# 16

# Assured Discretionary Access Control for Trusted RDBMS[1]

Marvin Schaefer[a] and Gary Smith[b]

[a]Arca Systems, Inc., 10320 Little Patuxent Parkway, Suite 1005, Columbia, MD 21044 USA

[b]Arca Systems, Inc., 8229 Boone Blvd., Suite 610, Vienna, VA 22182 USA

**ABSTRACT**

This paper investigates the problem of providing high levels of assurance for view-based discretionary access control mechanisms in multilevel DBMS. Assurance is considered from the effectiveness and correctness perspectives, and the gained insight is used to revisit issues of requirements, safety, policy, and mechanism. Three distinct view-based policies are introduced, and two should satisfy B2, B3 and A1 TCSEC requirements.

## 1. Introduction

In this paper, we define three distinct discretionary access control policies for multilevel relational database management systems. The three policies have been examined in order to understand the issues underlying the possibility of their incorporation in a high-assurance trusted DBMS architecture that we believe could satisfy the B2, B3 or A1 requirements of the *Trusted Computer System Evaluation Criteria (TCSEC)* and the *Trusted Database Management System Interpretation (TDI)* or their "equivalent". This paper reports on the conclusions of the authors in the study described by [11].

The primary area of concern for achieving these ratings, beyond the requirements identified for class B1, is to define a security architecture that satisfies the specific TCSEC correctness assurance requirements for:

- **B2**: internal structure and modularity, precise semantics, separation of protection critical from non-protection-critical execution domains, least privilege, and consistency with formal model and DTLS
- **B3, A1**: conceptual simplicity, precise semantics, modularity, least privilege, separation of protection critical from non-protection-critical execution domains, minimality (exclusion from the trusted computing base (TCB) of modules that are not protection-critical), layered architecture, abstraction, information hiding, consistency with formal model and DTLS

The complexity of the conceptual discretionary access control (DAC) policy requirements as well as that of the enforcement mechanism make it difficult to provide *assured* DAC in a DBMS. Further, under current evaluation interpretations, an attempt to incorporate a semantically rich (or of a complex) DAC policy could almost certainly exclude an otherwise satisfactory security architecture from consideration as a B2, B3 or A1 DBMS.

We have chosen to use *complexity* (be it of policy or of implementation) as a fundamental means of differentiating between choices of policy. We find, having conducted a short investigation with vendors of several commercial trusted RDBMSs, that they, too, have found complexity to be a central issue among National Computer Security Center (NCSC)-sanctioned evaluators, even at the C2 and B1 levels.

In our study, we identified three distinct formulations, with increasing richness (and of complexity!), of DBMS DAC policies. The simplest of these, Uninterpreted DAC Views, is our study's baseline policy. Two additional DAC policies, Interpreted Assured Primitive Views (*p*views) and Interpreted Qualified Views (*q*views) are introduced and analyzed. The first two of these are found to be compatible with B2, B3 or A1 assurance requirements. Attaining adequate levels of assurance for the third is well beyond the present state of the art.

A second theme key to our study was to identify the basic principles behind the notion of "balanced assurance" in trusted systems. This provided the focus necessary to identify and analyze candidate

[1]Research presented in this paper was performed by Arca Systems, Inc. under contract to Rome Laboratory through Infosystems Technology, Inc.

D. L. Spooner et al. (eds.), *Database Security IX*

mechanisms that could adequately enforce content- and context-dependent DAC policies in concert with the architecture of a DBMS suited for B2 or higher level of assurance. We based our analysis on first principles. Section 2 provides a discussion of several first principle topics related to requirements, policies, safety, assurance, containers, and objects.

We revisited the relationships between control and assurance over physical objects and over logical objects (objects that are instantiations of some abstraction on the physical objects). Control over access to logical objects depends heavily on the ability to partition individual physical objects such that the derived logical objects are disjoint. Achieving consistent access control over disjoint logical objects is far more well-defined than access control over intersecting logical objects. Although there are effectively no physical objects in a database, there are objects that can be considered to be sufficiently primitive abstractions that they can be treated as if they were physical. These are largely the class of named objects.

- named objects include databases, metadata, tables, view definitions, stored programs.
- logical objects include attributes, tuples, algebraic relational operations, integrity constraints and view instances.

Mandatory and discretionary access control over named objects can be controlled to as high a degree of assurance as can be achieved in a high assurance operating system. Access can also be controlled over certain of the logical objects to a high degree of assurance when the logical objects can be consistently identified and distinguished because of precise semantics and disjointness (*i.e.*, they form a *partition* of the physical/logical object from which they are derived).

However, assurance regarding views is complicated since view instances in a database generally need not be a set of disjoint logical objects, nor need the contents of any view instance remain constant — the contents of view instances may change as a result of database transactions (state changes). Notwithstanding, we find that high assurance access control can be achieved for subclasses of the logical objects and over all of the named objects. This is reflected in our identification in Section 3 of *p*view and, to a lesser extent, in the identification of *q*view.

## 2. First Principles discussions

This section addresses the first principles of specifying, designing, and analyzing high assurance in terms of discretionary access control in DBMS intended to satisfy the B2, B3 or A1 classes of the TCSEC. This section begins with a survey of functional and policy requirements for trusted DBMS. High assurance in a trusted system is based on satisfying theoretical requirements and functional relationships that serve as the basis for proof that an implementation is sufficient to support the policy requirements of the "customer" organization. The principal theoretical concerns are: formal policy formulation; formal policy consistency and completeness; and, properties of a policy, including safety, effectiveness (unambiguity and adequacy), and implementability (sufficiency and correctness).

### 2.1.Why There is a Problem

The view has long been considered to provide a viable basis for controlling access to databases.

The concept of secure views originated in IBM's System R DBMS (now SQL/DS), which was inspired by [3]'s seminal work on relational databases. System R introduced a view as a stored or derived relation expressed in SEQUEL. It tied its access control mechanism to views by making views the objects of authorization. The rationale was that views, being at a higher level of abstraction than the physical data, simplify specification and enforcement of context- and content-dependent constraints. For the same reason, Stonebraker adopted a high-level approach in INGRES [14], though their strategy uses query modification rather than views *per se*.

Proposals to use secure views as a basis for multilevel secure database systems were independently made by [2] and by Denning and Neumann [1] at the 1982 Air Force Summer Study on Multilevel Database Management Security. They observed that because views can define arbitrary sets of stored and derived data, views could provide a means of addressing the problems of context- and content-dependent classification, inference, aggregation, and sanitization on a dynamic basis. Denning and Neumann chaired a study group that discussed the benefits and issues associated with classifying views, concluding that the benefits justified further research, but that there were many open problems and issues. Correctness assurance was not addressed in their investigation, however.

The following is a possible decomposition that builds on existing [security] kernel technology. An alternative strategy is to design a relational database kernel that merges layers 1 and 2. The layers are listed from top to bottom, with the boundaries for the TCB and reference monitor as shown: Layer 5

enforces discretionary security for access views, *e.g.*, access-control lists or user/group/world vectors, as desired. Although Layer 5 cannot compromise the *mandatory* secrecy and integrity provided by Layer 4, its *discretionary* controls are still a part of the TCB. Everything above Layer 5 is untrusted in the sense of the TCB, although user application environments at layer 7 may themselves enforce additional policies [4].

| Layer | Function |
|---|---|
| 1 | User Interface and Presentation View Manager |
| 2 | Database Management Functions |
| **TCB Boundary** | |
| 3 | Discretionary View Manager |
| 4 | Mandatory View Manager |
| 5 | Relational Operators |
| **O/S Reference Monitor** | |
| 6 | Element Manager |
| 7 | Security Kernel |

Beginning with the B2 level, the TCSEC's System Architecture requirements mandate that the TCB

> ...shall be internally structured into well-defined largely independent modules. It shall make effective use of available hardware to separate those elements that are protection-critical from those that are not. The TCB modules shall be designed such that the principle of least privilege is enforced....

At the B3 level, the requirement is modified so that

> ...The TCB shall be designed and structured to use a complete, conceptually simple protection mechanism with precisely defined semantics. The mechanism shall play a central rôle in enforcing the internal structuring of the TCB and the system. The TCB shall incorporate significant use of layering, abstraction and data hiding. Significant system engineering shall be directed toward minimizing the complexity of the TCB and excluding from the TCB modules that are not protection-critical.

If DAC policy were expressed and controlled through relational view definitions and their manipulation, then the correctness of DAC enforcement would depend on all of the mechanism used for instantiating the view on the database. This would necessarily place most of the DBMS inside the TCB boundary, since the SQL engine is needed for this purpose. This necessary mechanism would certainly include the machinery involved in parsing, compiling, optimizing, and executing the relational algebra for queries and updates. Were this done, it is highly likely that the entire DBMS portion of the TCB would execute within a single domain with equal privilege, contrary to the B2 least privilege requirement, and there would be no separation between protection critical and non-protection critical elements.

Most commercial DBMS architectures do not incorporate the internal structure and discipline needed to meet the interpreted B2 modularity requirement. In addition to failing to meet the B3 TCB minimality and simplicity requirements, if the entire SQL engine were included within the TCB boundary, it is improbable that such an architecture could meet the additional B3 requirements to implement layering, abstraction and data hiding, particularly since such structuring would likely add considerable execution overhead to the product.

### 2.2. Policy, Assurance and Effectiveness

The TCSEC considers the system access control policy and its model as being integral to the overall assurance that can be provided by an implementation of a trusted system. Other trusted system criteria, particularly the ITSEC and Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), consider *effectiveness assurance* as well as *correctness assurance* aspects of policy and its implementation.

By its very nature, a system access control policy implements a subset of an enterprise's security policy requirements. The remainder of the enterprise's security policy (*e.g.*, physical protection of data, control of printed output, etc.) is typically enforced by procedural, rather than automated, means.

The *effectiveness* of an ITSEC Target of Evaluation relates to how well it provides security in the context of its actual or proposed operational use; its correctness assurance relates to a property of the Target of Evaluation (the ITSEC analogue of a Trusted Computing Base) such that it accurately reflects the stated security target for that system or product.

It is noteworthy that the ITSEC's division between effectiveness and correctness places many implementation considerations for TCSEC B2+ requirements (*e.g.*, TCB minimality, structure, modularity, information handling, etc.) into the category of *correctness* rather than *effectiveness*. It is also noteworthy that ITSEC effectiveness requirements bring together policy and implementation considerations as part of the architectural abstraction. We believe this is sound reasoning, as it entertains the question, under effectiveness assurance, of whether there exists *any* possible implementation of the policy that would enforce the security objectives of the enterprise; whilst correctness assurance issues pertain directly to the properties of one, specific, TOE instantiation.

## 2.3. The Safety Problem and DAC

To many INFOSEC professionals, *assured Discretionary Access Control* is an oxymoron. Either because of the frequent lack of a provable safety property, or because of the absence of a sound counter to the "Fundamental Flaw of DAC", discretionary access control is frequently treated as though it were an unimportant, second-class access control policy feature. The principal policy-deficiency issues are cogently summarized by [10]:

> In its most basic form, the safety question [7] for access control asks: Is there a reachable state in which a particular subject possesses a particular privilege for a specific object? It is the fundamental question which an access control model must confront. Since subjects are usually authorized to create new subjects and objects, the system is unbounded; and it is not certain that such analysis will be decidable, let alone tractable, without sacrificing generality.
>
> There is an essential conflict between the expressive power of an access control model and the tractability of safety analysis. The access matrix model, as formalized by Harrison, Ruzzo, and Ullman, (7) has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties. Safety is undecidable for most policies of practical interest, even in the monotonic version of HRU (which only allows revocation which is itself reversible).
>
> The safety problem is closely related to the so-called fundamental flaw of discretionary access control. DAC is vulnerable to Trojan horses, partly because Trojan horse laden programs can surreptitiously modify the protection state without explicit instruction from the users. These Trojan horses are, however, constrained by the authorization scheme. They can modify the protection state only by using commands which are authorized in the current state. Consequently, the Trojan horse vulnerability of DAC requires that we assume the worst case regarding propagation of access rights in a system....

To most organizations, individual- or rôle-based access control is essential because of specific need-to-know requirements under which the enterprise is bound. Sandhu has produced a series of papers that explore the underlying theoretical issues of safety and the degree to which safety properties can be proven in different policy model formulations.

The safety problem and the Fundamental Flaw of DAC largely relate to effectiveness of the policy *model*. As long as the policy model provides for *propagation* of access rights and *replication* of protected objects, these two effectiveness issues apply *independent* of the correctness properties of any particular implementation. As a consequence, no matter how structured or modular the mechanism's implementation, no matter how minimal the TCB in which the implementation resides, the policy's basic strengths or weaknesses will influence the overall degree of assurance one can claim relative to the *safety* properties of the resulting system.

The principal theoretical issues relating to the safety question are closely tied to abstract (and real) systems whose policies permit the *propagation of rights* to objects and that permit the unconstrained *replication or modification of persistent objects*. In particular, it is generally undecidable whether a system policy is safe if:

- Software $S$ acting in the name of an authorized user $u$ having access to an object $o$ may confer or revoke all possible subsets of those access rights to any other user $v$. The issue is a Trojan horse issue, as $S$ may be capable of conferring access rights possessed by $u$ to $v$ without the knowledge and informed consent of $u$.
- Software $S$ acting in the name of an authorized user $u$ having access to an object $o$ may copy data into a new or existing object $o_1$ in the persistent store such that $o_1$ is accessible to some user $v$ who

lacks the access to $o$. This issue, also, is a Trojan horse issue, as $S$ may be capable of conferring indirect access rights possessed by $u$ to $v$ without the knowledge and informed consent of $u$.

- Software $S$ acting in the name of an authorized user $u$ having access to an object $o$ may modify data in an existing object $o_1$ in the persistent store such that $o_1$ is modified on behalf of some user $v$ who lacks the access to $o_1$. This issue, also, is a Trojan horse issue, as $S$ may be capable of conferring indirect access rights possessed by $u$ to $v$ without the knowledge and informed consent of $u$.

The theoretical safety concerns are present as an artifact of the abstract access control model and are independent of all implementation-specific (*i.e.*, of all correctness assurance) issues.

*If propagation or replication, as discussed above, is part of the discretionary access control policy, then there is a safety issue that pertains uniquely to effectiveness assurance.*

### 2.4.DBMS Functionality of the Safety Problem

Policy effectiveness issues may arise because of differences between the more precisely understood operating system container-based access control policies and the more subtle issues of content- and context-based access and access control. Indeed, correctness assurance is concerned with the link between abstract policy requirements and the sufficiency of an implementation to carry these out.

There will be surprises. Access through views is preferred in many real applications over granting users direct access to the base relations. Access through views is a means of ensuring that the conditions of access or of update to the database be controlled in ways that relate to the content of the 'relevant' and 'important' elements and relationships within the database. These elements and relationships are determined by a person assigned administrative duties over all or a portion of the database. Consequently, it is expected that users will, in general, be able to query or update data through a view that they cannot access directly in the base relations — indeed, `select` or `update` access to the base tables may be explicitly prohibited to these users.

In some cases, analysis may show that an individual user is on an exclusionary list (NACL) for selecting on a specific base relation, while the user has select access to one or more views that permit select and update on a subset of a base relation or set of base relations. Since different named views may overlap, access to several named views may together result in the user being able to select or update on the *entire* relation. The above view properties hold independent of the means by which views and their interpretation/enforcement have been implemented in the DBMS architecture.

This gives richness to the meaning of discretionary access control and its relationship to a provable 'safety' property. In a container-based system, presence of an ACL is taken to mean that a user may obtain *direct* access to a container object if, subject to the constraints on modes imposed by the MAC policy, the (userID, mode) pair appears on the container object's ACL.

However, the introduction of content-based addressing characteristics of RDBMS leads to the possibility of an individual user having numerous means of addressing subsets, $S$ of a database through the first-order complete query language. If each of these means is achieved through an accorded view, then to claim that a user cannot *directly* view a given subset of a table is to claim that the user does not have access to any view whose defining `where` clause identifies all or part of $S$. It is a mathematically hard problem to determine whether this is the case. Hence, the strongest claim one could make about safety for DBMS access is probably of the form: a user may select (update) a portion $S$ of a base relation if the user is authorized to some set of views $V$ whose `where` predicate evaluates to a set that includes $S$ and the user has select (update) rights to $V$. This property probably does not have an only-if counterpart.

Note that this characterization of $V$ and $S$ is not very useful, and the restriction is not very satisfying. This property is independent of layering or modularity. It is much closer to trying to say something about a dynamically extensible TCB, since each DBA may add a new view to the system at any time, and this new view immediately changes the access properties of the set of users who have access to the new view. It is a mathematically hard problem to identify the range of each view, even though the domain and the selection predicate are both well-defined.

### 2.5. Unimplementable Access Control Policies

It is very important to understand that effectiveness assurance should address the issue of whether a given access control policy can be implemented at all, whether parts of the policy contradict or are in conflict with other parts of the policy, or if portions of the policy cannot be implemented.

In some cases, the specific formulation of the policy will determine the extent to which the expectations of the enterprise can be enforced. For example, if a policy is expressed to prohibit

unauthorized users from obtaining access to the information contained in a file $F$, then the policy may not be enforceable, since (a) two users may independently conceive and place the same *information* in two separate files, $F$ and $F_i$; (b) an authorized user may deliberately copy the information from $F$ to $F_i$; or (c) a Trojan horse may copy the information from $F$ to $F_i$

If the policy formulation were restated to prohibit unauthorized users from obtaining *direct* access to the *information* contained in a file $F$ (*i.e.,* by accessing the information as a direct effect of accessing the contents of the file $F$) then it *may* be possible to find an enforceable implementation of the policy. (The principal semantic issue is deciding whether two separate representations of the same *information* represent the *controlled* information — 'Jones speaks Pig Latin' contains the same information as 'Onesjay eakspay Igpay Atinlay', as does '4A6F 6E65 7320 7365 616B 7320 5069 6720 4C61 7469 6E').

There *are* accepted implementations of the policy formulation to prohibit unauthorized users from obtaining *direct* access to the *data* contained in a file $F$ (*i.e.,* by controlling accessing the contents of the file $F$).

Not all formally expressible access control policies can be implemented on any system. For example, Denning and Schlörer demonstrated the infeasibility of concealing specific numerical data associations in DBMS that permit statistical queries on that data — *i.e.,* if a census database pairs information about individual citizens with personal salary data that must not be disclosed under a privacy policy, but the policy permits users to ask for statistical information about salaries of sets containing $k>1$ individuals, then an attack always exists from which to derive the salaries of individuals if a user can issue an unbounded number of queries.

## 2.6. Summary of High Assurance DAC for *Basic Database Objects*

This section discusses discretionary access controls that may be applied to basic database objects independent of their internal contents or the context of the operation. A form of assurance comparable to operating system security may be provided for such objects. This form of content- and context-independent discretionary access control ($C^2$IDAC) may interchangeably be provided either by the underlying trusted operating system or by the trusted database management system.

These particular objects and access modes are interpreted in the coarsest sense of *container* and involve *minimal* use or dependency on the DBMS compiler. A protected "translation table" could map between the "database modes" and those recognized by the operating system (*e.g.,* `select` may be mapped into the operating system's read operation; `insert` into append, etc.).

Many DBMS objects can be covered by a form of high assurance $C^2$IDAC. That is, if the object is protected by the operating system, then the database object is considered as an operating system *named object* (*e.g.,* a file or program), and the object is protected with the operating system security policy model's semantics (*i.e.,* the protection is expressed in terms of operations like read, write, delete, append, concatenate, execute, etc.). Such a named object may be any of the following: database, metadata, table, view definition,[2] column or row.[3]

If the protection is provided by the trusted database management system, the controlled modes of access can be considered as relatively "coarse modes" applying to the *entire* object such as `create, select, update, alter, insert, delete, drop, reference, use, execute.` The typical trusted operating system's `set_acl` operation lacks the richness of SQL's approximately corresponding grant/revoke operator, and generally does not address the `with grant` or `cascade`, and these additional controls would necessarily have to be provided by the trusted DBMS.

Here, it is important to stress that there is *no* forced connection in the implementation between a view (named) and those tables whose names or attributes may appear as character strings within the view definition's select clause or where clause. *I.e.,* from the operating system perspective, if *foo* is a table and $u$ a user, if $u$ is on an ACL with mode $m$ then $u$ may (subject to MAC policy constraints) access *foo* in mode $m$ independent of rights $u$ may have (or be prohibited from having) in any named views.

---

[2]Considered as an uninterpreted container, and *not* in terms of its semantic content as, e.g., an SQL expression.

[3]If the row is stored by itself as an operating system file.

## 2.7. Assurance Hierarchies

It is well-recognized in the literature that some enforcement mechanisms are more effective than others and that some forms of assurance are more significant than others. The following discussion assumes the existence of an effective means of encapsulating protected objects so that mediation will be performed by the TCB on each data access request.

Label-based access control policies like MAC are usually stronger than policies with richer semantics and more flexible rules (*e.g.*, DAC) since it easier to prove "safety" properties of label-based policies (*i.e.*, data bearing a label *l* will only be visible to subjects properly associated with *l*) than to prove similar properties for DAC (*i.e.*, only users who have been granted read access to *o* may read the information in *o*). This is generally true because:

- MAC policies generally impose severe constraints on the set of users and subjects permitted to change the label of an object or of a subject
- DAC policies are generally very permissive in the ways they allow users and subjects to modify the access permissions on an object and the way subjects may accord access permissions to other users or subjects
- MAC policies generally call only for a simple comparison between the labels associated with the object and the subject in order to determine if a given access mode is to be permitted
- MAC policies may call only for a simple comparison to determine if a user or subject has been granted a specific mode of access to an object
- it is generally not possible to prove a "safety" property for complex DAC policies
- DAC policies may call for potentially complex evaluation of a logical predicate or of a program in order to determine whether a given access mode is to be permitted

Some MAC policies may derive an object's label via a complex dynamic function of the object's (and possibly other objects') contents. It may not be possible to prove a safety policy for such policies.

On the other hand, some forms of DAC may have provable "safety" properties, and these can be implemented with no less assurance than simple forms of MAC. [Sandhu 1992] characterizes these DAC policies as :

- very restrictively constraining who may grant or revoke access to *any* object
- very restrictively constraining to whom access may be granted, based on subject's current context and other considerations
- controlling access to the *container* of data rather than to the *contents* of the object

The highest forms of assurance are normally associated with implementations of policies that have highly constrained rules for propagating access privileges and that determine access rights independent of the content of the object or the context of the access operation. Essentially, such policies are the simple forms of MAC and content-independent DAC. All other forms of access control policy are richer and more complex, and present difficulties in proving an appropriate "safety" theorem.

In this sense, it is possible to order the potential assurance of an implementation of a policy in terms of the *effectiveness* of the abstract policy, where effectiveness is based on the potential for proving a "safety" theorem for the policy. The most effective policies can be ordered:

1 static label-based MAC policy on the container of information
2 DAC policy on the container of information with limited propagation of rights
3 dynamic[4] label-based MAC policy on the container of information
4 DAC policy on the container of information with arbitrary propagation of rights
5 $C^2$IDAC MAC or DAC with limited set of users authorized to define rules and controls
6 $C^2$IDAC MAC or DAC with unlimited set of users authorized to define rules and controls

The first four of the above classes are often supported by trusted operating system TCBs at all levels of the TCSEC. The implementation of DAC policy is usually made to depend partially on the correctness of the implementation of the underlying MAC policy. In essence, if a MAC test would deny access, the request is not even considered under the DAC policy.

This structuring is sometimes explained in terms of potential damage that could result from an error in implementation: many consider it more serious for an unauthorized disclosure compromise to occur between different classification levels than within a single level. However, a more fundamental

[4]Policies of this form permit modification to the label of an object (e.g., downgrade operations). In many TCB implementations, the dynamic policy mechanism is implemented as a "trusted subject" — a part of the TCB that executes as a subject under control of the static MAC portion of the TCB.

rationale is to note that one should obtain a higher degree of assurance in an implementation if the foundations on which it depends are sound; in terms of policy, one should obtain a higher degree of assurance if a not-provably-"safe" policy is implemented to depend only on a "safe" policy foundation rather than conversely.

It is clear that the content-dependent policies can be partly decided in terms of content-independent policies: if access to the container should always be denied to a given subject, then access to all or part of the contents of the container will necessarily be denied. Hence, there is a logical case for making the more complex content-dependent policies subservient to the content-independent container-based policies.

Because of the granularity of most DBMS objects, it is likely that many distinct objects such as tuples or columns will be stored in one operating system object (such as a file). Hence, unless all of the objects stored in a single operating system object have *identical* access properties (*i.e.*, classification label *and* DAC constraints), a form of content-based access control will be required. This is because the DBMS TCB must be capable of correctly identifying and distinguishing between individual objects within a larger data structure[5] to which the access controls are to be applied prior to mediating the access request.

Note that content-based access controls may be nested and are not necessarily dependent only on the correctness of the DBMS compiler or optimizer. That is, a set of tuples may be stored in an operating system file. A form of content-based access control is required just to identify individual tuples. If an additional form of access control is to be enforced on the tuples (*e.g.*, a view that projects out a specific set of attributes), additional parsing based on the structure of the content of each tuple must be performed, and the correctness of the view depends in part on the correctness of the identification of each tuple. Similarly, views based on a predicate involving several attributes of the same or different tuples depend on several other correct identifications or parsings at the data storage and representation level. Although this observation may appear to be gratuitous, note that most commercial DBMSs are designed to be portable, so the correctness of portable algorithms is dependent on the ability to accommodate data-storage independent representation.

If the foregoing is not implemented correctly, it appears clear that the effectiveness assurance associated with $C^2$IDAC controls is questionable. If they are correct, then the correctness of translation of the access control predicates is itself a security consideration.

## 2.8. Abstraction, Containers and Contents of Containers

It is clear that one person's container is another person's content. This is pointed out in terms of a DBMS implemented on an operating system such that the DBMS containers and their security attributes are represented as contents of an operating system object. The same applies to an operating system object defined by bits that are part of the contents of a disk. One could extend this concept to views and think of the actual computed instance of a view as a container. However, there are obvious differences here: (a) the view is computed from objects that are directly accessible by users, whereas (with proper encapsulation) neither the operating system objects used to store the database nor the disk sectors used to store the files should be; (b) views are user defined as opposed to being defined by the system; and (c) the computation of view instances is typically much more complex than the other instances. Clearly it is necessary to show that the "right" data is returned as the result of executing a view definition. This will be very difficult and it will be hard to achieve high assurance for a general view-based DAC mechanism.

However, there is a range of other restrictions on what is accessed by a view (and by whom) which then forms the basis of range of policies of different complexity. At one end of the spectrum is the requirement that only the data described by the view definition is returned. At the other extreme is an ACL on the view definition that just controls who can execute the named view without any policy element regarding what the view accesses. In between one could state more or less about what the view is allowed to access and state this in a form separate from the view definition itself. For example, one could keep a list for a given view definition of which users were allowed to access which tables in which modes through that view. One could even have a simple vocabulary for stating a very

[5]Even with respect to heterogeneous storage of classification-labeled tuples, MAC assurance is complicated because of the need to assure that the label is correctly identified and interpreted, and that all labels are protected from being overwritten by untrusted code.

constrained set of restrictions based on the content of the data. These restrictions would be enforced prior to making the underlying tables available for the engine computing an instance of the view for a particular user. The question in all these is determining policies that are useful, that are simple enough to be subject to high assurance implementation, *and* (if one cares about relevance) where there would be some benefit in having high correctness assurance of the policy's enforcement.

One may try to build on the security-related differences between a physical container and a virtual one. This is particularly significant since even files are just a virtual partitioning of a physical disk's surfaces (even though the interpretation of the software-controlled slot:sector mappings are the responsibility of hardware). The abstraction is accepted as a *physical* encapsulation because (a) accesses to the mapping and data are restricted to those primitive operating system procedures that can modify or directly use the mappings; (b) addressing/accessing the file is based on external descriptors of the container's (the file's) location and extent rather than only to data within the file that identifies it as a file. Not only is associative memory access to files more complex than address-based addressing, but it is accepted as an intuitive fact by most people that ISAM and VSAM methods *are* more complex and are likely to be done in software.

This is another key part of the DAC-view problem. If a view were based on simple (or *external*) relationships involving just the primary key, then a DBMS could easily build a [very large] physical addressing table linking primary keys to the address and physical extent of the tuple on disk. View definition would be based on the values of specific primary keys, and these could be compared to scalars, just as filenames get compared in directory searches. Although this is not a very flexible form of view, it is of no greater computational complexity than the mediation of operating system file mediation. More importantly, it could be done by a mechanism that is bound to the *fetch* rather than to the *manipulations* that may have been performed by the compiler and optimizer.[6] The primary-key mechanism could even be used to support views and joins without significant loss of 'assurance' or 'effectiveness'.

This would work because of the uniqueness and functional dependence characteristics of primary key. By definition, the primary key uniquely determines the tuple, so there would be a bijection between these value-based tuples (virtual files) and their descriptors and their bindings to ACLs and locations.

Interestingly, one might ask about the possibility of indexing relations and generalizing the mechanism to full views via the use of many of these DBMS value physical address⇒ACL bindings. The problems are greatly complicated at this point. The above bijection is lost, since arbitrary fields in a tuple may indeed hold duplicate values in some other tuples. Updates add complexity to maintenance of the address associations that define views, and portions of some tuples may be involved in very different views. So complexity for full associative addressing (*i.e.,* full content-based views) necessarily increases very rapidly and the potential is increased for a user to modify or read a forbidden view because of this kind of overlap.

We believe that the above discussion suggests a form of DAC view management that can control access to the structurally-distinct elements of tables. It also helps to establish insight on the limits for how far the content-*cum*-container analogy could be extended with assurance or effectiveness. This limit is linked to the ability to establish a persistent and tuple-unique tag that can be interpreted by a fairly primitive [perhaps even schema-independent] portion of a DBMS TCB.

Now, the simple form of DAC management described above could be added to the layer of the DBMS TCB that manages coarse forms of container-based access control (*e.g.,* DATABASE or TABLE). The extension could be performed in much the same way as the TCB extension proposed in [12]. The approach is the basis used in this paper for identifying and defining the architecture on which the three policy mechanisms, described in Section 3, are based.

---

[6]There is a minor problem of faithfully entering the view definition into the system and of potential interference by the untrusted SQL compiler. But this is analogous to an operating system's untrusted command line interpreter being used to set DAC for file A when the user specifies it for another file B, over which he holds appropriate Control access. This could either be treated with the same interpretation of the TCSEC as is used for operating system DAC and be ignored, or a trusted path mechanism could be employed for defining views. Overlaps of these views should pose no more of a problem than overlap of the sets $b_1$ and $b_2$ of objects open under Bell-LaPadula of two different subjects $s_1$ and $s_2$.

### 2.9. Disjoint *vs.* Overlapping Objects

Base tables *are* disjoint by definition. Views are not only *not* disjoint in general, but it can be mathematically hard to determine by inspection if their instances do or do not intersect. Further, a pair of view instances may be disjoint at time $t_i$ and they may intersect (or fully coincide) at time $t_{i+1}$.

However, it is clear that at *fetch* time (at a very low level of abstraction in the DBMS engine) it is possible to compare individual attributes in individual tuples of a base table. This is because the low-level engine is capable of identifying each attribute of a tuple in terms of its data type (extracted from the schema) and from implementation detail about its displacement and extent from the base address of the tuple.

Thus, if a view is defined on a single table and includes the primary key, it is possible for the DAC enforcement layer of the TCB to evaluate projections of expressions on table $\boldsymbol{R}$ of the form

$$\boldsymbol{R}.attr_i \ \textbf{<relOp>}\ \boldsymbol{R}.attr_j$$

that correspond to view where clause expressions extracted from the definition of view $V$ of the form

$$V.attr_i \ \textbf{<relOp>}\ V.attr_j$$

where **<relOp>** is any of the standard comparison operators (*e.g.*, $\leq < = \neq > \geq \in \notin \neg$). This enforcement can be performed directly from the schema and from definitions of the view that could be accessible in a canonical form to the DBMS TCB in an association with named users. The DAC view instantiation mechanism would not need to rely on or be based on the SQL compiler or any of its supporting mechanisms. Indeed, even were the query to be modified by the untrusted parser, compiler, optimizer, or SQL engine, the test could still be applied to ensure that the only data extracted would be data to which the user possessed a valid `select`, `delete`, `update`, or `insert` right.

We do not believe that high assurance can be extended to the use of aggregate functions or `group_by` on the table $\boldsymbol{R}$. We believe that it may be possible to extend the definition to those elements of a view taken from a join of two separate tables $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ providing that elements of the two tables are in a 1:1 relationship. These concepts are the basis for the definition of *p*views.

### 2.10. Balanced Assurance

Policy and security architecture are inextricably linked. As a first order term in determining architecture, one needs to determine what is included in the TCB, and the TCB has to implement a sufficient mechanism to enforce the policy. This places a lower bound on the complexity of the TCB. Thus, while different policies can be ranked in terms of effectiveness, they may also be ranked in terms of "verifiability" or potential assurance based on the inherent complexity of the policies. This ranking may be identical to the one based on effectiveness because effectiveness is, in part, based on the degree of difficulty of proving safety properties and depends on the policy's complexity. Yet, were one to rank the policies in terms of *flexibility*, an attribute that may be important to usefulness to the enterprise in its applications, one would probably get the reverse ranking. This has led to the claim that a full-featured trusted DBMS cannot be evaluated above the B1 level because of the supposed need to include the SQL compiler and SQL engine in the TCB.

The original notion of balanced assurance was introduced by the SeaView project [9] extending ideas in [12] and [13]. It provided "a methodology for achieving high level (Class A1) of assurance for a system as a whole by applying high assurance techniques to the part of the system enforcing the mandatory access control policy while requiring assurance measures equivalent to C2 for those parts of the TCB enforcing non-mandatory access control policy." [8]

Many proponents of balanced assurance have based their argument on the basic fact that any implementation in which the DAC enforcement mechanism is *fully* constrained by the underlying MAC enforcement mechanism cannot, by its very nature, violate the MAC policy and lead to a compromise of the MAC policy's safety properties. This argument can readily be shown to be sound relative to *all* of the system's lattice-based confinement and non-interference properties: if the DAC enforcement mechanism is fully constrained by an adequate MAC enforcement mechanism, neither overt nor covert channels can obtain in a B2 or higher security architecture.

It is clear that an ineffective policy can be implemented soundly (correctly and in a form that can be assessed by a certifier or evaluator). It is also clear that an effective policy can be implemented unsoundly (with errors, poor structure, no information hiding, unsound coding discipline, etc.). Many Balanced Assurance advocates have taken the position that, because of the lack of a DAC safety

property, no correctness assurance requirements beyond those of C2 should be levied on the portion a TCB that enforces DAC.

However, the TDI did not embrace balanced assurance, concluding for hierarchical composition of TCB subsets that the composed TCB could not receive a rating [16].

## 3. Analysis and Architectures

In this section we present analysis based on the surveys and first principle discussions in the previous sections.

A policy cannot be considered to be effective if it cannot be enforced in such a way as to meet the expectations of the enterprise. Hence, prior to considering the issues and merits of any particular security architecture, one must determine whether or not there exists at least one possible way of implementing the policy such that that implementation would enforce the policy.

Hence, it is important to determine whether or not the policy statement itself is capable of being consistently and correctly enforced. Not all policies can be (*e.g.*, unsafe policies cannot be enforced). Even in the absence of propagation and replication, some policies cannot be consistently enforced (*e.g.*, a policy that restricts releasing the identity of AIDS patients but makes the records of philosophers public may not be able to protect the identities of all AIDS patients — in particular, nondisclosure of the records of known philosophers could suggest that they are AIDS patients).

*An inconsistent policy cannot be consistently enforced. An access control policy cannot be consistently enforced on overlapping abstractions that are covered by different rules.*

Hence, if a policy is both safe and effective, there is a possible consistent implementation of the policy. Agreeing with [4], we conclude that policy over persistent base containers (*i.e.*, those Real Named Objects in the table below) is always achievable to a high degree of assurance.

| Real Named Objects |
|---|
| database |
| metadata |
| base tables |
| primitive view (*p*view) definitions |
| view definitions |
| stored programs |
| triggers |

| Virtual Name Objects |
|---|
| projection (=columns) |
| select (based on specific values of field(s) in tuple including primary key |
| tuple (select *) when entity integrity is enforced |
| primitive view (*p*view) instantiations |
| qualified view (*q*view) instantiations |

**Named Objects for Assured DAC**

DAC safety concerns can be eliminated as an issue as follows:

- If a system discretionary access control policy prohibits the free-creation and the free-modification of persistent objects[7], then the Fundamental Flaw of DAC is eliminated.
- If rights assignment actions by untrusted software is limited by the system discretionary access control policy limits (*e.g.*, as is done in MVS/RACF, or via a trusted path to the user), the classical HRU safety problem is also eliminated from DAC enforcement.

If there were a safe implementation of DAC under the above constraints, then it is possible to consider other issues relative to effectiveness assurance and correctness assurance. In particular, if the discretionary access control policy formulation is internally consistent, and if the objects to which the discretionary access control policy is applied are disjoint, then an effective implementation is possible.

[7]The free-creation of objects is the ability of a subject to create an object that bears unconstrained access control attributes (e.g., an arbitrary ACL). Free-modification of an object is the ability to modify the contents of an object without modifying its access control attributes. In some access control models (e.g., Walter, Rounds, et al. [17], [6], and [McCollum, et al., 1990]), the ACL of a new object is defined to be the intersection of the ACLs of all objects that had been observed by the subject *s* during *s*'s lifetime, and the ACL on an existing object is modified to be the intersection of the ACLs of all objects that had been observed by the subject *s* during *s*'s lifetime. These policies are safe with respect to Trojan horse attacks.

Under the above constraints, an assured correct implementation can be considered and a strategy is discussed below for achieving it.

Three degrees of assured DAC policy can be envisioned and are next described. These are Uninterpreted DAC Views (*u*views), Interpreted Assured Primitive Views *p*views), and Interpreted Qualified Views (*q*views). Their essential characteristics are summarized as follows:

**Uninterpreted DAC Views:** A simple family of **high** assurance DAC policies can be implemented that control access strictly on the basis of ACLs for subjects and their access modes to any combination of the following disjoint Real Named Objects: Database, Metadata, Base Tables, View Definitions, Stored Programs and Triggers. The propagation of access modes to Uninterpreted DAC Views can be assured to be in compliance with the ANSI SQL standard. The semantic correctness of the access control check can be assured to the same level as is DAC to operating system objects. *No* enforced semantics are assigned, in the assurances associated with this family of policies, to the mapping between specific view definitions and their instantiations as qualified views (*q*views). Semantics of creating a view definition *V* can be implemented to a high degree of correctness to ensure that the creator of *V* possess the appropriate mode of access to each real named object referenced in *V*. If untrusted software is granted privilege to perform propagation of access rights or to freely-create/modify objects, this family of DAC policies is *not* safe.

**Interpreted Assured Primitive Views (*p*views):** A family of **high** assurance DAC policies can be implemented that control access strictly on the basis of ACLs for subjects and their access modes to any combination of the following disjoint Real Named Objects: Database, Metadata, Base Tables, Primitive View Definitions, View Definitions, Stored Programs and Triggers and to the virtual objects formed from Primitive View Definitions and their manipulation. The semantic correctness of the access control check can be assured to the same level as is DAC applied to traditional operating system named objects. The propagation of access modes to *p*views can be assured to be in compliance with the ANSI SQL standard. The correctness of semantic interpretation of *p*view definitions and their instantiation *is* assured, even though no semantics is assigned, in the assurances associated with this family of policies to the mapping between specific view definitions and their instantiations as qualified views (*q*views). Semantics of creating a view definition *V* or a *p*view definition *P* can be implemented to a high degree of correctness to ensure that the creator of *V* possess the appropriate mode of access to each real named object referenced in *V* and additionally that the creator of *P* act in the rôle of DBA or SSO for the Real Named Objects named in *P*. If untrusted software is granted privilege to perform propagation of access rights or to freely-create/modify objects, this family of DAC policies is *not* safe.

**Interpreted Qualified Views (*q*views):** A family of **low** assurance DAC policies can be implemented that control access on the basis of ACLs for subjects and their access modes to any combination of the following disjoint Real Named Objects: Database, Metadata, Base Tables, View Definitions, Stored Programs and Triggers, and on the resulting qualified view instantiations (*q*views). The propagation of access modes to *q*views can be assured to be in compliance with the ANSI SQL standard. The semantic correctness of the model access control check can be assured to the same level as is DAC to traditional operating system named objects. The correctness of semantic interpretation of view definitions *is* assured, and full semantic interpretation is provided for their instantiation as *q*views. Semantics of creating a view definition *V* can be implemented to a high degree of correctness to ensure that the creator of *V* possess the appropriate mode of access to each real named object referenced in *V*. If untrusted software is granted privilege to perform propagation of access rights or to freely-create/modify objects, this family of DAC policies is *not* safe.

These view policies are discussed below.

## 3.1. Uninterpreted DAC Views

Uninterpreted DAC Views is the access control policy implemented for the majority of trusted DBMS products that have been evaluated by the National Computer Security Center to date. In this family of DAC policies, the three following named object policy rules can be implemented with high assurance:

1 The requesting user may use the view definition's name in queries in accord with the modes for which authorization has been granted.
2 Access to base tables is interpreted by mode with no qualifications on attribute name or contents within base table.

3 Modulo safety concerns, propagation of rights to named objects is assured to comply with the ANSI SQL standard.

All three of these are easily achievable within the current state of the art for the B2, B3 and A1 classes of the TCSEC. Vendors offer standard correctness assurances for their view mechanisms, relying upon only correctness testing by their quality assurance group and by customer feedback to identify flaws in the SQL engine's implementation and of *q*views.

### 3.2. Interpreted Assured Primitive Views (*p*views)

The primitive view (*p*view) is a simplification of the view abstraction. Each named *p*view is defined to relate to precisely one base table. In a notation borrowed from Query By Example [19] with modified semantics to fit our needs, a *p*view definition can be expressed as shown below:

| *Key* | $A_1$ | $A_2$ | $A_3$ | | $A_n$ |
|---|---|---|---|---|---|
| + | $\pm_i e_1$ | $\pm_j e_2$ | $\pm_k e_3$ | ... | $\pm_l e_n$ |

where $e_i \in \{A_i\} \cup$ ScalarConstants

It is possible that algebraic expressions in terms of the $A_i$ and constants would be practical, but this begins to make the checker fairly complex. Rule: If every instance of a non vacuous expression $\pm\, \mathbf{r}_i\, e_i$ is satisfied for the tuple, then the tuple is in the view and for each attribute $A_i$, if +, then attribute $A_i$ is included in the exported tuple instance, if -, then attribute $A_i$ is "projected out" (*i.e.*, not selected) and replaced by a standard default value for $A_i$ (*e.g.*, ***null***) in the exported tuple instance. The exported tuple instance is passed out of the TCB boundary and presented to the untrusted SQL engine for subsequent compiler-dependent processing.

The *p*view instantiation can thereby be expressed as

| *Key* | $a_1$ | $a_2$ | $a_3$ | ... | $a_n$ |
|---|---|---|---|---|---|

where each $a_i$ represents the value of $A_i$ or its default value as derived from the application of the *p*view definition. All applicable *p*view definitions connecting a userID to a base table are applied to produce the *p*view instantiation.

### 3.3. Interpreted Qualified Views (*q*views)

The correctness requirements for showing that *q*views are enforced as part of DAC policy would require showing that *q*view instantiations are produced correctly from their view definitions. This is true for the class of *q*view instantiations that are identical to *p*view instantiations (*i.e.*, when they correspond fully to disjoint *virtual* objects). However, the *q*view is generalized to include aggregate functions and references to multiple relations (joins, group-by) into the actual statement of DAC policy, and are thereby potentially far more complex than *p*views. This added generality would necessitate including the SQL compiler, optimizer and engine completely within the TCB boundary. It is presently beyond the state of the art to argue formally or through deep code analysis for the correctness of so large a mechanism. Additionally assurance requirements for modularity, least privilege, TCB minimality, and information hiding would make it unfeasible to satisfy the TCSEC's security architecture correctness assurance requirements for B2, B3 and A1. For those reasons, *q*views are still considered to be a low assurance DAC policy.

### 3.4. Implications on Security Architecture and Implementation Strategies

The conceptual architecture shown below can be used to implement either of the high assurance DAC architectures (with the *p*view layer deleted in the case of Uninterpreted DAC Views).

| ***LAYERING*** |
|---|
| SQL Compiler (untrusted) |
| SQL Engine (untrusted) |
| ***TCB Boundary*** |
| DAC: *p*view interpretation |
| DAC: real object ACL |
| MAC real objects |
| fetch from base tables |
| ***O S TCB*** |

In the figure, DAC and MAC are interpreted within the TCB at the time of fetch/store from/to base tables. The SQL engine and compiler are permitted to perform full interpretation of any SQL expression on data retrieved from the TCB or that is to be sent to the TCB. Recall that in the Uninterpreted DAC Views policies, the three following named object policy rules can be implemented with high assurance:

1 The requesting user may use the view definition's name in queries in accord with the modes for which authorization has been granted.
2 Access to base tables is interpreted by mode with no qualifications on attribute name or contents within base table.
3 Modulo safety concerns, propagation of rights to named objects is assured to comply with the ANSI SQL standard.

Conceptually, for Uninterpreted DAC Views policies, the DBMS TCB could be invoked following the parse of the SQL query and invocation of the SQL Compiler. This could be forced by placing the SQL Compiler and SQL Engine in a dominance domain separate from the execution domain of the user, and having a small primitive trusted parser in the DBMS TCB. The trusted parser could produce a normalized request string that could be interpreted by the TCB to mediate modal access by the user to each named object in the SQL query. This could be established by recursively searching the following TCB-controlled internal table:

| viewName | userName | authModes | viewOwner | baseObjectName | baseObjectType |
|---|---|---|---|---|---|

The DBMS TCB could retain a small table associated with the user and the specific query for subsequent use prior to passing the query into a restricted domain of execution of the SQL compiler and SQL engine. This table would consist of:

| viewOwner | baseTableName | authorizedMode |
|---|---|---|

Once the query has been compiled, planned and optimized, and the retrieval or update has begun, the DBMS TCB will necessarily be called by the SQL engine in order to access the base tables[8]. This table can be consulted by the TCB prior to accessing data in the base tables to ensure that untrusted SQL processing did not modify the set of named objects and modes named in the validated query.

Conceptually, the Interpreted Assured Primitive Views (*p*view) mechanism is based on the use of two TCB relations (tables). These are shown below.

TCB *p*view-to-user binding (PUB): [userID table name *p*view modes]

| userID | table | name | *p*view | *modes* |
|---|---|---|---|---|

*p*view-to-canonical Form binding (PCB): [*p*view canonical form]

| *p*view | Canonical Form |
|---|---|

Here the PUB is searched for all *p*views to which the userID has access for the named base table in the requested mode. The canonical form of a *p*view is nothing but the low level implementation, derived from the metadata, needed to identify each attribute in the table, associated with the QBE definition of the *p*view. The relevant *p*view definitions are applied, in any sequence, to each retrieved tuple, and the resulting *pview* instantiation is exported from the TCB to the SQL Engine for processing. This process is HRU safe, as the application of any *pview* to a tuple produces a restriction of whatever results from the application of any other *p*view to the same tuple; commutativity and associativity concerns are not relevant to the safety problem under these semantics. Updates are also performed in accord with a set of associated *p*view definitions. (Updatable *p*views have the same semantics as updatable views.) It should be noted that a *q*view may correspond identically to a *p*view, and under this scheme (since the SQL engine is not trusted), double checking would be performed, with the SQL engine performing a set of transformations that should always evaluate to the identity transform on the TCB's exported *p*view instantiations.

[8]Actually, of course, since the MRDBMS' SQL engine is untrusted, the TCB will intervene in order to mediate all access attempts by the untrusted subject to named and storage objects.

## 4. Conclusions and future Work

In this paper, we have investigated high assurance discretionary access control concepts from the first principles perspective and based our analysis on historical principles. We found that the concept of *assurance,* in general, was amenable to analysis once its effectiveness and correctness considerations were separated. This permitted the identification of those properties of a specific DAC policy that were independent of any particular implementation (*e.g.,* safety, internal consistency, etc.) and those that are sensitive to particular implementation strategies ($C^2$IDAC, *p*views, *q*views).

Establishing a baseline with Uninterpreted DAC Views, a particular form of $C^2$IDAC for which a high-assurance implementation is possible, we also identified Interpreted Assured Primitive Views (*p*views) as a content *dependent* policy for which a high-assurance mechanism is possible. Traces of the *p*view concept can be found in the Sybase implementation [15], as well as in paper studies including ASD Views [5] and [18]. A third policy formulation, Interpreted Qualified Views (*q*views), was introduced and analyzed. While the latter is very close to the complete formulation of fully general view mechanisms (and hence strongly related to $C^2$IDAC), we found that *q*views are presently dependent on far more complex implementation mechanisms than can meet the correctness assurance requirements of the TCSEC, even when the *q*views appear to be disjoint.

Additional research is recommended in three separate directions.

- ***p*view generality** — research needs to be conducted to investigate the feasibility of optimizing interaction between the untrusted SQL engine and an assured implementation of *p*view instantiation in order to eliminate redundant operations.
- **other data models** — research needs to be conducted to investigate assurance issues relative to DAC in the Entity-Relation, Extended Relational and Object Oriented data models.
- **audit** — research needs to be conducted to investigate assurance issues of correlating the user query and the executed plan if the compiler is not part of the TCB.

We also suggest that research be conducted on the performance characteristics of *p*views in conjunction with the unconstrained *u*view and *q*view mehanisms. We believe that an efficient mechanism could be produced, possibly in hardware, that could mediate access with relatively little overhead at the time data records are individually fetched from the data store. However, such a mechanism could undermine optimized storage and retrieval strategies and produce overall degradation of retrievals in the large. We do not believe that the compounded effects will be severe of interpretive *q*view evaluation atop [compiled] *p*view mediation, largely because of the simplicity of the *p*view checking algorithms and the speed with which they can conceptually be performed. We would also observe that if the nature of the *p*view constraints is not secret, then an optimizer could eliminate redundancy between what is performed in the *p*view and what would be checked at the *q*view or untrusted *u*view level.

## 5. Acknowledgments

The authors gratefully acknowledge the contribution made by Simon Wiseman in suggesting that we consider effectiveness and correctness aspects of assurance. Special thanks go to Jim O'Connor for his insights into balanced assurance and DAC. We also thank our colleagues Larry Halme, Stan Wisseman and Doug Landoll for their participation in our research. Bill Wilson assisted considerably in our investigation and provoked one of the authors into producing an unsettling *reductio ad absurdum* argument on the utility of *u*views. We appreciate the insights provided by LouAnna Notargiacomo, Mary Denz and Joe Nasevich who participated in reviewing an early draft of this paper and for suggestions made by anonymous referees 4 and 13. Finally, we wish to thank Aryeh Tal-Nir and Rammohan Varadarajan (Informix); Linda Mundy, Don Brinkley and Mark Quinn (Sybase); and Bill Maimone and Richard Allen (Oracle) for the candid insights they provided on their products' policies and mechanisms.

## 6. References

[1] M. Schaefer, Ed., Committee on Multilevel Data Management Security, Multilevel Data Management Security, Technical Report, Air Force Studies Board, National Research Council, National Academy Press, 1983

[2] Claybrook, BAG., "Using Views in a Multilevel Secure Database Management System", *Proceedings of the 1983 Symposium on Security and Privacy,* IEEE Computer Society, April 1983, pp. 4-17.

[3] Codd, E. F., A Relational Model of Data for Large Shared Data Banks, *Communications of the ACM*, June, 1970, pp. 377-387.

[4] Denning, Dorothy E., Schell, Roger R., et al., *Views for Multilevel Database Security*, IEEE Transactions on Software Engineering, 1986.

[5] Garvey C. and Wu, A., ASD-Views, *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April, 1988

[6] Graubart, R., "On the Need for a Third Form of Access Control," *Proceedings of the XIIth National Computer Security Conference,* October 1989, pp. 296-303.

[7] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D., "Protection in Operating Systems," Communications of the ACM 19(8):461-471 (1976).

[8] Irvine, C. E., Schell, R. R., Thompson, M. F., *Using TNI Concepts for the Near Term Use of High Assurance Database Management Systems*, Proceedings of the Fourth Rome Laboratory Multilevel Database Security Workshop, Research Directions in Database Security IV, June 1993.

[9] Lunt, T. F., Schell, R.R., Shockley, W. R., Heckman, M., and Warren, D, A Near-Term Design for the SeaView Multilevel Database System, *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988, pp. 234-244.

[10] Sandhu, R. S., "The Typed Access Matrix Model," *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, May 1992, pp. 122-136.

[11] M. Schaefer, "A Contemporary Survey of Discretionary Access Control Policy Assurance in Commercial Trusted Relational DBMS", in *Database Security, VIII : Status and Prospects,* J Biskup, M. Morgenstern, C.E. Landwehr, Eds., IFIP/North Holland, 1994 p. 376.

[12] Schaefer, Marvin., Schell, Roger R., Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products, *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, May 1994, pp. 41-49.DoD Computer Security Center.

[13] Shockley, W.R., Schell, R.R., TCB Subsets for Incremental Evaluation, *Proceedings of the Third Aerospace Computer Security Conference*, December, 1987, pp. 131-139.

[14] Stonebraker, M. and Wong, E., *Access Control in a Relational Database Management System by Query Modification*, Proceedings 1971 ACM.

[15]*SYBASE Secure SQL Server™ Security Administration Guide*, (Release 10.0), Document ID: 36051-01-1000-01, Change Level: 1, 20 December 1993.

[16] Tinto, M., *The Design and Evaluation of INFOSEC Systems: The Computer Security Contribution to the Composition Discussion*, National Computer Security Center, C Technical Report 32-92, June 1992

[17] Walter, K.G., Ogden, W.F., Rounds, W.C., Bradshaw, F.T., Ames, S.R., and Shumway, D.G., "Primitive Models for Computer Security," Dept. Computing and Information Sciences, Case Western Reserve University, Cleveland, January 1974.

[18] Wilson, Jackson, "Views as the Security Objects in a Multilevel Secure Relational Database Management System," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April, 1988, pp. 70-84

[19] Zloof, M.M., "Query by Example: a data base language," *IBM Systems Journal*, 16:4, 1977, pp. 324-343.