorization for CE.

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:

$$AUT_{14} = (DBA, UB, RW, Account, 1111, *, \phi)$$

The following flow constraint is then defined:

$$CONC_5 = (DBA, Account, RW, UB, UP, *).$$

and AUT_14 is tagged with $CONC_5$. This constraint prevents UB from transferring $READ$ or $WRITE$ permission on the relation *Account* to user UP.

V. Enforcement Algorithm

This section contains the description of an algorithm which dynamically enforces the rules defined by authorizations and constraints. The algorithm is termed "dynamic" because it is applied at the time a database transaction is executed. A more restrictive "static" algorithm — one which is applied when a constraint is defined — is presented in [LARIB85]. The algorithm has two phases: the authorization enforcement and the constraint enforcement.

The authorization enforcement phase is based on Hartson's algorithm [HARTS76] [HARTS84]. 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. Moreover, 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.

The authorization enforcement phase of the algorithm consists of steps (1)–(6). Steps (1) and (2) of the algorithm find all of the authorizations which are relevant to the query. The query may be rejected in step (3) if some domain in the result is not authorized. The check made in step (4) may reject a join query if insufficient authorizations are available to conduct the required join operations. Finally, steps (5) and (6) evaluate the system predicates for all of the authorizations and reject the query if the predicate is false. The constraint enforcement phase consists of steps (7)–(10). Steps (7) and (8) collect all of the constraints applicable to the query. The computational constraints are applied in step (9) and the flow constraints are applied in step (10). The query is rejected in steps (9) or (10) if query would violate any of the constraints.

For the purpose of presenting the enforcement algorithm, a query is assumed to have one of two, somewhat simplified, forms. The first form is taken by queries involving access operations. In this case:

$$Q = (U, O, R, T, \{D_1, D_2, \ldots, D_m\})$$

where the access operation O is applied to relation R by user U. In the case of READ operations $D_1, \ldots, D_m$ represent the domains projected in the result. For WRITE, UPDATE, and DELETE operations T is a (set of) tuple(s) relevant to the operations. The second form of a query is:

$$Q = (U, J, \{R_1, R_2, \ldots, R_n\}, \{D_1, D_2, \ldots, D_m\})$$

denoting that join operations are performed among relations $R_1, \ldots, R_n$ and projecting domains $D_1, \ldots, D_m$ to form the result. As before the notation $R(Q)$ or $R_i(Q)$ will be used to denote a relation named in the query, $U(Q)$ will denote the user who is making the query, etc..

The following algorithm is executed each time a query **Q** is submitted to the system.

(1) ***Find all authorizations for user*** $U(Q)$. This is called the franchise of the user and is denoted

$$F = \{AUT_i \quad | \quad U(Q) = U(Aut_i)\}.$$

(2) ***Select authorizations for query operations.*** Find in F the authorizations that have relations appearing in the query and are for the operation specified in the query. For queries involving access operations this is expressed as:

$$G = \{AUT_i \quad | \quad AUT_i \in F \quad \text{and} \quad O(AUT_i) = O(Q) \quad \text{and} \quad R(AUT_i) = R(Q)\}$$

while for queries involving join operations this is expressed as:

$$G = \{AUT_i \quad | \quad AUT_i \in F \wedge O(AUT_i) = JOIN \wedge (R_1(AUT_i) \in R(Q) \vee R_2(AUT_i) \in R(Q))\}$$

(3) ***Check that all projected domains are accessible.*** That is determine if "for all D_i in $D_1, \ldots, D_m$ there is an AUT_j in G such that $R(AUT_j) = R_n$ and $B^k(AUT_j) = 1$ where D_i is the *kth* domain in relation R_n and B^k denotes the value of the *kth* bit in the bit field B of the authorization. The query is rejected if one or more domains are not accessible.

(4) *Check join rights.* If the query is a join query then check that there are authorizations in G to allow the JOINS with all other relations involved in the query. Evaluate "for all R_i in Q there is AUT_{i_k} in G such that $R_1(AUT_{i_k}) = R_i$ and $R_2(AUT_{i_k}) = R_k$ for $k = 1,\ldots,n$ and $k \neq i$". Reject the query if not all combinations of joins are authorized.

(5) *Determine equivalence classes.* Partition G into equivalence classes by having two authorizations be in the same equivalence class if they authorize an operation on the same relation

$$G_i = \{AUT_j \quad | \quad AUT_j \in G \quad \text{and} \quad R(AUT_j) = R_i\}.$$

(6) *Evaluate the effective access condition for the query.* This 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. Formally:

$$EAC_i = \bigvee_j S(AUT_j)$$

for each AUT_j in G_i. Then, the effective access condition is:

$$EAC = \prod_k EAC_k$$

The query is rejected if does not satisfy the effective access condition.

(7) *Find all constraints.*

$$CONC = \{CONC_i \quad | \quad AUT_j \in G \quad \text{and} \quad CONC(AUT_j) = CONC_i\}$$

(8) *Select all effective constraints.* Evaluate the access condition of each constraint in CONC and eliminate the ones which do not satisfy their access condition.

$$ECON = \{CONC_i \quad | \quad CONC_i \in CONC \quad \text{and} \quad CONC_i = true\}$$

(9) *Enforce the computational constraints.* For each computational constraint evaluate if the two domains specified in the constraint appear together in the result of the query. If it is true for at least one constraint then the query is rejected.

Formally, reject the query if there exists two constraints $CONC_i$ and $CONC_j$ in $ECON$ for which $D_1(CONC_i) = D_2(CONC_j)$ and $D_2(CONC_i) = D_1(CONC_j)$. and where both D_1 and D_2 are in the projected domains of the query.

(10) ***Enforce the flow constraints.*** The flow constraints are enforced when $O(Q) = UPDATE$ and $R(Q) = AUTHORIZATION$. In this case:

(a) Use steps (1)–(8) to find the effective constraint set, $ECON'$, for $U(Q)$ to perform the operation $O(T)$ on relation $R(T)$.

(b) The update is rejected if there exists a flow constraint $C_i \in ECON'$ such that $U(C_i) = U(T)$ and $O(C_i) = O(T)$. If the update is not rejected then allow the update modifying the authorization being written so that $C(T) = C(T) \cup ECON'$. This modification accounts for the inheritance of constraints.

Example

To illustrate the enforcement algorithm the authorizations of our previous example are considered again. The complete set of authorizations and constraints is repeated here for convenience:

$$AUT_1 = (DBA, U_1, R, Employee, 111, P_1, \{CONC_1\})$$
$$AUT_2 = (PM, U_1, RW, Employee, 011, P_2, \{CONC_1\})$$
$$AUT_3 = (DBA, U_1, R, Department, 1101, P_3, \phi)$$
$$AUT_4 = (PM, U_1, RW, Department, 1101, P_4, \phi)$$
$$AUT_5 = (DBA, U_1, R, Account, 1111, P_5, \{CONC_2, CONC_3\})$$
$$AUT_6 = (UR, U_1, RW, Course, 111, *, P_6, \{CONC_4\})$$
$$AUT_7 = (PM, U_1, J, Employee, *, 111, P_7, \{CONC_1\})$$
$$AUT_8 = (DBA, U_1, J, Employee, *, 111, P_8, \{CONC_1\})$$
$$AUT_9 = (DBA, U_1, J, Department, *, 1111, P_9, \phi)$$
$$AUT_{10} = (PM, U_1, J, Department, *, 1111, P_{10}, \phi)$$
$$AUT_{11} = (BM, U_1, J, Account, *, 1111, P_{11}, \{CONC_2, CONC_3\})$$
$$AUT_{12} = (UR, U_1, J, Course, *, 111, P_{11}, \{CONC_4\})$$
$$CONC_1 = (DBA, Name, Balance, P_{21})$$
$$CONC_2 = (DBA, Balance, Name, P_{21})$$
$$CONC_3 = (DBA, SSN, Account, P_{22})$$
$$CONC_4 = (DBA, Account, SSN, P_{22})$$

Suppose at time t user U_1 submits query Q_1 requesting a list of *SSN*, *Name*, *Balance*, and *Address* of all instructors.

$$Q_1 = (U_1, J, \{Employee, Course, Account\}, \{SSN, Name, Balance, Address\})$$

Also suppose that at time t the following access conditions are true P_1, P_4, P_5, P_6, P_7, P_{10}, P_{11}, and P_{21}, while all others are false. The query is evaluated as follows:

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

(2) $G = \{AUT_7, AUT_8, AUT_{11}, AUT_{12}\}$

(3) all projected domains are accessible through authorizations in G.

(4) AUT_7, AUT_8, AUT_{11}, and AUT_{12} give rights to *JOIN Employee*, *Account*, and *Course* with any other relation.

(5) The equivalence classes are:

$$G_1 = \{AUT_7, AUT_8\}$$
$$G_2 = \{AUT_{11}\}$$
$$G_3 = \{AUT_{12}\}$$

(6) evaluate the effective access condition:

$$(P_7 \vee P_8) \wedge (P_{11}) \wedge (P_{12})$$

which is true.

(7) $CONC = \{CONC_1, CONC_2, CONC_3, CONC_4\}$

(8) $ECON = \{CONC_1, CONC_2\}$

(9) *Name* and *Balance* appear together in the projected domains. This violates $CONC_1$ and $CONC_2$. The query is therefore rejected.

Suppose now that a later time t' user U_1 requests a list of *Names*, *Departmentname*, and *Courses* for all instructors. This is represented by the query:

$$Q_2 = (U_1, J, \{Course, Employee, Department\}, \{Name, Department, Course\}$$

Suppose that at t' the access conditions have the same values as at time t. Q_2 is then processed as follows:

(1) the franchise of U_1 is composed of all 12 authorizations.

(2) $G = \{AUT_7, AUT_8, AUT_9, AUT_{10}, AUT_{12}\}$

(3) all projected domains are accessible through authorizations in G.

(4) $AUT_7, AUT_8, AUT_9, AUT_{10}$ authorize $JOINS$ of *Course*, *Employee*, and *Department* to be made with any other relation.

(5) the equivalence classes are:

$$G_1 = \{AUT_7, AUT_8\}$$
$$G_2 = \{AUT_9, AUT_{10}\}$$
$$G_3 = \{AUT_{12}\}$$

(6) evaluate the effective access condition as:

$$(P_7 \vee P_8) \wedge (P_9 \vee P_{10}) \wedge P_{12}$$

which is true.

(7) $CONC = \{CONC_1, CONC_4\}$

(8) $ECONC = \{CONC_1\}$

(9) Since $NAME$ and $BALANCE$ do not appear together in the projected domains, $CONC_1$ is not violated and the query is accepted.

The result of Q_2 may be stored in the DDB for later use. If U_1 then asked the $JOIN$ of this resulting relation and the relation *Account* to be made, the constraint would be violated.

Efficient Implementation

It should be noted that step (9), which corresponds to the computational constraints evaluation, has been purposefully left abstract. This was done to avoid introducing unnecessary implementation details to the algorithm presentation and to the example. We will now study in a little more detail the implementation of step (9) 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:

$$CONC_1 = (A, U, N, S)$$

and

$$CONC_2 = (A, U, -N, S)$$

All authorizations giving access to D_1 are then tagged with $CONC_1$ and all authorizations giving access to D_2 are tagged with $CONC_2$. 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 correspondence acts as an interface between the constraint imposer and the system.

Step (9) of the algorithm can then be replaced by the following three steps:

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

$$ECONP = \{CON_i | TAG(CON_i) > 0\}$$
$$ECONN = \{CON_i | TAG(CON_i) < 0\}$$

(9.2) Order ECONP and ECONN.

(9.3) 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. That is, 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:

$$CONC_1 = (DBA, 1, P_{21})$$
$$CONC_2 = (DBA, -1, P_{21})$$
$$CONC_3 = (DBA, 2, P_{22})$$
$$CONC_4 = (DBA, -2, P_{22})$$

The following table of correspondence is also created:

Unique Integer	Domain 1 Name	Domain 2 Name
1	Name	Balance
2	SSN	Account

When query Q_1 is submitted step 13 of the preceding algorithm is replaced by the following steps:

(9.1) $ECONP = \{CONC1\} and ECONN = \{CONC2\}$.

(9.2) Both $ECONP$ and $ECONN$ are already ordered.

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

Query Q_2 is still accepted since $ECONN$ is empty in this case.

VI. 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 several examples of such problems. These problems are typically difficult to solve with systems that use either only authorizations or only constraints.

Decidability of the Safety Problem

The set of tools offered to the authorizer by the hybrid system provides for a limited, practical 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_1 cannot obtain a certain right O_1 over data R_1 if the authorizer of R_1 did not intend U_1 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_1 grants a right to a user U_1 over data R_1, but does not want to allow U_1 to use R_1 in computations and then transfer some rights over the computed data to user U_2. To do so A has only to write a flow constraint expressing that U_1 cannot transfer any right over R_1 under any form to U_2. The inheritance process assures that the constraint will be transmitted along with any computations performed on R_1.

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 nondiscretionary 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.

Multiple Authorizers

This problem occurs when when a users wishes to obtain a join authorization between relations which are controlled by two 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 A_1 controls relation R_1 and authorizer A_2 controls relation R_2. Lets further suppose that U_1 needs to be granted *JOIN* right over the two relations. A1 has then to introduce the partial authorization:

$$PAUT_1 = (A1, U, J, R_1, R_2, A2, S1)$$

which indicates that A_1 authorizes U_1 to *JOIN* R_1 with R_2 provided U_1 obtains the same right from A_2. Similarly A_2 has to grant U_1 the following partial authorization:

$$PAUT_2 = (A2, U, J, R_2, R_1, A1, S2).$$

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

$$AUT = (A1 \bigcup A2, U, J, R_1, R_2, S1 \vee S2).$$

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.

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 no longer 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 authorization 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_1 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_1 would need access to *Account*, *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 hybrid system can also be used to define different types of views of the data base for various groups of users. This is illustrated by the following possible views which can be defined:

- 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_1 without giving U_1 the right to access either the *Employee* relation or the relation *Department*. An authorization granting U_1 access to the *RESTRICTION* on domains *Name* and *Department* 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 a 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.

VII. 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 protection issues, particularly those protection issues which arise in DDBMS. We identified the main protection issues and requirements associated with DDBMS and presented a system satisfying these requirements. This system possesses a certain number of important characteristics:

- The hybrid system proves to be flexible and of practical value since it can easily accommodate different types of environment represented by the different possible combinations of use of the main protection tools, i.e. authorizations and constraints. Fur-

thermore, 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.

- 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 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 core 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.

BIBLIOGRAPHY

[ADIBA82] Adiba, Michel, "Distributed Data Base Research at Grenoble University", *Database Engineering*, IEEE, September 1982, Vol 5, No 4, pp 2-8.

[ASTRA76] Astrahan, M. M. et. al., "System R: Relation Approach to Data Base Management," *ACM Transactions on Database Systems*, June, 1976.

[BRACC79] Bracchi, G., Furtado, A., Pelaggatti, G., "Constraint Specification in Evolutionary Data Base Design", in *Formal Models and Pactical Tools for Information Systems Design*, Schneider, H.J. (editor), North Holland, 1979, pages 149-164.

[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.

[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, Computer Science Dept., Ohio State University, 1977.

[DENNI82] Denning, D.E., *Cryptography and Data Security.* Addison Wesley Publishing Co., 1982.

[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.

[GRAHA72] Graham, G.S.; and P.J. Denning, "Protection—Principles and Practice", in *Proc. Spring Joint Computer Conference*, Vol. 40, AFIPS Press, Montvale, N.J., 1979, pages 417–429.

[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*, Vol. 19,No. 8, August 1976, pp 461-471.

[HARTS76] Hartson, H.R., and Shiao, D.K., "A Semantic Model for Database Protection Languages," *Proceedings: International Conference on Very Large Databases,* Brussells, September 1976.

[HARTS84] Hartson, H., Rex; "Implementation of Predicate Based Protection in Multisafe", *Software: Practice and Experience,* Vol. 14, No. 3, March 1984, pp.207–234.

[HOFFM71] L.J. Hoffman, "The Formulary Model for Flexible Privacy and Access Controls", *Proceedings of the AFIPS FJCC,* 1971, pp. 587-601.

[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.

[LARIB85] Laribi, Atika, *A Protection Model for Distributed Data Base Management Systems*, PhD Thesis, Computer Science Department, Virginia Tech, 1985.

[LAMPS71] 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*, Vol. 8,No. 1, January 1974, pp 18-24.

[MINSK85] Minsky, N.H.; and A. Lockman, "Extending Authorization by Adding Obligations to Permissions", *Technical Report LCSR–TR–67*, Laboratory for Computer Science Research, Rutgers University, January 1985.

[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.

[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.

[TRUEB80] Trueblood, Robert, P.; H., Rex, Hartson, and Johannes, J., Martin; "Multisafe — A Modular Multiprocessor Approach to Secure Data Base Management", *ACM Transactions on Database Systems,* Vol. 8, No. 3, September 1983, pp.382–409.

[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.